typst

Typst Documentation

Version: 0.15.1

Copyright © 2019 – 2026 Laurenz Mädje, Martin Haug, and The Typst Project Developers.

The contents of this document and the Typst compiler are licensed under the terms of the Apache License, Version 2.0.

This documentation is open source and open for contribution on https://github.com/typst/typst. It has been typeset using Typst 0.15.1, HK Grotesk, and Cascadia Mono.

Published by Typst GmbH, Heidestraße 34, 10557 Berlin, Germany.

https://typst.app/

Contents

1 Overview 9

2 Tutorial 10

Introduction ... 10
2.0.1 When Typst ... 10
2.0.2 Learnings ... 10
2.1 Writing in Typst ... 11
2.1.1 Figure ... 12
2.1.2 Bibliography ... 14
2.1.3 Maths ... 15
2.1.4 Review ... 16
2.2 Formatting ... 17
2.2.1 Set rules ... 17
2.2.2 Autocomplete ... 18
2.2.3 Page Setup ... 18
2.2.4 Sophistication ... 19
2.2.5 Show rules ... 20
2.2.6 Review ... 21
2.3 Advanced Styling ... 22
2.3.1 Guidelines ... 22
2.3.2 Set Rules ... 22
2.3.3 Title And Abstract ... 23
2.3.4 Columns And Headings ... 26
2.3.5 Review ... 30
2.4 Making a Template ... 31
2.4.1 Variables ... 31
2.4.2 Toy Template ... 32
2.4.3 Set And Show Rules ... 33
2.4.4 Named Arguments ... 34
2.4.5 Separate File ... 36
2.4.6 Review ... 37

3 Reference 38

Introduction ... 38
3.0.1 Language ... 38
3.0.2 Library ... 38
3.0.3 Export ... 38
3.0.4 Reading the reference ... 38
3.1 Syntax ... 39
3.1.1 Modes ... 39
3.1.2 Markup ... 39
3.1.3 Math ... 39
3.1.4 Code ... 40
3.1.5 Comments ... 41
3.1.6 Escapes ... 41
3.1.7 Identifiers ... 42
3.2 Styling ... 43
3.2.1 Set rules ... 43
3.2.2 Show rules ... 43
3.3 Scripting ... 46
3.3.1 Expressions ... 46
3.3.2 Blocks ... 46
3.3.3 Bindings ... 46
3.3.4 Conditionals ... 48
3.3.5 Loops ... 48

3.3.6 Fields ... 49
3.3.7 Methods ... 49
3.3.8 Modules ... 50
3.3.9 Packages ... 50
3.3.10 Operators ... 50
3.4 Context ... 52
3.4.1 Style context ... 52
3.4.2 Location context ... 52
3.4.3 Nested contexts ... 54
3.4.4 Compiler iterations ... 54
3.5 Foundations ... 56
3.5.1 Arguments ... 57
3.5.2 Array ... 60
3.5.3 Assert ... 70
3.5.4 Auto ... 72
3.5.5 Boolean ... 73
3.5.6 Bytes ... 74
3.5.7 Calculation ... 76
3.5.8 Content ... 91
3.5.9 Datetime ... 93
3.5.10 Decimal ... 98
3.5.11 Dictionary ... 100
3.5.12 Duration ... 103
3.5.13 Evaluate ... 105
3.5.14 Float ... 106
3.5.15 Function ... 109
3.5.16 Integer ... 112
3.5.17 Label ... 117
3.5.18 Module ... 118
3.5.19 None ... 119
3.5.20 Panic ... 120
3.5.21 Path ... 121
3.5.22 Plugin ... 123
3.5.23 Regex ... 126
3.5.24 Representation ... 127
3.5.25 Selector ... 128
3.5.26 Standard Library ... 131
3.5.27 String ... 132
3.5.28 Symbol ... 139
3.5.29 System ... 140
3.5.30 Target ... 141
3.5.31 Type ... 142
3.5.32 Version ... 143
3.6 Model ... 144
3.6.1 Asset ... 145
3.6.2 Bibliography ... 146
3.6.3 Bullet List ... 153
3.6.4 Cite ... 156
3.6.5 Divider ... 161
3.6.6 Document ... 162
3.6.7 Emphasis ... 165
3.6.8 Figure ... 166
3.6.9 Footnote ... 173
3.6.10 Heading ... 176
3.6.11 Link ... 180
3.6.12 Numbered List ... 183
3.6.13 Numbering ... 187
3.6.14 Outline ... 189

3.6.15 Paragraph ... 195
3.6.16 Paragraph Break ... 203
3.6.17 Quote ... 204
3.6.18 Reference ... 208
3.6.19 Strong Emphasis ... 211
3.6.20 Table ... 212
3.6.21 Term List ... 222
3.6.22 Title ... 224
3.7 Text ... 225
3.7.1 Highlight ... 226
3.7.2 Line Break ... 228
3.7.3 Lorem ... 229
3.7.4 Lowercase ... 230
3.7.5 Overline ... 231
3.7.6 Raw Text / Code ... 233
3.7.7 Small Capitals ... 240
3.7.8 Smartquote ... 241
3.7.9 Strikethrough ... 243
3.7.10 Subscript ... 245
3.7.11 Superscript ... 246
3.7.12 Text ... 248
3.7.13 Underline ... 261
3.7.14 Uppercase ... 263
3.8 Math ... 264
3.8.1 Variables ... 264
3.8.2 Symbols ... 264
3.8.3 Line Breaks ... 264
3.8.4 Function calls ... 264
3.8.5 Alignment ... 265
3.8.6 Math fonts ... 265
3.8.7 Math module ... 265
3.8.8 Accessibility ... 266
3.8.9 Accent ... 267
3.8.10 Attach ... 269
3.8.11 Binomial ... 271
3.8.12 Cancel ... 272
3.8.13 Cases ... 274
3.8.14 Class ... 275
3.8.15 Equation ... 276
3.8.16 Fraction ... 279
3.8.17 Left/Right ... 281
3.8.18 Matrix ... 285
3.8.19 Primes ... 288
3.8.20 Roots ... 289
3.8.21 Sizes ... 290
3.8.22 Stretch ... 292
3.8.23 Styles ... 293
3.8.24 Text Operator ... 294
3.8.25 Under/Over ... 295
3.8.26 Variants ... 299
3.8.27 Vector ... 302
3.9 Symbols ... 304
3.9.1 Shorthands ... 304
3.9.2 General Symbols ... 305
3.9.3 Emoji ... 316
3.10 Layout ... 329
3.10.1 Align ... 330
3.10.2 Alignment ... 332

3.10.3 Angle ... 334
3.10.4 Block ... 335
3.10.5 Box ... 339
3.10.6 Column Break ... 342
3.10.7 Columns ... 343
3.10.8 Direction ... 345
3.10.9 Fraction ... 348
3.10.10 Grid ... 349
3.10.11 Hide ... 361
3.10.12 Layout ... 362
3.10.13 Length ... 364
3.10.14 Measure ... 366
3.10.15 Move ... 368
3.10.16 Padding ... 369
3.10.17 Page ... 371
3.10.18 Page Break ... 381
3.10.19 Place ... 382
3.10.20 Ratio ... 386
3.10.21 Relative Length ... 387
3.10.22 Repeat ... 389
3.10.23 Rotate ... 390
3.10.24 Scale ... 392
3.10.25 Skew ... 394
3.10.26 Spacing (H) ... 396
3.10.27 Spacing (V) ... 398
3.10.28 Stack ... 399
3.11 Visualize ... 400
3.11.1 Accessibility ... 400
3.11.2 Circle ... 401
3.11.3 Color ... 403
3.11.4 Curve ... 417
3.11.5 Ellipse ... 424
3.11.6 Gradient ... 425
3.11.7 Image ... 434
3.11.8 Line ... 438
3.11.9 Polygon ... 440
3.11.10 Rectangle ... 442
3.11.11 Square ... 444
3.11.12 Stroke ... 446
3.11.13 Tiling ... 450
3.12 Introspection ... 454
3.12.1 Counter ... 455
3.12.2 Here ... 460
3.12.3 Locate ... 461
3.12.4 Location ... 462
3.12.5 Metadata ... 464
3.12.6 Query ... 465
3.12.7 State ... 468
3.13 Data Loading ... 473
3.13.1 Encoding ... 473
3.13.2 CBOR ... 474
3.13.3 CSV ... 475
3.13.4 JSON ... 476
3.13.5 Read ... 478
3.13.6 TOML ... 479
3.13.7 XML ... 481
3.13.8 YAML ... 483
3.14 PDF ... 485

3.14.1 Exporting as PDF ... 485
3.14.2 PDF standards ... 485
3.14.3 PDF-specific functionality ... 487
3.14.4 Artifact ... 489
3.14.5 Attach ... 491
3.14.6 Data Cell ... 493
3.14.7 Header Cell ... 495
3.14.8 Table Summary ... 497
3.15 HTML ... 499
3.15.1 Exporting as HTML ... 499
3.15.2 HTML-specific functionality ... 499
3.15.3 Elem ... 501
3.15.4 Frame ... 502
3.15.5 Typed HTML ... 503
3.16 PNG ... 560
3.16.1 Exporting as PNG ... 560
3.17 SVG ... 561
3.17.1 Exporting as SVG ... 561
3.18 Bundle ... 562
3.18.1 Exporting as a bundle ... 562
3.18.2 Creating files ... 562
3.18.3 Introspection ... 563

4 Guides **564**

4.1 Guide for LaTeX Users ... 565
4.1.1 Installation ... 565
4.1.2 Getting Started ... 565
4.1.3 Elements ... 565
4.1.4 Commands ... 567
4.1.5 Templates ... 569
4.1.6 Packages ... 570
4.1.7 Maths ... 571
4.1.8 Latex Look ... 573
4.1.9 Bibliographies ... 573
4.1.10 Limitations ... 573
4.2 Page Setup Guide ... 575
4.2.1 Customize Margins ... 575
4.2.2 Headers And Footers ... 578
4.2.3 Page Numbers ... 579
4.2.4 Columns ... 583
4.2.5 One-off modifications ... 584
4.3 Table Guide ... 586
4.3.1 Basic Tables ... 586
4.3.2 Column Sizes ... 587
4.3.3 Captions And References ... 588
4.3.4 Fills ... 589
4.3.5 Strokes ... 593
4.3.6 Alignment ... 598
4.3.7 Merge Cells ... 600
4.3.8 Rotate Table ... 601
4.3.9 Table Across Pages ... 605
4.3.10 Importing Data ... 606
4.3.11 Table And Grid ... 611
4.4 Accessibility Guide ... 612
4.4.1 Basics ... 612
4.4.2 Maintaining semantics ... 612
4.4.3 Reading order ... 613
4.4.4 Layout containers ... 614

4.4.5 Artifacts 614
4.4.6 Color use and contrast 615
4.4.7 Textual representations 616
4.4.8 Natural Language 618
4.4.9 Document Title and Headings 619
4.4.10 Accessibility Standards and Legislation 620
4.4.11 Testing for Accessibility 621
4.4.12 Limits and considerations for export formats 622

5 Changelog **624**

0.15.1 625
0.15.0 626
0.14.2 638
0.14.1 639
0.14.0 641
0.13.1 649
0.13.0 650
0.12.0 656
0.11.1 662
0.11.0 663
0.10.0 668
0.9.0 670
0.8.0 673
0.7.0 675
0.6.0 678
0.5.0 680
0.4.0 681
0.3.0 683
0.2.0 685
0.1.0 686
Earlier 688

1 Overview

Welcome to Typst's documentation! Typst is a markup-based typesetting system that combines powerful automation and high-quality typography with speed and ease of use. This makes it suitable for documents of any complexity. Typst is a great alternative to both word processors and LaTeX.

This documentation is split into multiple parts, serving different needs:

- If you are new to Typst, we highly recommend starting with our beginner-friendly tutorial[2]. Throughout the tutorial, we will introduce you to Typst through a practical example.
- To answer targeted questions about Typst and familiarize yourself with advanced features, use the reference[3]. It describes the fundamental features of the Typst language and contains sections for all the functions, types, and more that come with Typst.
- For tailored, in-depth how-tos on specific features, use cases, and audiences, check out our guides[4]. They provide copyable snippets throughout and allow you to build confidence with a specific feature area. If you are coming from LaTeX, the Guide for LaTeX Users[4.1] provides an alternative introduction to Typst, building on concepts you already know.

The term *Typst* refers to three concepts: The Typst language, the Typst compiler, and the Typst web app. The language is what you write, the compiler translates files in the Typst language into PDFs, HTML pages, and other formats, and the Typst web app lets you work collaboratively on Typst projects in your browser. The Typst language and the compiler are open-source.

This documentation primarily documents the Typst language, although the tutorial and various pages will refer to the web app and the command line Typst compiler. In the copy of the docs hosted on https://typst.app/docs/[A], we also include documentation about the web app. To learn how to install the Typst compiler CLI, visit the Open Source page[B] on our website. There, you can also learn more about the relationship between the Typst compiler and the web app. Once you have installed the Typst compiler CLI, run `typst help` for more information on how to use it.

Our GitHub repository[C] provides additional developer-facing documentation about how to contribute to Typst and how to integrate it into your applications.

The documentation also contains a changelog[5], in which you can track the evolution of Typst and what changes to the markup language mean for your projects. This documentation applies to Typst 0.15.1.

[A] https://typst.app/docs/
[B] https://typst.app/open-source
[C] https://github.com/typst/typst

2 Tutorial

Welcome to Typst's tutorial! In this tutorial, you will learn how to write and format documents in Typst. We will start with everyday tasks and gradually introduce more advanced features. This tutorial does not assume prior knowledge of Typst, other markup languages, or programming. We do assume that you know how to edit a text file.

The best way to start is to sign up to the Typst app for free and follow along with the steps below. The app gives you instant preview, syntax highlighting and helpful autocompletions. Alternatively, you can follow along in your local text editor with the open-source CLI[A].

2.0.1 When to use Typst

Before we get started, let's check what Typst is and when to use it. Typst is a markup language for typesetting documents. It is designed to be easy to learn, fast, and versatile. Typst takes text files with markup in them and outputs PDFs.

Typst is a good choice for writing any long form text such as essays, articles, scientific papers, books, reports, and homework assignments. Moreover, Typst is a great fit for any documents containing mathematical notation, such as papers in the math, physics, and engineering fields. Finally, due to its strong styling and automation features, it is an excellent choice for any set of documents that share a common style, such as a book series.

2.0.2 What you will learn

This tutorial has four chapters. Each chapter builds on the previous one. Here is what you will learn in each of them:

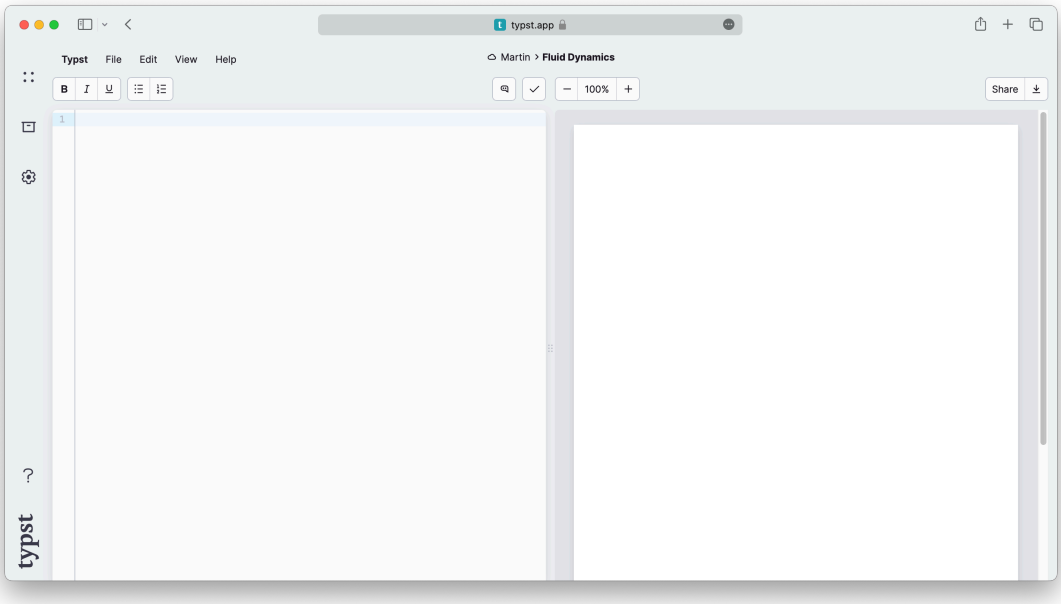
1. Writing in Typst:[2.1] Learn how to write text and insert images, equations, and other elements.
2. Formatting:[2.2] Learn how to adjust the formatting of your document, including font size, heading styles, and more.
3. Advanced Styling:[2.3] Create a complex page layout for a scientific paper with typographic features such as an author list and run-in headings.
4. Making a Template:[2.4] Build a reusable template from the paper you created in the previous chapter.

We hope you'll enjoy Typst!

[A] https://github.com/typst/typst

2.1 Writing in Typst

Let's get started! Suppose you got assigned to write a technical report for university. It will contain prose, maths, headings, and figures. To get started, you create a new project on the Typst app. You'll be taken to the editor where you see two panels: A source panel where you compose your document and a preview panel where you see the rendered document.



You already have a good angle for your report in mind. So let's start by writing the introduction. Enter some text in the editor panel. You'll notice that the text immediately appears on the previewed page.

```
In this report, we will explore the
various factors that influence fluid
dynamics in glaciers and how they
contribute to the formation and
behaviour of these natural structures.
```

In this report, we will explore the various factors that influence fluid dynamics in glaciers and how they contribute to the formation and behaviour of these natural structures.

Throughout this tutorial, we'll show code examples like this one. Just like in the app, the first panel contains markup and the second panel shows a preview. We shrunk the page to fit the examples so you can see what's going on.

The next step is to add a heading and emphasize some text. Typst uses simple markup for the most common formatting tasks. To add a heading, enter the = character and to emphasize some text with italics, enclose it in `_underscores_`.

```
= Introduction
In this report, we will explore the
various factors that influence _fluid
dynamics_ in glaciers and how they
contribute to the formation and
behaviour of these natural structures.
```

Introduction

In this report, we will explore the various factors that influence *fluid dynamics* in glaciers and how they contribute to the formation and behaviour of these natural structures.

That was easy! To add a new paragraph, just add a blank line in between two lines of text. If that paragraph needs a subheading, produce it by typing == instead of =. The number of = characters determines the nesting level of the heading.

Now we want to list a few of the circumstances that influence glacier dynamics. To do that, we use a numbered list. For each item of the list, we type a + character at the beginning of the line. Typst will automatically number the items.

```
+ The climate
+ The topography
+ The geology
```

1. The climate
2. The topography
3. The geology

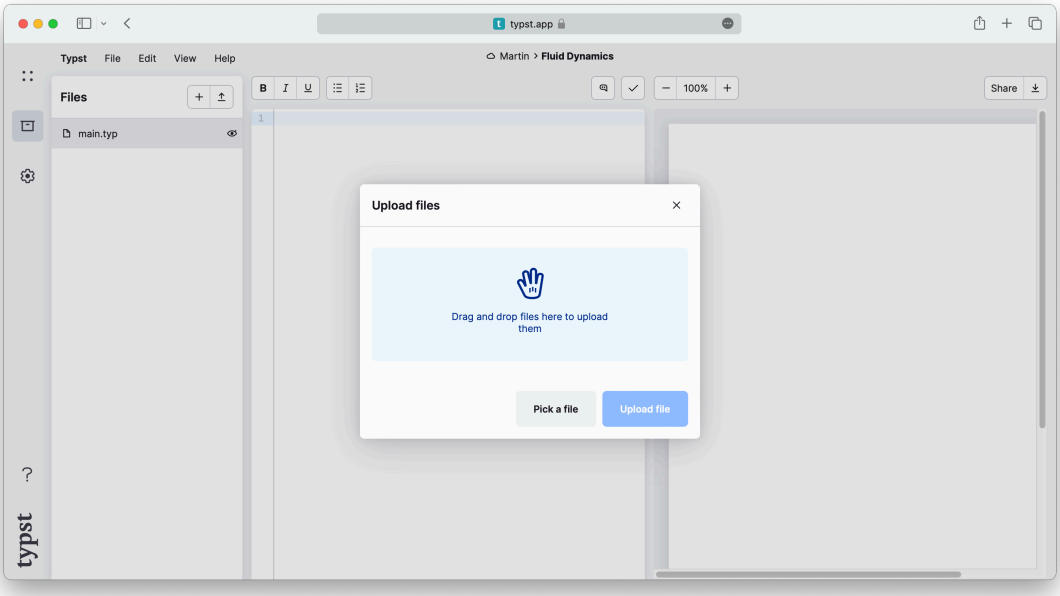
If we wanted to add a bulleted list, we would use the – character instead of the + character. We can also nest lists: For example, we can add a sub-list to the first item of the list above by indenting it.

```
+ The climate
  – Temperature
  – Precipitation
+ The topography
+ The geology
```

1. The climate
   - Temperature
   - Precipitation
2. The topography
3. The geology

2.1.1 Adding a figure

You think that your report would benefit from a figure. Let's add one. Typst supports images in the formats PNG, JPEG, GIF, SVG, PDF, and WebP. To add an image file to your project, first open the *file panel* by clicking the box icon in the left sidebar. Here, you can see a list of all files in your project. Currently, there is only one: The main Typst file you are writing in. To upload another file, click the button with the arrow in the top-right corner. This opens the upload dialog, in which you can pick files to upload from your computer. Select an image file for your report.



We have seen before that specific symbols (called *markup*) have specific meaning in Typst. We can use =, –, +, and _ to create headings, lists and emphasized text, respectively. However, having a special symbol for everything we want to insert into our document would soon become cryptic and unwieldy. For this reason,

Typst reserves markup symbols only for the most common things. Everything else is inserted with *functions*. For our image to show up on the page, we use Typst's `image`[3.11.7] function.

```
#image("glacier.jpg")
```



In general, a function produces some output for a set of *arguments*. When you *call* a function within markup, you provide the arguments and Typst inserts the result (the function's *return value*) into the document. In our case, the `image` function takes one argument: The path to the image file. To call a function in markup, we first need to type the `#` character, immediately followed by the name of the function. Then, we enclose the arguments in parentheses. Typst recognizes many different data types within argument lists. Our file path is a short string of text[3.5.27], so we need to enclose it in double quotes.

The inserted image uses the whole width of the page. To change that, pass the `width` argument to the `image` function. This is a *named* argument and therefore specified as a `name: value` pair. If there are multiple arguments, they are separated by commas, so we first need to put a comma behind the path.

```
#image("glacier.jpg", width: 70%)
```



The `width` argument is a relative length[3.10.21]. In our case, we specified a percentage, determining that the image shall take up `70%` of the page's width. We also could have specified an absolute value like `1cm` or `0.7in`.

Just like text, the image is now aligned at the left side of the page by default. It's also lacking a caption. Let's fix that by using the figure[3.6.8] function. This function takes the figure's contents as a positional argument and an optional caption as a named argument.

Within the argument list of the `figure` function, Typst is already in code mode. This means, you now have to remove the hash before the image function call. The hash is only needed directly in markup (to disambiguate text from function calls).

The caption consists of arbitrary markup. To give markup to a function, we enclose it in square brackets. This construct is called a *content block*.

```
#figure(
  image("glacier.jpg", width: 70%),
  caption: [
    _Glaciers_ form an important part
    of the earth's climate system.
  ],
)
```



Figure 1: *Glaciers* form an important part of the earth's climate system.

You continue to write your report and now want to reference the figure. To do that, first attach a label to figure. A label uniquely identifies an element in your document. Add one after the figure by enclosing some name in angle brackets. You can then reference the figure in your text by writing an `@` symbol followed by that name. Headings and equations can also be labelled to make them referenceable.

```
Glaciers as the one shown in
@glaciers will cease to exist if
we don't take action soon!

#figure(
  image("glacier.jpg", width: 70%),
  caption: [
    _Glaciers_ form an important part
    of the earth's climate system.
  ],
) <glaciers>
```

Glaciers as the one shown in Figure 1 will cease to exist if we don't take action soon!



Figure 1: *Glaciers* form an important part of the earth's climate system.

INFO

So far, we've passed content blocks (markup in square brackets) and strings (text in double quotes) to our functions. Both seem to contain text. What's the difference?

A content block can contain text, but also any other kind of markup, function calls, and more, whereas a string is really just a *sequence of characters* and nothing else.

For example, the image function expects a path to an image file. It would not make sense to pass, e.g., a paragraph of text or another image as the image's path parameter. That's why only strings are allowed here. In contrast, strings work wherever content is expected because text is a valid kind of content.

2.1.2 Adding a bibliography

As you write up your report, you need to back up some of your claims. You can add a bibliography to your document with the `bibliography`[3.6.2] function. This function expects a path to a bibliography file.

Typst's native bibliography format is Hayagriva[A], but for compatibility you can also use BibLaTeX files. As your classmate has already done a literature survey and sent you a `.bib` file, you'll use that one. Upload the file through the file panel to access it in Typst.

Once the document contains a bibliography, you can start citing from it. Citations use the same syntax as references to a label. As soon as you cite a source for the first time, it will appear in the bibliography section of your document. Typst supports different citation and bibliography styles. Consult the reference[3.6.2/4.4] for more details.

[A] https://github.com/typst/hayagriva/blob/main/docs/file-format.md

```
= Methods
We follow the glacier melting models
established in @glacier-melt.

#bibliography("works.bib")
```

Methods

We follow the glacier melting models established in [1].

Bibliography

[1] R. Hock, "Glacier melt: a review of processes and their modelling," *Progress in Physical Geography: Earth and Environment*, vol. 29, no. 3, pp. 362–391, 2005, doi: 10.1191/0309133305pp453ra.

2.1.3 Maths

After fleshing out the methods section, you move on to the meat of the document: Your equations. Typst has built-in mathematical typesetting and uses its own math notation. Let's start with a simple equation. We wrap it in $ signs to let Typst know it should expect a mathematical expression:

```
The equation $Q = rho A v + C$
defines the glacial flow rate.
```

The equation $Q = \rho A v + C$ defines the glacial flow rate.

The equation is typeset inline, on the same line as the surrounding text. If you want to have it on its own line instead, you should insert a single space at its start and end:

```
The flow rate of a glacier is
defined by the following equation:

$ Q = rho A v + C $
```

The flow rate of a glacier is defined by the following equation:

$$Q = \rho A v + C$$

We can see that Typst displayed the single letters `Q`, `A`, `v`, and `C` as-is, while it translated `rho` into a Greek letter. Math mode will always show single letters verbatim. Multiple letters, however, are interpreted as symbols, variables, or function names. To imply a multiplication between single letters, put spaces between them.

If you want to have a variable that consists of multiple letters, you can enclose it in quotes:

```
The flow rate of a glacier is given
by the following equation:

$ Q = rho A v + "time offset" $
```

The flow rate of a glacier is given by the following equation:

$$Q = \rho A v + \text{time offset}$$

You'll also need a sum formula in your paper. We can use the `sum` symbol and then specify the range of the summation in sub- and superscripts:

```
Total displaced soil by glacial flow:

$ 7.32 beta +
  sum_(i=0)^nabla Q_i / 2 $
```

Total displaced soil by glacial flow:

$$7.32\beta + \sum_{i=0}^{\nabla} \frac{Q_i}{2}$$

To add a subscript to a symbol or variable, type a `_` character and then the subscript. Similarly, use the `^` character for a superscript. If your sub- or superscript consists of multiple things, you must enclose them in round parentheses.

The above example also showed us how to insert fractions: Simply put a `/` character between the numerator and the denominator and Typst will automatically turn it into a fraction. Parentheses are smartly resolved, so you can enter your expression as you would into a calculator and Typst will replace parenthesized sub-expressions with the appropriate notation.

```
Total displaced soil by glacial flow:

$ 7.32 beta +
  sum_(i=0)^nabla
    (Q_i (a_i - epsilon)) / 2 $
```

Total displaced soil by glacial flow:

$$7.32\beta + \sum_{i=0}^{\nabla} \frac{Q_i(a_i - \varepsilon)}{2}$$

Not all math constructs have special syntax. Instead, we use functions, just like the `image` function we have seen before. For example, to insert a column vector, we can use the `vec`[3.8.27] function. Within math mode, function calls don't need to start with the `#` character.

```
$ v := vec(x_1, x_2, x_3) $
```

$$v := \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Some functions are only available within math mode. For example, the `cal`[3.8.26/1.6] function is used to typeset calligraphic letters commonly used for sets. The math section of the reference[3.8] provides a complete list of all functions that math mode makes available.

One more thing: Many symbols, such as the arrow, have a lot of variants. You can select among these variants by appending a dot and a modifier name to a symbol's name:

```
$ a arrow.squiggly b $
```

$$a \rightsquigarrow b$$

This notation is also available in markup mode, but the symbol name must be preceded with `#sym.` there. See the symbols section[3.9.2] for a list of all available symbols.

2.1.4 Review

You have now seen how to write a basic document in Typst. You learned how to emphasize text, write lists, insert images, align content, and typeset mathematical expressions. You also learned about Typst's functions. There are many more kinds of content that Typst lets you insert into your document, such as tables[3.6.20], shapes[3.11], and code blocks[3.7.6]. You can peruse the reference[3] to learn more about these and other features.

For the moment, you have completed writing your report. You have already saved a PDF by clicking on the download button in the top right corner. However, you think the report could look a bit less plain. In the next section, we'll learn how to customize the look of our document.

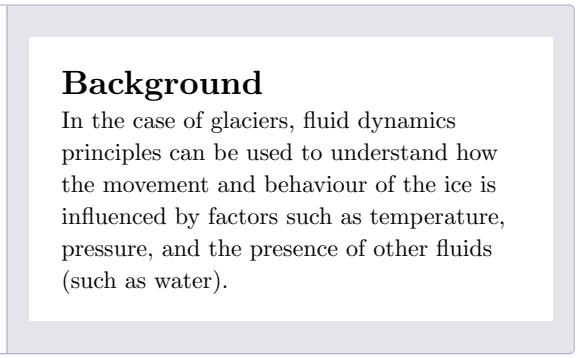
2.2 Formatting

So far, you have written a report with some text, a few equations and images. However, it still looks very plain. Your teaching assistant does not yet know that you are using a new typesetting system, and you want your report to fit in with the other student's submissions. In this chapter, we will see how to format your report using Typst's styling system.

2.2.1 Set rules

As we have seen in the previous chapter, Typst has functions that *insert* content (e.g. the `image`[3.11.7] function) and others that *manipulate* content that they received as arguments (e.g. the `align`[3.10.1] function). The first impulse you might have when you want, for example, to change the font, could be to look for a function that does that and wrap the complete document in it.

```
#text(font: "New Computer Modern")[
  = Background
  In the case of glaciers, fluid
  dynamics principles can be used
  to understand how the movement
  and behaviour of the ice is
  influenced by factors such as
  temperature, pressure, and the
  presence of other fluids (such as
  water).
]
```



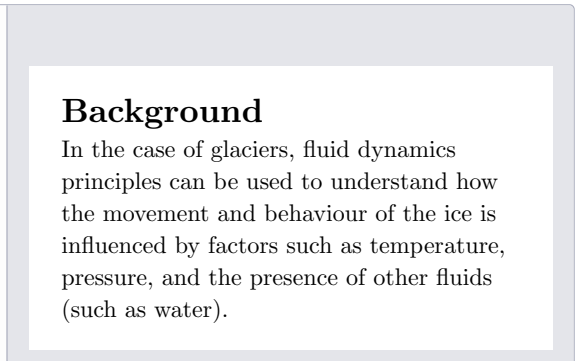
Wait, shouldn't all arguments of a function be specified within parentheses? Why is there a second set of square brackets with content *after* the parentheses? The answer is that, as passing content to a function is such a common thing to do in Typst, there is special syntax for it: Instead of putting the content inside of the argument list, you can write it in square brackets directly after the normal arguments, saving on punctuation.

As seen above, that works. With the `text`[3.7.12] function, we can adjust the font for all text within it. However, wrapping the document in countless functions and applying styles selectively and in-situ can quickly become cumbersome.

Fortunately, Typst has a more elegant solution. With *set rules,* you can apply style properties to all occurrences of some kind of content. You write a set rule by entering the `set` keyword, followed by the name of the function whose properties you want to set, and a list of arguments in parentheses.

```
#set text(
  font: "New Computer Modern"
)

= Background
In the case of glaciers, fluid
dynamics principles can be used
to understand how the movement
and behaviour of the ice is
influenced by factors such as
temperature, pressure, and the
presence of other fluids (such as
water).
```



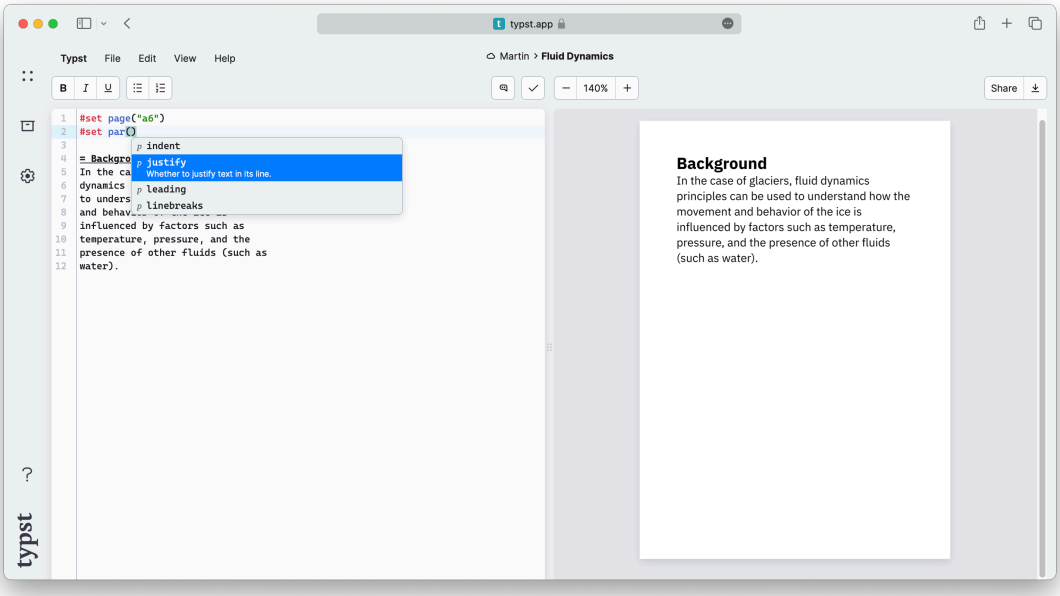
INFO

Want to know in more technical terms what is happening here?

Set rules can be conceptualized as setting default values for some of the parameters of a function for all future uses of that function.

2.2.2 The autocomplete panel

If you followed along and tried a few things in the app, you might have noticed that always after you enter a # character, a panel pops up to show you the available functions, and, within an argument list, the available parameters. That's the autocomplete panel. It can be very useful while you are writing your document: You can apply its suggestions by hitting the Return key or navigate to the desired completion with the arrow keys. The panel can be dismissed by hitting the Escape key and opened again by typing # or hitting `Ctrl` + `Space`. Use the autocomplete panel to discover the right arguments for functions. Most suggestions come with a small description of what they do.



2.2.3 Set up the page

Back to set rules: When writing a rule, you choose the function depending on what type of element you want to style. Here is a list of some functions that are commonly used in set rules:

- `text`[3.7.12] to set font family, size, color, and other properties of text
- `page`[3.10.17] to set the page size, margins, headers, enable columns, and footers
- `par`[3.6.15] to justify paragraphs, set line spacing, and more
- `heading`[3.6.10] to set the appearance of headings and enable numbering
- `document`[3.6.6] to set the metadata contained in the PDF output, such as title and author

Not all function parameters can be set. In general, only parameters that tell a function *how* to do something can be set, not those that tell it *what* to do it with. The function reference pages indicate which parameters are settable.

Let's add a few more styles to our document. We want larger margins and a serif font. For the purposes of the example, we'll also set another page size.

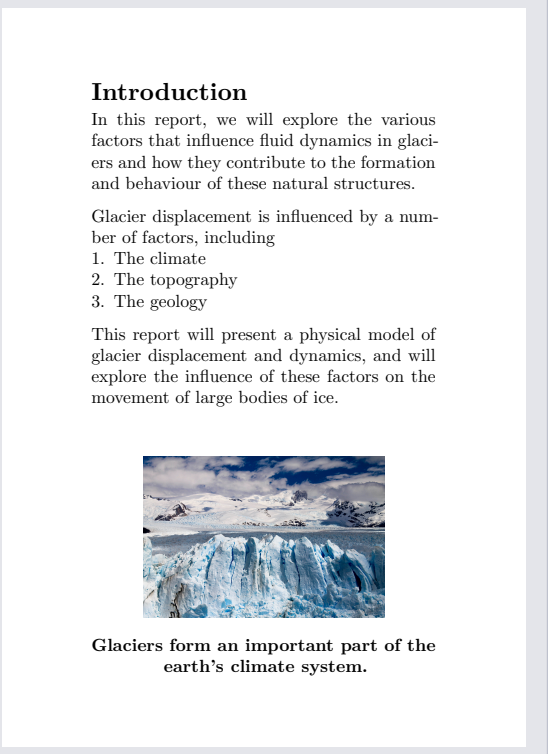
```
#set page(
  paper: "a6",
  margin: (x: 1.8cm, y: 1.5cm),
)
#set text(
  font: "New Computer Modern",
  size: 10pt
)
#set par(
  justify: true,
  leading: 0.52em,
)

= Introduction
In this report, we will explore the
various factors that influence fluid
dynamics in glaciers and how they
contribute to the formation and
behaviour of these natural structures.

...

#align(center + bottom)[
  #image("glacier.jpg", width: 70%)

  *Glaciers form an important
  part of the earth's climate
  system.*
]
```



There are a few things of note here.

First is the page[3.10.17] set rule. It receives two arguments: the page size and margins for the page. The page size is a string. Typst accepts many standard page sizes,[3.10.17/4.1] but you can also specify a custom page size. The margins are specified as a dictionary.[3.5.11] Dictionaries are a collection of key-value pairs. In this case, the keys are `x` and `y`, and the values are the horizontal and vertical margins, respectively. We could also have specified separate margins for each side by passing a dictionary with the keys `left`, `right`, `top`, and `bottom`.

Next is the set `text`[3.7.12] set rule. Here, we set the font size to `10pt` and font family to `"New Computer Modern"`. The Typst app comes with many fonts that you can try for your document. When you are in the text function's argument list, you can discover the available fonts in the autocomplete panel.

We have also set the spacing between lines (a.k.a. leading): It is specified as a length[3.10.13] value, and we used the `em` unit to specify the leading relative to the size of the font: `1em` is equivalent to the current font size (which defaults to `11pt`).

Finally, we have bottom aligned our image by adding a vertical alignment to our center alignment. Vertical and horizontal alignments can be combined with the `+` operator to yield a 2D alignment.

2.2.4 A hint of sophistication

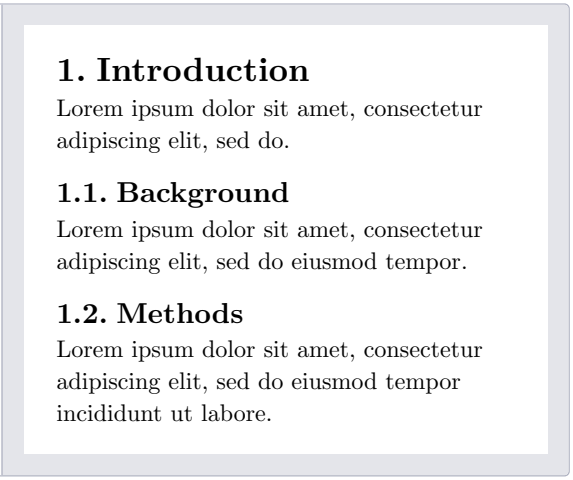
To structure our document more clearly, we now want to number our headings. We can do this by setting the `numbering` parameter of the `heading`[3.6.10] function.

```
#set heading(numbering: "1.")

= Introduction
#lorem(10)

== Background
#lorem(12)

== Methods
#lorem(15)
```



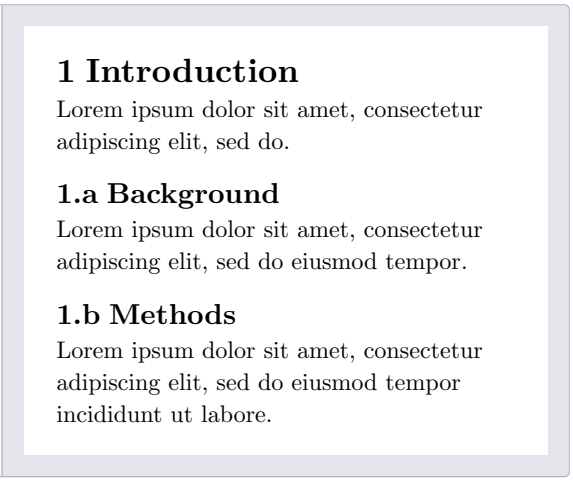
We specified the string `"1."` as the numbering parameter. This tells Typst to number the headings with arabic numerals and to put a dot between the number of each level. We can also use letters, roman numerals, and symbols[3.6.13] for our headings:

```
#set heading(numbering: "1.a")

= Introduction
#lorem(10)

== Background
#lorem(12)

== Methods
#lorem(15)
```



This example also uses the `lorem`[3.7.3] function to generate some placeholder text. This function takes a number as an argument and generates that many words of *Lorem Ipsum* text.

INFO

Did you wonder why the headings and text set rules apply to all text and headings, even if they are not produced with the respective functions?

Typst internally calls the `heading` function every time you write `= Conclusion`. In fact, the function call `#heading[Conclusion]` is equivalent to the heading markup above. Other markup elements work similarly, they are only *syntax sugar* for the corresponding function calls.

2.2.5 Show rules

You are already pretty happy with how this turned out. But one last thing needs to be fixed: The report you are writing is intended for a larger project and that project's name should always be accompanied by a logo, even in prose.

You consider your options. You could add an `#image("logo.svg")` call before every instance of the logo using search and replace. That sounds very tedious. Instead, you could maybe define a custom function[3.5.15/4] that always yields the logo with its image. However, there is an even easier way:

With show rules, you can redefine how Typst displays certain elements. You specify which elements Typst should show differently and how they should look. Show rules can be applied to instances of text, many functions, and even the whole document.

```
#show "ArtosFlow": name => box[
  #box(image(
    "logo.svg",
    height: 0.7em,
  ))
  #name
]

This report is embedded in the
ArtosFlow project. ArtosFlow is a
project of the Artos Institute.
```

This report is embedded in the ArtosFlow project. ArtosFlow is a project of the Artos Institute.

There is a lot of new syntax in this example: We write the `show` keyword, followed by a string of text we want to show differently and a colon. Then, we write a function that takes the content that shall be shown as an argument. Here, we called that argument `name`. We can now use the `name` variable in the function's body to print the ArtosFlow name. Our show rule adds the logo image in front of the name and puts the result into a box to prevent linebreaks from occurring between logo and name. The image is also put inside of a box, so that it does not appear in its own paragraph.

The calls to the first box function and the image function did not require a leading # because they were not embedded directly in markup. When Typst expects code instead of markup, the leading # is not needed to access functions, keywords, and variables. This can be observed in parameter lists, function definitions, and code blocks[3.3].

2.2.6 Review

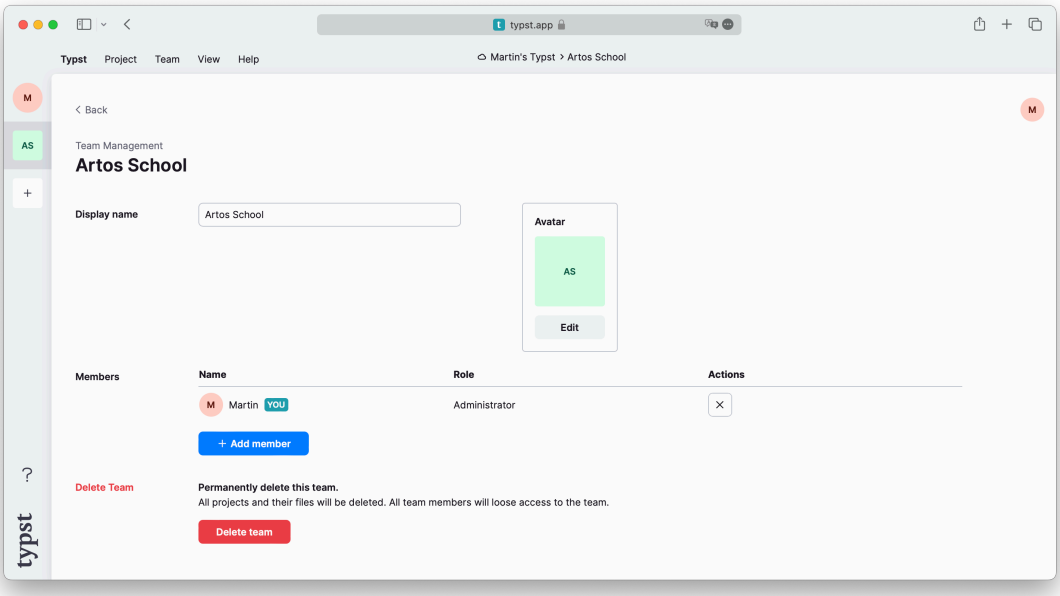
You now know how to apply basic formatting to your Typst documents. You learned how to set the font, justify your paragraphs, change the page dimensions, and add numbering to your headings with set rules. You also learned how to use a basic show rule to change how text appears throughout your document.

You have handed in your report. Your supervisor was so happy with it that they want to adapt it into a conference paper! In the next section, we will learn how to format your document as a paper using more advanced show rules and functions.

2.3 Advanced Styling

In the previous two chapters of this tutorial, you have learned how to write a document in Typst and how to change its formatting. The report you wrote throughout the last two chapters got a straight A and your supervisor wants to base a conference paper on it! The report will of course have to comply with the conference's style guide. Let's see how we can achieve that.

Before we start, let's create a team, invite your supervisor and add them to the team. You can do this by going back to the app dashboard with the back icon in the top left corner of the editor. Then, choose the plus icon in the left toolbar and create a team. Finally, click on the new team and go to its settings by clicking 'manage team' next to the team name. Now you can invite your supervisor by email.



Next, move your project into the team: Open it, going to its settings by choosing the gear icon in the left toolbar and selecting your new team from the owners dropdown. Don't forget to save your changes!

Now, your supervisor can also edit the project and you can both see the changes in real time. You can join our Discord server[A] to find other users and try teams with them!

2.3.1 The conference guidelines

The layout guidelines are available on the conference website. Let's take a look at them:

- The font should be an 11pt serif font
- The title should be in 17pt and bold
- The paper contains a single-column abstract and two-column main text
- The abstract should be centered
- The main text should be justified
- First level section headings should be 13pt, centered, and rendered in small capitals
- Second level headings are run-ins, italicized and have the same size as the body text
- Finally, the pages should be US letter sized, numbered in the center of the footer and the top right corner of each page should contain the title of the paper

We already know how to do many of these things, but for some of them, we'll need to learn some new tricks.

2.3.2 Writing the right set rules

Let's start by writing some set rules for the document.

[A] https://discord.gg/2uDybryKPe

```
#set page(
  paper: "us-letter",
  header: align(right)[
    A Fluid Dynamic Model for
    Glacier Flow
  ],
  numbering: "1",
)
#set par(justify: true)
#set text(
  font: "Libertinus Serif",
  size: 11pt,
)

#lorem(600)
```



You are already familiar with most of what is going on here. We set the text size to `11pt` and the font to Libertinus Serif. We also enable paragraph justification and set the page size to US letter.

The `header` argument is new: With it, we can provide content to fill the top margin of every page. In the header, we specify our paper's title as requested by the conference style guide. We use the `align` function to align the text to the right.

Last but not least is the `numbering` argument. Here, we can provide a numbering pattern[3.6.13] that defines how to number the pages. By setting it to `"1"`, Typst only displays the bare page number. Setting it to `"(1/1)"` would have displayed the current page and total number of pages surrounded by parentheses. And we could even have provided a completely custom function here to format things to our liking.

2.3.3 Creating a title and abstract

Now, let's add a title and an abstract. We'll start with the title. Typst comes with a `title`[3.6.22] function. Let's start by providing our title as an argument:

```
#title[
  A Fluid Dynamic Model
  for Glacier Flow
]
```

A Fluid Dynamic Model for Glacier Flow

You can see that the title is already boldfaced and has some space around it. However, it is left-aligned and not exactly 17pt large. Hence, we need to adjust its appearance. The title function does not come with any arguments for font or text size we could set. Instead, these properties are defined on the `text` and `align` functions.

INFO

What is the difference between what the `title` function inserted and the headings we produced with equals signs?

Headings, even first-level headings, can appear multiple times in your document whereas a title only appears once, usually at the beginning. Differentiating between the two helps Typst make your document accessible for users of Assistive Technology such as screen readers.

When we want to customize the properties of some element inside of another kind of element, we can use show-set rules. First, we use `show` to select which element we want to customize. We call this a *selector*. Then, we type a colon. Next, we write the set rule that should apply to elements matching the selector. Summarized, the syntax looks like this:

```
#show your-selector: set some-element(/* ... */)
```

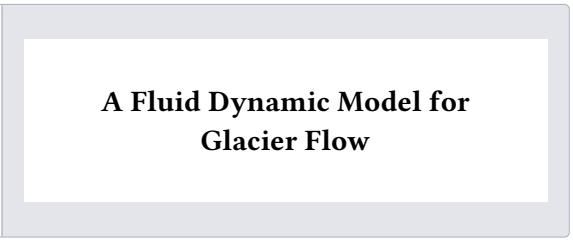
Let's recall: We want to center-align the title and make it 17pt large. Hence, we need two show-set rules:

- One with the selector `title` and the rule `set text(size: 17pt)`
- One with the selector `title` and the rule `set align(center)`

Our example now looks like this:

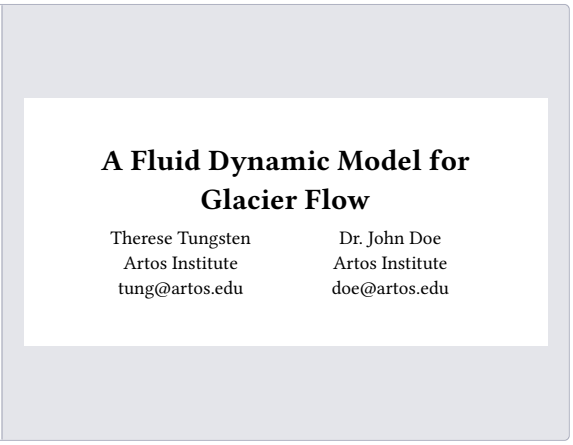
```
#show title: set text(size: 17pt)
#show title: set align(center)

#title[
  A Fluid Dynamic Model
  for Glacier Flow
]
```



This looks right. Let's also add the author list: Since we are writing this paper together with our supervisor, we'll add our own and their name.

```
#grid(
  columns: (1fr, 1fr),
  align(center)[
    Therese Tungsten \
    Artos Institute \
    #link("mailto:tung@artos.edu")
  ],
  align(center)[
    Dr. John Doe \
    Artos Institute \
    #link("mailto:doe@artos.edu")
  ]
)
```



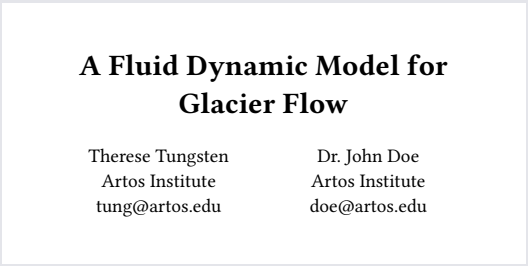
The two author blocks are laid out next to each other. We use the `grid`[3.10.10] function to create this layout. With a grid, we can control exactly how large each column is and which content goes into which cell. The `columns` argument takes an array of relative lengths[3.10.21] or fractions[3.10.9]. In this case, we passed it two equal fractional sizes, telling it to split the available space into two equal columns. We then passed two content arguments to the grid function. The first with our own details, and the second with our supervisors'. We again use the `align` function to center the content within the column. The grid takes an arbitrary number of content arguments specifying the cells. Rows are added automatically, but they can also be manually sized with the `rows` argument.

Looking at the authors and the title, they are a bit too close together. You can address this by using another show-set rule to configure the space below the title. The title, the grid, and all other elements that Typst arranges from the top to the bottom of the page (except for paragraphs) are called *blocks*. Each block is controlled by the `block`[3.10.4] function. It controls behaviors like their distance and whether a block can contain a page break. That means that we can write another show-set rule that selects the title to set the block spacing:

```
#show title: set text(size: 17pt)
#show title: set align(center)
#show title: set block(below: 1.2em)

#title[
  A Fluid Dynamic Model
  for Glacier Flow
]

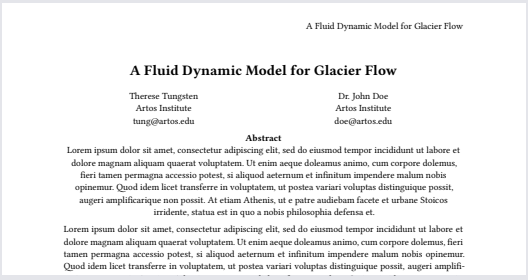
#grid(
  // ...
)
```



With this show-set rule, we overrode the spacing below the title. We have used the `em` unit: It allows us to express lengths as multiples of the font size. Here, we used it to space the title and the author list exactly 1.2× the font size apart. Now, let's add the abstract. Remember that the conference wants the abstract to be set ragged and centered.

```
...

#align(center)[
  #set par(justify: false)
  *Abstract* \
  #lorem(80)
]
```



Well done! One notable thing is that we used a set rule within the content argument of `align` to turn off justification for the abstract. This does not affect the remainder of the document even though it was specified after the first set rule because content blocks *scope* styling. Anything set within a content block will only affect the content within that block.

Another tweak could be to remove the duplication between the header and the title element's argument. Since they share the title, it would be convenient to store it in a place designed to hold metadata about the document. We would then need a way to retrieve the title in both places. The `document` element can help us with the former: By using it in a set rule, we can store document metadata like title, description, and keywords.

```
#set document(title: [A Fluid Dynamic Model for Glacier Flow])
```

When exporting a PDF, the title set here will appear in the title bar of your PDF reader. Your operating system will also use this title to make the file retrievable with search. Last but not least, it contributes to making your document more accessible and is required if you choose to comply with PDF/UA, a PDF standard focused on accessibility.

Now, we need a way to retrieve the value we set in the main title and the header. Because the `title` function is designed to work together with the `document` element, calling it with no arguments will just print the title. For the header, we will need to be more explicit: Because Typst has no way of knowing that we want to insert the title there, we will need to tell it to do so manually.

Using *context,* we can retrieve the contents of any values we have set on elements before. When we use the `context` keyword, we can access any property of any element, including the document element's title property. Its use looks like this:

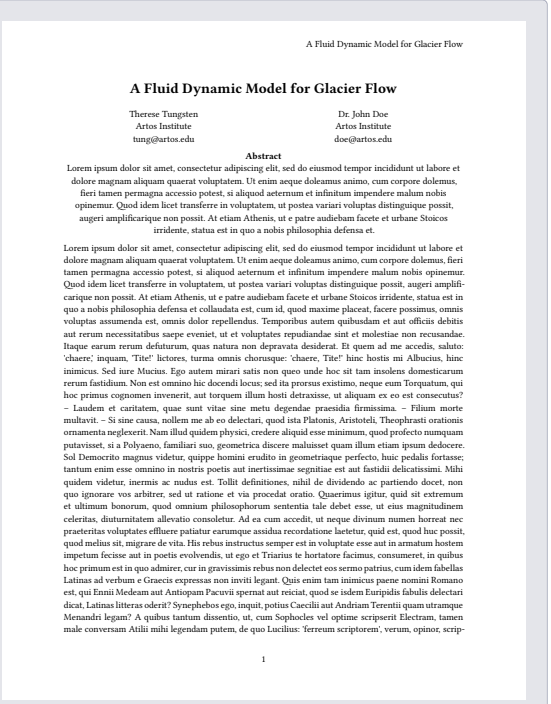
```
#set document(title: [
  A Fluid Dynamic Model
  for Glacier Flow
])

...

#set page(
  header: align(
    right + horizon,
    // Retrieve the document
    // element's title property.
    context document.title,
  ),
  ...
)

#title()

...
```



First, notice how we called the title function with empty, round parentheses. Because no argument was passed, it defaulted to what we set for the document element above. The distinction between empty round and empty square brackets is important: While empty round brackets show that you are passing nothing, empty square brackets mean that you are passing one argument: an empty content block. If called that way, the title would have no visible content.

Next, take a look at the header. Instead of the title in square parentheses, we used the context keyword to access the document title. This inserted exactly what we set above. The role of context is not limited to accessing properties: With it, you can check if some elements are present in the document, measure the physical dimensions of others, and more. Using context, you can build powerful templates that react to the preferences of the end-user.

INFO

Why is the context keyword required to access element properties?

Normally, when we access a variable, we know exactly what its value is going to be:

- The variable could be a constant built into Typst, like `#sym.pi`
- The variable could be defined by an argument
- The variable could be defined or overwritten in the current scope

However, sometimes, that's not enough. In this chapter of the tutorial, we have inserted a page header with the title. Even though we pass only one piece of content for the header, we may want different pages to have different headers. For example, we may want to print the chapter name or use the page number. When we use context, we can write a single context block that tells Typst to take a look at where it's inserted, look for the last heading, the current page number, or anything else, and go from there. That means that the same context block, inserted on different pages, can produce different output.

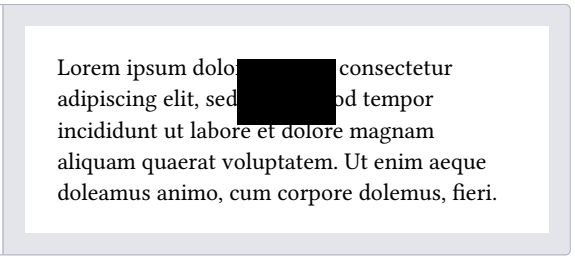
For more information, read up on context in its docs[3.4] after completing this tutorial.

2.3.4 Adding columns and headings

The paper above unfortunately looks like a wall of lead. To fix that, let's add some headings and switch our paper to a two-column layout. Fortunately, that's easy to do: We just need to amend our `page` set rule with the `columns` argument.

By adding `columns: 2` to the argument list, we have wrapped the whole document in two columns. However, that would also affect the title and authors overview. To keep them spanning the whole page, we can wrap them in a function call to `place`[3.10.19]. Place expects an alignment and the content it should place as positional arguments. Using the named `scope` argument, we can decide if the items should be placed relative to the current column or its parent (the page). There is one more thing to configure: If no other arguments are provided, `place` takes its content out of the flow of the document and positions it over the other content without affecting the layout of other content in its container:

```
#place(
  top + center,
  rect(fill: black),
)
#lorem(30)
```



If we hadn't used `place` here, the square would be in its own line, but here it overlaps the few lines of text following it. Likewise, that text acts as if there was no square. To change this behavior, we can pass the argument `float: true` to ensure that the space taken up by the placed item at the top or bottom of the page is not occupied by any other content.

```
#set page(
  paper: "us-letter",
  header: align(
    right + horizon,
    context document.title,
  ),
  numbering: "1",
  columns: 2,
)

#place(
  top + center,
  float: true,
  scope: "parent",
  clearance: 2em,
)[
  ...

  #par(justify: false)[
    *Abstract* \
    #lorem(80)
  ]
]

= Introduction
#lorem(300)

= Related Work
#lorem(200)
```



In this example, we also used the `clearance` argument of the `place` function to provide the space between it and the body instead of using the `v`[3.10.27] function. We can also remove the explicit `align(center, ..)` calls around the various parts since they inherit the center alignment from the placement.

Now there is only one thing left to do: Style our headings. We need to make them centered and use small capitals. These properties are not available on the `heading` function, so we will need to write a few show-set rules and a show rule:

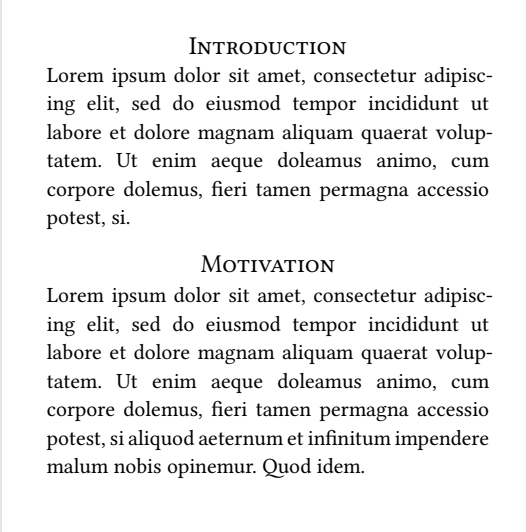
- A show-set rule to make headings center-aligned
- A show-set rule to make headings 13pt large and use the regular weight
- A show rule to wrap the headings in a call to the `smallcaps` function

```
#show heading: set align(center)
#show heading: set text(
  size: 13pt,
  weight: "regular",
)
#show heading: smallcaps

...

= Introduction
...

== Motivation
...
```

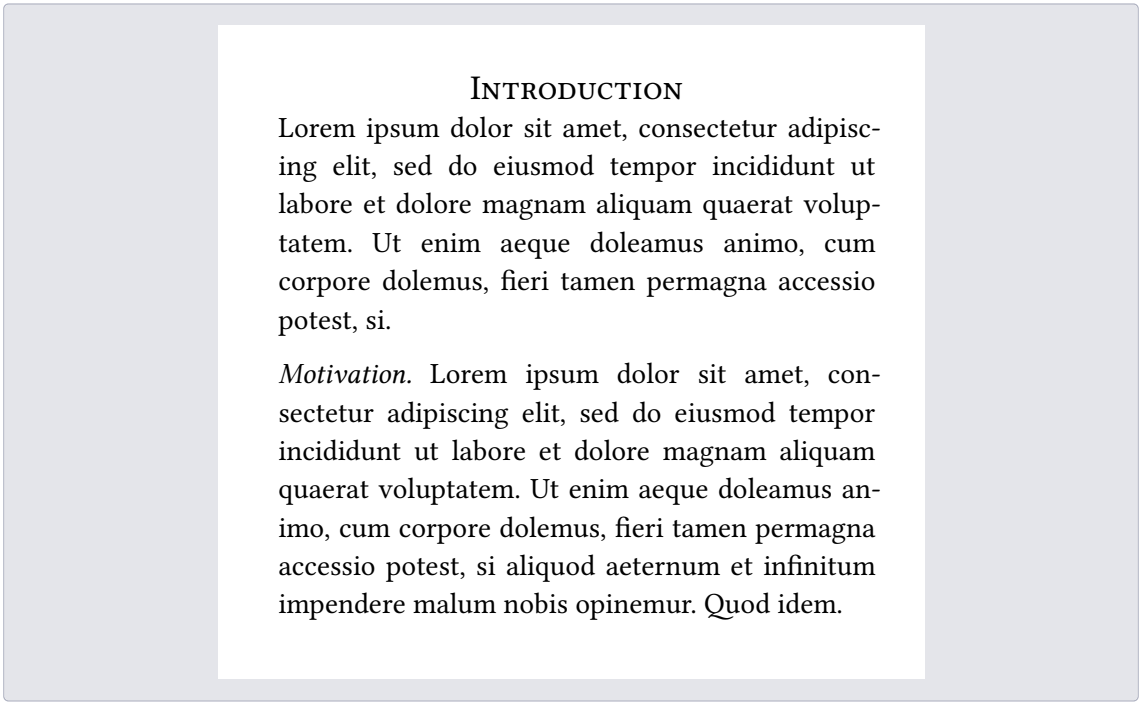


This looks great! We used show rules that apply to all headings. In the final show rule, we applied the `smallcaps` function to the complete heading. As we will see in the next example, we can also provide a custom rule to completely override the default look of headings.

The only remaining problem is that all headings look the same now. The "Motivation" and "Problem Statement" subsections ought to be italic run-in headers, but right now, they look indistinguishable from the section headings. We can fix that by using a `where` selector on our show rule: This is a method[3.3.7] we can call on headings (and other elements) that allows us to filter them by their properties. We can use it to differentiate between section and subsection headings:

```
#show heading.where(level: 1): set align(center)
#show heading.where(level: 1): set text(size: 13pt, weight: "regular")
#show heading.where(level: 1): smallcaps

#show heading.where(level: 2): set text(
  size: 11pt,
  weight: "regular",
  style: "italic",
)
#show heading.where(level: 2): it => {
  it.body + [.]
}
```



In this example, we first scope our previous rules to first-level headings by using `.where(level: 1)` to make the selector more specific. Then, we add a show-set rule for the second heading level. Finally, we need a show rule with a custom function: Headings enclose their contents with a block by default. This has the effect that the heading gets its own line. However, we want it to run into the text, so we need to provide our own show rule to get rid of this block.

We provide the rule with a function that takes the heading as a parameter. This parameter is conventionally called `it`, but can have another name. The parameter can be used as content and will just display the whole default heading. Alternatively, when we want to build our own heading instead, we can use its fields like `body`, `numbering`, and `level` to compose a custom look. Here, we are just printing the body of the heading with a trailing dot and leave out the block that the built-in show rule produces. Note that this heading will no longer react to set rules for heading numbering and similar because we did not explicitly use `it.numbering` in the show rule. If you are writing show rules like this and want the document to remain customizable, you will need to take these fields into account.

This looks great! We wrote show rules that selectively apply to the first and second level headings. We used a `where` selector to filter the headings by their level. We then rendered the subsection headings as run-ins. We also automatically add a period to the end of the subsection headings.

Let's review the conference's style guide:
✓ The font should be an 11pt serif font
✓ The title should be in 17pt and bold
✓ The paper contains a single-column abstract and two-column main text
✓ The abstract should be centered
✓ The main text should be justified
✓ First level section headings should be centered, rendered in small caps and in 13pt
✓ Second level headings are run-ins, italicized and have the same size as the body text
✓ Finally, the pages should be US letter sized, numbered in the center and the top right corner of each page should contain the title of the paper

We are now in compliance with all of these styles and can submit the paper to the conference! The finished paper looks like this:



2.3.5 Review

You have now learned how to create titles, headers, and footers, how to use functions, show-set rules, and scopes to locally override styles, how to create more complex layouts with the `grid`[3.10.10] function, how to access element properties with context, and how to write show rules for individual functions, and the whole document. You also learned how to use the `where` selector[3.2.2] to filter the headings by their level.

The paper was a great success! You've met a lot of like-minded researchers at the conference and are planning a project which you hope to publish at the same venue next year. You'll need to write a new paper using the same style guide though, so maybe now you want to create a time-saving template for you and your team?

In the next section, we will learn how to create templates that can be reused in multiple documents. This is a more advanced topic, so feel free to come back to it later if you don't feel up to it right now.

2.4 Making a Template

In the previous three chapters of this tutorial, you have learned how to write a document in Typst, apply basic styles, and customize its appearance in-depth to comply with a publisher's style guide. Because the paper you wrote in the previous chapter was a tremendous success, you have been asked to write a follow-up article for the same conference. This time, you want to take the style you created in the previous chapter and turn it into a reusable template. In this chapter you will learn how to create a template that you and your team can use with just one show rule. Let's get started!

2.4.1 Reusing data with variables

In the past chapters, most of the content of the document was entered by hand. In the third chapter, we used the `document` element and context to cut down on repetition and only enter the title once. But in practice, there may be many more things that occur multiple times in your document. There are multiple good reasons to just define these repeated values once:

1. It makes changing them later easier
2. It allows you to quickly find all instances where you used something
3. It makes it easy to be consistent throughout
4. For long or hard-to-enter repeated segments, a shorter variable name is often more convenient to type

If you were using a conventional word processor, you might resort to using a placeholder value that you can later search for. In Typst, however, you can instead use variables to safely store content and reuse it across your whole document through a variable name.

The technique of using context to reproduce an element's property we have learned earlier is not always the most appropriate for this: Typst's built-in elements focus on semantic properties like the title and description of a document, or things that directly relate to typesetting, like the text size.

For our example, we want to take a look at Typst's pronunciation. One of the best ways to transcribe pronunciation is the International Phonetic Alphabet (IPA). But because it uses characters not found on common keyboards, typing IPA repeatedly can become cumbersome. So let's instead define a variable that we can reference multiple times.

```
#let ipa = [taɪpst]
```

Here, we use a new keyword, `let`, to indicate a variable definition. Then, we put the name of our variable, in this case, `ipa`. Finally, we type an equals sign and the value of our variable. It is enclosed in square brackets because it is content, mirroring how you would call a function accepting content. In other words, this syntax mirrors the phrase *"Let the variable* `ipa` *have the value* `[taɪpst]`*."*

Now, we can use the variable in our document:

```
#let ipa = [taɪpst]

The canonical way to
pronounce Typst is #ipa.

#table(
  columns: (1fr, 1fr),
  [Name], [Typst],
  [Pronunciation], ipa,
)
```

The canonical way to pronounce Typst is taɪpst.

Name	Typst
Pronunciation	taɪpst

In the example, you can see that the variable can be used both in markup (prefixed with a #) and in a function call (by just typing its name). Of course, we can change the value of the variable and all its occurrences will automatically change with it. Let's make it a bit clearer what is IPA and what is normal prose by rendering IPA in italics. We are also using slashes which, by convention, often enclose IPA.

```
#let ipa = text(
  style: "italic",
)[/taɪpst/]

The canonical way to
pronounce Typst is #ipa.

#table(
  columns: (1fr, 1fr),
  [Name], [Typst],
  [Pronunciation], ipa,
)
```

The canonical way to pronounce Typst is */taɪpst/*.

Name	Typst
Pronunciation	*/taɪpst/*

Here, we called the text function and assigned its *return value* to the variable. When you call a function, it processes its arguments and then yields another value (often content). So far in this tutorial, we called most functions directly in markup, like this: `#text(fill: red)[CRIMSON!]`. This call to the text function returns the red text as a return value. Because we placed it in markup, its return value just immediately got inserted into the content we wrote. With variables, we can instead store it to use it later or compose it with other values.

Variables are not limited to storing content: they can store any data type Typst knows about. Throughout this tutorial, you made use of many data types when you passed them to Typst's built-in functions. Here is an example assigning each of them to a variable:

```
// Content with markup inside
#let blind-text = [_Lorem ipsum_ dolor sit amet]

// Unformatted strings
#let funny-font = "MS Comic Sans"

// Absolute lengths (see also pt, in, ...)
#let mile = 160934cm

// Lengths relative to the font size
#let double-space = 2em

// Ratios
#let progress = 80%

// Integer numbers
#let answer = 42

// Booleans
#let truth = false

// Horizontal and vertical alignment
#let focus = center
```

In this chapter of the tutorial, you will leverage variables and your own functions to build templates that can be reused across multiple documents.

2.4.2 A toy template

In Typst, templates are functions in which you can wrap your whole document. To learn how to do that, let's first review how to write your very own functions. They can do anything you want them to, so why not go a bit crazy?

```
#let amazed(term) = box[✨ #term ✨]

You are #amazed[beautiful]!
```

You are ✨ beautiful ✨!

Comparing this against the previous section, you may have noticed that this looks a lot like a variable definition using `let`. This instinct is correct: Functions are just another data type. Here, we are defining the variable `amazed`, assigning it a function that takes a single argument, `term`, and returns content with the `term` surrounded by sparkles. We also put the whole thing in a `box`[3.10.5] so that the term we are amazed by cannot be separated from its sparkles by a line break. The special function definition syntax makes the definition shorter and more readable, but you can also use the regular variable definition syntax (see the scripting reference[3.3.3] for details). After its definition, we are able to call the function just like all built-in functions.

Many functions that come with Typst have optional named parameters. Our functions can also have them. Let's add a parameter to our function that lets us choose the color of the text. We need to provide a default color in case the parameter isn't given.

```
#let amazed(term, color: blue) = {
  text(color, box[✨ #term ✨])
}

You are #amazed[beautiful]!
I am #amazed(color: purple)[amazed]!
```

You are ✨ beautiful ✨! I am ✨ amazed ✨!

Templates now work by wrapping our whole document in a custom function like `amazed`. But wrapping a whole document in a giant function call would be cumbersome! Instead, we can use an "everything" show rule to achieve the same with cleaner code. To write such a show rule, put a colon directly after the show keyword and then provide a function. This function is given the rest of the document as a parameter. The function can then do anything with this content. Since the `amazed` function can be called with a single content argument, we can just pass it by name to the show rule. Let's try it:

```
#show: amazed
I choose to focus on the good
in my life and let go of any
negative thoughts or beliefs.
In fact, I am amazing!
```

✨ I choose to focus on the good in my life
and let go of any negative thoughts or beliefs.
In fact, I am amazing! ✨

Our whole document will now be passed to the `amazed` function, as if we wrapped it around it. Of course, this is not especially useful with this particular function, but when combined with set rules and named arguments, it can be very powerful.

2.4.3 Embedding set and show rules

To apply some set and show rules to our template, we can use `set` and `show` within a content block in our function and then insert the document into that content block.

```
#let template(doc) = [
  #set text(font: "Inria Serif")
  #show "something cool": [Typst]
  #doc
]

#show: template
I am learning something cool today.
It's going great so far!
```

I am learning Typst today. It's going great so far!

Just like we already discovered in the previous chapter, set rules will apply to everything within their content block. Since the everything show rule passes our whole document to the `template` function, the text set rule and string show rule in our template will apply to the whole document. Let's use this knowledge to create a template that reproduces the body style of the paper we wrote in the previous chapter.

```typ
#let conf(title, doc) = {
  set page(
    paper: "us-letter",
    header: align(
      right + horizon,
      title
    ),
    columns: 2,
    ...
  )
  set par(justify: true)
  set text(
    font: "Libertinus Serif",
    size: 11pt,
  )

  // Heading show rules.
  ...

  doc
}

#show: doc => conf(
  [Paper title],
  doc,
)

= Introduction
...
```



We copy-pasted most of that code from the previous chapter. The two differences are this:

1. We wrapped everything in the function `conf` using an everything show rule. The function applies a few set and show rules and echoes the content it has been passed at the end.
2. Moreover, we used a curly-braced code block instead of a content block. This way, we don't need to prefix all set rules and function calls with a #. In exchange, we cannot write markup directly in the code block anymore.

Also note where the title comes from: We previously had it inside of a variable. Now, we are receiving it as the first parameter of the template function. To do so, we passed a closure (that's a function without a name that is used right away) to the everything show rule. We did that because the `conf` function expects two positional arguments, the title and the body, but the show rule will only pass the body. Therefore, we add a new function definition that allows us to set a paper title and use the single parameter from the show rule.

2.4.4 Templates with named arguments

Our paper in the previous chapter had a title and an author list. We can keep the title as `document`[3.6.6] metadata, but our template should also accept a list of authors with their affiliations and the paper's abstract. We'll add those as named arguments. In the end, we want it to work like this:

```typ
#set document(title: [
  A Fluid Dynamic Model for
  Glacier Flow
])

#show: doc => conf(
  authors: (
    (
      name: "Theresa Tungsten",
      affiliation: "Artos Institute",
      email: "tung@artos.edu",
```

```
    ),
    (
      name: "Eugene Deklan",
      affiliation: "Honduras State",
      email: "e.deklan@hstate.hn",
    ),
  ),
  abstract: lorem(80),
  doc,
)

...
```

Let's build this new template function. The title can be displayed with the `title`[3.6.22] function and accessed through `document.title`, so the template only needs named `authors` and `abstract` parameters with empty defaults. Next, we copy the code that generates title, abstract and authors from the previous chapter into the template, replacing the fixed details with the parameters.

The new `authors` parameter expects an array[3.5.2] of dictionaries[3.5.11] with the keys `name`, `affiliation` and `email`. Because we can have an arbitrary number of authors, we dynamically determine if we need one, two or three columns for the author list. First, we determine the number of authors using the `.len()`[3.5.2/3.1] method on the `authors` array. Then, we set the number of columns as the minimum of this count and three, so that we never create more than three columns. If there are more than three authors, a new row will be inserted instead. For this purpose, we have also added a `row-gutter` parameter to the `grid` function. Otherwise, the rows would be too close together. To extract the details about the authors from the dictionary, we use the field access syntax[3.3.6].

We still have to provide an argument to the grid for each author: Here is where the array's `map` method[3.5.2/3.15] comes in handy. It takes a function as an argument that gets called with each item of the array. We pass it a function that formats the details for each author and returns a new array containing content values. We've now got one array of values that we'd like to use as multiple arguments for the grid. We can do that by using the spread operator[3.5.1]. It takes an array and applies each of its items as a separate argument to the function.

The resulting template function looks like this:

```
#let conf(
  authors: (),
  abstract: [],
  doc,
) = {
  // Set and show rules from before.
  // ...

  place(
    top + center,
    float: true,
    scope: "parent",
    clearance: 2em,
    {
      title()

      let count = authors.len()
      let ncols = calc.min(count, 3)
      grid(
        columns: (1fr,) * ncols,
        row-gutter: 24pt,
        ..authors.map(author => [
          #author.name \
          #author.affiliation \
          #link("mailto:" + author.email)
        ]),
```

```
      )

      par(justify: false)[
        *Abstract* \
        #abstract
      ]

    }
  )

  doc
}
```

2.4.5 A separate file

Most of the time, a template is specified in a different file and then imported into the document. This way, the main file you write in is kept clutter free and your template is easily reused. Create a new text file in the file panel by clicking the plus button and name it `conf.typ`. Move the `conf` function definition inside of that new file. Now you can access it from your main file by adding an import before the show rule. Specify the path of the file between the `import` keyword and a colon, then name the function that you want to import.

Another thing that you can do to make applying templates just a bit more elegant is to use the `.with`[3.5.15/8.1] method on functions to pre-populate all the named arguments. This way, you can avoid spelling out a closure and appending the content argument at the bottom of your template list. Templates on Typst Universe[A] are designed to work with this style of function call.

```
#import "conf.typ": conf

#set document(title: [
  A Fluid Dynamic Model for
  Glacier Flow
])

#show: conf.with(
  authors: (
    (
      name: "Theresa Tungsten",
      affiliation: "Artos Institute",
      email: "tung@artos.edu",
    ),
    (
      name: "Eugene Deklan",
      affiliation: "Honduras State",
      email: "e.deklan@hstate.hn",
    ),
  ),
  abstract: lorem(80),
)

= Introduction
#lorem(90)

== Motivation
#lorem(140)

== Problem Statement
#lorem(50)

= Related Work
#lorem(200)
```



[A] https://typst.app/universe

We have now converted the conference paper into a reusable template for that conference! Why not share it in the Forum[A] or on Typst's Discord server[B] so that others can use it too?

2.4.6 Review

Congratulations, you have completed Typst's Tutorial! In this section, you have learned how to define your own functions and how to create and apply templates that define reusable document styles. You've made it far and learned a lot. You can now use Typst to write your own documents and share them with others.

We are still a super young project and are looking for feedback. If you have any questions, suggestions or you found a bug, please let us know in the Forum[C], on our Discord server[D], on GitHub[E], or via the web app's feedback form (always available in the Help menu).

So what are you waiting for? Sign up[F] and write something!

[A] https://forum.typst.app/
[B] https://discord.gg/2uDybryKPe
[C] https://forum.typst.app/
[D] https://discord.gg/2uDybryKPe
[E] https://github.com/typst/typst/
[F] https://typst.app

3 Reference

This reference documentation is a comprehensive guide to all of Typst's syntax, concepts, functions, types, and other definitions. Use the reference to answer specific questions about Typst and to broaden your understanding of the available features.

If you are completely new to Typst, we recommend starting with the tutorial[2] and then coming back to the reference to learn more about Typst's features as you need them.

3.0.1 Language

The reference starts by covering fundamentals of the Typst language. First, we give an overview of Typst's syntax.[3.1] The following sections cover core concepts central to the Typst language such as styling documents,[3.2] using Typst's scripting capabilities,[3.3] and reasoning about the contents of your document.[3.4]

3.0.2 Library

Starting with Foundations[3.5], the reference includes sections on all functions, types, and other definitions provided by the *standard library* of the Typst language.

The definition sections are grouped by topic. For example, if you would like to explore all tools Typst provides to adjust where elements land on the page, you should start in the Layout[3.10] section.

3.0.3 Export

Some of the features in Typst only apply to certain output file formats. Starting in the PDF[3.14] section, you can find chapters for each of the output formats Typst supports. This is where you find the available format-specific settings and learn what features are available to customize your document for a given format.

3.0.4 Reading the reference

This reference uses a few graphical conventions and labels to let you quickly scan its sections.

`str` These pills indicate that a value is of a particular type. Each type's chapter uses the respective pill as its title. Similar types share a color. For example, all numeric types have the same color.

Element Some functions are labelled as elements. This means that they can be used with set and show rules. Some elements can be located[3.12.3] and used with the `query`[3.12.6] function. Elements generally produce visible output in the document. You may be using elements even if you are not calling functions, as there is dedicated markup for some elements.

Contextual These functions can reason about the contents of your document. They can only be used when *context* is available, for example through a context block. Refer to the Context[3.4] section for more information.

Required Appears on a function parameter if calling the function without that parameter would result in an error.

Positional Appears on a function parameter that is specified without a parameter name and colon. Instead, Typst will use the parameter order to determine which argument is which. Parameters not marked as positional are *named* parameters.

Variadic Appears on function parameters that can be specified multiple times.

Settable Appears on function parameters of element functions that can be customized with a set rule.

3.1 Syntax

Typst is a markup language. This means that you can use simple syntax to accomplish common layout tasks. The lightweight markup syntax is complemented by set and show rules, which let you style your document easily and automatically. All this is backed by a tightly integrated scripting language with built-in and user-defined functions.

3.1.1 Modes

Typst has three syntactical modes: Markup, math, and code. Markup mode is the default in a Typst document, math mode lets you write mathematical formulas, and code mode lets you use Typst's scripting features.

You can switch to a specific mode at any point by referring to the following table:

New mode	Syntax	Example
Code	Prefix the code with #	`Number: #(1 + 2)`
Math	Surround equation with `$..$`	`$-x$ is the opposite of $x$`
Markup	Surround markup with `[..]`	`let name = [*Typst!*]`

Once you have entered code mode with #, you don't need to use further hashes unless you switched back to markup or math mode in between.

3.1.2 Markup

Typst provides built-in markup for the most common document elements. Most of the syntax elements are just shortcuts for a corresponding function. The table below lists all markup that is available and links to the best place to learn more about their syntax and usage.

Name	Example	See
Paragraph break	Blank line	parbreak[3.6.16]
Strong emphasis	`*strong*`	strong[3.6.19]
Emphasis	`_emphasis_`	emph[3.6.7]
Raw text	`` `print(1)` ``	raw[3.7.6]
Link	`https://typst.app/`	link[3.6.11]
Label	`<intro>`	label[3.5.17]
Reference	`@intro`	ref[3.6.18]
Heading	`= Heading`	heading[3.6.10]
Bullet list	`- item`	list[3.6.3]
Numbered list	`+ item`	enum[3.6.12]
Term list	`/ Term: description`	terms[3.6.21]
Math	`$x^2$`	Math[3.8]
Line break	`\`	linebreak[3.7.2]
Smart quote	`'single' or "double"`	smartquote[3.7.8]
Symbol shorthand	`~`, `---`	Symbols[3.9.1]
Code expression	`#rect(width: 1cm)`	Scripting[3.3.1]
Character escape	`Tweet at us \#ad`	Below[3.1.6]
Comment	`/* block */`, `// line`	Below[3.1.5]

3.1.3 Math mode

Math mode is a special markup mode that is used to typeset mathematical formulas. It is entered by wrapping an equation in $ characters. This works both in markup and code. The equation will be typeset

into its own block if it starts and ends with at least one space (e.g. `$ x^2 $`). Inline math can be produced by omitting the whitespace (e.g. `$x^2$`). An overview over the syntax specific to math mode follows:

Name	Example	See
Inline math	`$x^2$`	Math[3.8]
Block-level math	`$ x^2 $`	Math[3.8]
Bottom attachment	`$x_1$`	`attach`[3.8.10]
Top attachment	`$x^2$`	`attach`[3.8.10]
Fraction	`$1 + (a+b)/5$`	`frac`[3.8.16]
Line break	`$x \ y$`	`linebreak`[3.7.2]
Alignment point	`$x &= 2 \ &= 3$`	Math[3.8]
Variable access	`$#x$, $pi$`	Math[3.8]
Field access	`$arrow.r.long$`	Scripting[3.3.6]
Implied multiplication	`$x y$`	Math[3.8]
Symbol shorthand	`$->$, $!=$`	Symbols[3.9.1]
Text/string in math	`$a "is natural"$`	Math[3.8]
Math function call	`$floor(x)$`	Math[3.8]
Code expression	`$#rect(width: 1cm)$`	Scripting[3.3.1]
Character escape	`$x\^2$`	Below[3.1.6]
Comment	`$/* comment */$`	Below[3.1.5]

3.1.4 Code mode

Within code blocks and expressions, new expressions can start without a leading # character. Many syntactic elements are specific to expressions. Below is a table listing all syntax that is available in code mode:

Name	Example	See
None	`none`	`none`[3.5.19]
Auto	`auto`	`auto`[3.5.4]
Boolean	`false`, `true`	`bool`[3.5.5]
Integer	`10`, `0xff`	`int`[3.5.16]
Floating-point number	`3.14`, `1e5`	`float`[3.5.14]
Length	`2pt`, `3mm`, `1em`, ..	`length`[3.10.13]
Angle	`90deg`, `1rad`	`angle`[3.10.3]
Fraction	`2fr`	`fraction`[3.10.9]
Ratio	`50%`	`ratio`[3.10.20]
String	`"hello"`	`str`[3.5.27]
Label	`<intro>`	`label`[3.5.17]
Math	`$x^2$`	Math[3.8]
Raw text	`` `print(1)` ``	`raw`[3.7.6]
Variable access	`x`	Scripting[3.3.2]
Code block	`{ let x = 1; x + 2 }`	Scripting[3.3.2]
Content block	`[*Hello*]`	Scripting[3.3.2]
Parenthesized expression	`(1 + 2)`	Scripting[3.3.2]
Array	`(1, 2, 3)`	Array[3.5.2]
Dictionary	`(a: "hi", b: 2)`	Dictionary[3.5.11]
Unary operator	`-x`	Scripting[3.3.10]
Binary operator	`x + y`	Scripting[3.3.10]
Assignment	`x = 1`	Scripting[3.3.10]

Name	Example	See
Field access	`x.y`	Scripting[3.3.6]
Method call	`x.flatten()`	Scripting[3.3.7]
Function call	`min(x, y)`	Function[3.5.15]
Argument spreading	`min(..nums)`	Arguments[3.5.1]
Unnamed function	`(x, y) => x + y`	Function[3.5.15/6]
Let binding	`let x = 1`	Scripting[3.3.3]
Named function	`let f(x) = 2 * x`	Function[3.5.15]
Set rule	`set text(14pt)`	Styling[3.2.1]
Set-if rule	`set text(..) if ..`	Styling[3.2.1]
Show-set rule	`show heading: set block(..)`	Styling[3.2.2]
Show rule with function	`show raw: it => {..}`	Styling[3.2.2]
Show-everything rule	`show: template`	Styling[3.2.2]
Context expression	`context text.lang`	Context[3.4]
Conditional	`if x == 1 {..} else {..}`	Scripting[3.3.4]
For loop	`for x in (1, 2, 3) {..}`	Scripting[3.3.5]
While loop	`while x < 10 {..}`	Scripting[3.3.5]
Loop control flow	`break, continue`	Scripting[3.3.5]
Return from function	`return x`	Function[3.5.15]
Include module	`include "bar.typ"`	Scripting[3.3.8]
Import module	`import "bar.typ"`	Scripting[3.3.8]
Import items from module	`import "bar.typ": a, b, c`	Scripting[3.3.8]
Comment	`/* block */`, `// line`	Below[3.1.5]

3.1.5 Comments

Comments are ignored by Typst and will not be included in the output. This is useful to exclude old versions or to add annotations. To comment out a single line, start it with `//`:

```
// our data barely supports
// this claim

We show with $p < 0.05$
that the difference is
significant.
```

We show with $p < 0.05$ that the difference is significant.

Comments can also be wrapped between `/*` and `*/`. In this case, the comment can span over multiple lines:

```
Our study design is as follows:
/* Somebody write this up:
   - 1000 participants.
   - 2x2 data design. */
```

Our study design is as follows:

3.1.6 Escape sequences

Escape sequences are used to insert special characters that are hard to type or otherwise have special meaning in Typst. To escape a character, precede it with a backslash. To insert any Unicode codepoint, you can write a hexadecimal escape sequence: `\u{1f600}`. The same kind of escape sequences also work in strings[3.5.27].

```
I got an ice cream for
\$1.50! \u{1f600}
```

I got an ice cream for $1.50! 😀

3.1.7 Identifiers

Names of variables, functions, and so on (*identifiers*) can contain letters, numbers, hyphens (–), and underscores (_). They must start with a letter or an underscore.

More specifically, the identifier syntax in Typst is based on the Unicode Standard Annex #31[A], with two extensions: Allowing _ as a starting character, and allowing both _ and – as continuing characters.

For multi-word identifiers, the recommended case convention is Kebab case[B]. In Kebab case, words are written in lowercase and separated by hyphens (as in `top-edge`). This is especially relevant when developing modules and packages for others to use, as it keeps things predictable.

```
#let kebab-case = [Using hyphen]
#let _schön = "😊"
#let 始料不及 = "😱"
#let π = calc.pi

#kebab-case
#if -π < 0 { _schön } else { 始料不及 }
// -π means -1 * π,
// so it's not a valid identifier
```

Using hyphen 😊

[A] https://www.unicode.org/reports/tr31/
[B] https://en.wikipedia.org/wiki/Letter_case#Kebab_case

3.2 Styling

Typst includes a flexible styling system that automatically applies styling of your choice to your document. With *set rules,* you can configure basic properties of elements. This way, you create most common styles. However, there might not be a built-in property for everything you wish to do. For this reason, Typst further supports *show rules* that can completely redefine the appearance of elements.

3.2.1 Set rules

With set rules, you can customize the appearance of elements. They are written as a function call[3.5.15] to an element function[3.5.15/2] preceded by the `set` keyword (or `#set` in markup). Only optional parameters of that function can be provided to the set rule. Refer to each function's documentation to see which parameters are optional. In the example below, we use two set rules to change the font family[3.7.12/2.1] and heading numbering[3.6.10/5.4].

```
#set heading(numbering: "I.")
#set text(
  font: "New Computer Modern"
)

= Introduction
With set rules, you can style
your document.
```

I. Introduction

With set rules, you can style your document.

A top level set rule stays in effect until the end of the file. When nested inside of a code or content block, it is only in effect until the end of that block. With a block, you can thus restrict the effect of a rule to a particular segment of your document. Below, we use a content block to scope the list styling to one particular list.

```
This list is affected: #[
  #set list(marker: [--])
  - Dash
]

This one is not:
- Bullet
```

This list is affected:
– Dash

This one is not:
• Bullet

Sometimes, you'll want to apply a set rule conditionally. For this, you can use a *set-if* rule.

```
#let task(body, critical: false) = {
  set text(red) if critical
  [- #body]
}

#task(critical: true)[Food today?]
#task(critical: false)[Work deadline]
```

- Food today?
- Work deadline

3.2.2 Show rules

With show rules, you can deeply customize the look of a type of element. The most basic form of show rule is a *show-set rule.* Such a rule is written as the `show` keyword followed by a selector[3.5.25], a colon and then a set rule. The most basic form of selector is an element function[3.5.15/2]. This lets the set rule only apply to the selected element. In the example below, headings become dark blue while all other text stays black.

```
#show heading: set text(navy)

= This is navy-blue
But this stays black.
```

This is navy-blue

But this stays black.

With show-set rules you can mix and match properties from different functions to achieve many different effects. But they still limit you to what is predefined in Typst. For maximum flexibility, you can instead write a *transformational* show rule that defines how to format an element from scratch. To write such a show rule, replace the set rule after the colon with an arbitrary function[3.5.15]. This function receives the element in question and can return arbitrary content. The function is often defined inline as `it => ..` using the unnamed function syntax[3.5.15/6]. The function's parameter is typically named `it` by convention.

The available fields[3.3.6] on the element passed to the function match the parameters of the respective element function. Below, we define a show rule that formats headings for a fantasy encyclopedia.

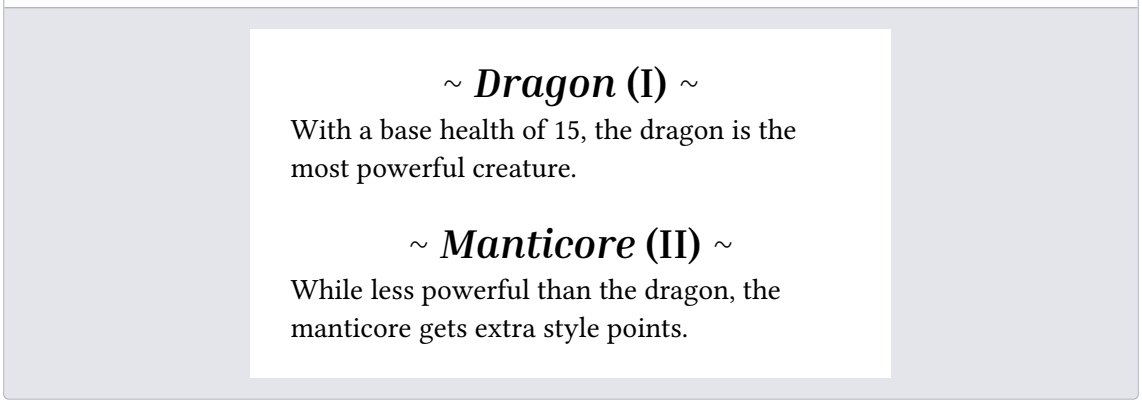
The show rule itself adds tilde characters around the title (these must be escaped with a backslash because otherwise they would indicate a non-breaking space), emphasizes the title with italics, and then displays the heading counter after the title.

For this example, we also wanted center alignment and a different font. While we could've added these set rules into the existing show rule, we instead added them as separate show-set rules. This is good practice because now these rules can still be overridden by later show-set rules in the document, keeping styling composable. In contrast, set rules within a transformational show rule would not be overridable anymore.

```
#set heading(numbering: "(I)")
#show heading: set align(center)
#show heading: set text(font: "Inria Serif")
#show heading: it => block[
  \~
  #emph(it.body)
  #counter(heading).display()
  \~
]

= Dragon
With a base health of 15, the dragon is the most
powerful creature.

= Manticore
While less powerful than the dragon, the manticore
gets extra style points.
```



Like set rules, show rules are in effect until the end of the current block or file.

Instead of a function, the right-hand side of a show rule can also take a literal string or content block that should be directly substituted for the element. And apart from a function, the left-hand side of a show rule can also take a number of other *selectors* that define what to apply the transformation to:

- **Everything:** `show: rest => ..`
  Transform everything after the show rule. This is useful to apply a more complex layout to your whole document without wrapping everything in a giant function call.
- **Text:** `show "Text": ..`
  Style, transform or replace text.

- **Regex:** `show regex("\w+"): ..`
  Select and transform text with a regular expression for even more flexibility. See the documentation of the `regex` type[3.5.23] for details.
- **Function with fields:** `show heading.where(level: 1): ..`
  Transform only elements that have the specified fields. For example, you might want to only change the style of level-1 headings.
- **Label:** `show <intro>: ..`
  Select and transform elements that have the specified label. See the documentation of the `label` type[3.5.17] for more details.

```
#show "Project": smallcaps
#show "badly": "great"

We started Project in 2019
and are still working on it.
Project is progressing badly.
```

We started Project in 2019 and are still working on it. Project is progressing great.

3.3 Scripting

Typst embeds a powerful scripting language. You can automate your documents and create more sophisticated styles with code. Below is an overview over the scripting concepts.

3.3.1 Expressions

In Typst, markup and code are fused into one. All but the most common elements are created with *functions*. To make this as convenient as possible, Typst provides compact syntax to embed a code expression into markup: An expression is introduced with a hash (#) and normal markup parsing resumes after the expression is finished. If a character would continue the expression but should be interpreted as text, the expression can forcibly be ended with a semicolon (;). You can escape a literal # or ; with a backslash[3.1.6].

```
#emph[Hello] \
#emoji.face \
#"hello".len()
```

Hello
😀
5

The example above shows a few of the available expressions, including function calls[3.5.15], field accesses[3.3.6], and method calls[3.3.7]. More kinds of expressions are discussed in the remainder of this chapter. A few kinds of expressions are not compatible with the hash syntax (e.g. binary operator expressions). To embed these into markup, you can use parentheses, as in `#(1 + 2)`.

3.3.2 Blocks

To structure your code and embed markup into it, Typst provides two kinds of *blocks:*

- **Code block:** `{ let x = 1; x + 2 }`
  When writing code, you'll probably want to split up your computation into multiple statements, create some intermediate variables and so on. Code blocks let you write multiple expressions where one is expected. The individual expressions in a code block should be separated by line breaks or semicolons. The output values of the individual expressions in a code block are joined to determine the block's value. Expressions without useful output, like `let` bindings yield `none`, which can be joined with any value without effect.
- **Content block:** `[*Hey* there!]`
  With content blocks, you can handle markup/content as a programmatic value, store it in variables and pass it to functions[3.5.15]. Content blocks are delimited by square brackets and can contain arbitrary markup. A content block results in a value of type content[3.5.8]. An arbitrary number of content blocks can be passed as trailing arguments to functions. That is, `list([A], [B])` is equivalent to `list[A][B]`.

Content and code blocks can be nested arbitrarily. In the example below, `[hello ]` is joined with the output of `a + [ the ] + b` yielding `[hello from the *world*]`.

```
#{
  let a = [from]
  let b = [*world*]
  [hello ]
  a + [ the ] + b
}
```

hello from the **world**

3.3.3 Bindings and Destructuring

As already demonstrated above, variables can be defined with `let` bindings. The variable is assigned the value of the expression that follows the = sign. A valid variable name[3.1.7] may contain –, but cannot start with –. The assignment of a value is optional, if no value is assigned, the variable will be initialized as `none`. The `let` keyword can also be used to create a custom named function[3.5.15/4]. Variables can be accessed for the rest of the containing block (or the rest of the file if there is no containing block).

```
#let name = "Typst"
This is #name's documentation.
It explains #name.

#let my-add(x, y) = x + y
Sum is #my-add(2, 3).
```

This is Typst's documentation. It explains Typst.

Sum is 5.

Let bindings can also be used to destructure arrays[3.5.2] and dictionaries[3.5.11]. In this case, the structure of the left-hand side of the assignment should mirror the array or dictionary: With bindings corresponding by position for arrays and by key name for dictionaries. The `..` operator can be used once in the pattern to collect the remainder of the array's or the dictionary's items.

```
#let (x, y) = (1, 2)
The coordinates are #x, #y.

#let (a, .., b) = (1, 2, 3, 4)
The first element is #a.
The last element is #b.

#let books = (
  Shakespeare: "Hamlet",
  Homer: "The Odyssey",
  Austen: "Persuasion",
)

#let (Austen,) = books
Austen wrote #Austen.

#let (Homer: h) = books
Homer wrote #h.

#let (Homer, ..other) = books
#for (author, title) in other [
  #author wrote #title.
]
```

The coordinates are 1, 2.

The first element is 1. The last element is 4.

Austen wrote Persuasion.

Homer wrote The Odyssey.

Shakespeare wrote Hamlet. Austen wrote Persuasion.

You can use the underscore to discard elements in a destructuring pattern:

```
#let (_, y, _) = (1, 2, 3)
The y coordinate is #y.
```

The y coordinate is 2.

Destructuring also works in argument lists of functions …

```
#let left = (2, 4, 5)
#let right = (3, 2, 6)
#left.zip(right).map(
  ((a,b)) => a + b
)
```

(5, 6, 11)

… and on the left-hand side of normal assignments. This can be useful to swap variables among other things.

```
#{
  let a = 1
  let b = 2
  (a, b) = (b, a)
  [a = #a, b = #b]
}
```

a = 2, b = 1

3.3.4 Conditionals

With a conditional, you can display or compute different things depending on whether some condition is fulfilled. Typst supports `if`, `else if` and `else` expressions. When the condition evaluates to `true`, the conditional yields the value resulting from the if's body. Otherwise, it yields the value resulting from the else's body.

```
#if 1 < 2 [
  This is shown
] else [
  This is not.
]
```

This is shown

Each branch can have a code or content block as its body.

- `if condition {..}`
- `if condition [..]`
- `if condition [..] else {..}`
- `if condition [..] else if condition {..} else [..]`

3.3.5 Loops

With loops, you can repeat content or compute something iteratively. Typst supports two types of loops: `for` and `while` loops. The former iterate over a specified collection whereas the latter iterate as long as a condition stays fulfilled. Just like blocks, loops *join* the results from each iteration into one value.

In the example below, the three sentences created by the for loop join together into a single content value and the length-1 arrays in the while loop join together into one larger array.

```
#for c in "ABC" [
  #c is a letter.
]

#let n = 2
#while n < 10 {
  n = (n * 2) - 1
  (n,)
}
```

A is a letter. B is a letter. C is a letter.

`(3, 5, 9, 17)`

For loops can iterate over a variety of collections:

- `for value in array {..}`
  Iterates over the items in the array[3.5.2]. The destructuring syntax described in Let binding[3.3.3] can also be used here.
- `for pair in dict {..}`
  Iterates over the key-value pairs of the dictionary[3.5.11]. The pairs can also be destructured by using `for (key, value) in dict {..}`. It is more efficient than `for pair in dict.pairs() {..}` because it doesn't create a temporary array of all key-value pairs.
- `for letter in "abc" {..}`
  Iterates over the characters of the string[3.5.27]. Technically, it iterates over the grapheme clusters of the string. Most of the time, a grapheme cluster is just a single codepoint. However, a grapheme cluster could contain multiple codepoints, like a flag emoji.
- `for byte in bytes("😀") {..}`
  Iterates over the bytes[3.5.6], which can be converted from a string[3.5.27] or read[3.13.5] from a file without encoding. Each byte value is an integer[3.5.16] between `0` and `255`.

To control the execution of the loop, Typst provides the `break` and `continue` statements. The former performs an early exit from the loop while the latter skips ahead to the next iteration of the loop.

```
#for letter in "abc nope" {
  if letter == " " {
    break
  }

  letter
}
```

abc

The body of a loop can be a code or content block:

- `for .. in collection {..}`
- `for .. in collection [..]`
- `while condition {..}`
- `while condition [..]`

3.3.6 Fields

You can use *dot notation* to access fields on a value. For values of type `content`[3.5.8], you can also use the `fields`[3.5.8/3.4] function to list the fields.

The value in question can be either:
- a dictionary[3.5.11] that has the specified key,
- a symbol[3.5.28] that has the specified modifier,
- a module[3.5.18] containing the specified definition,
- content[3.5.8] consisting of an element that has the specified field. The available fields match the arguments of the element function[3.5.15/2] that were given when the element was constructed.

```
#let it = [= Heading]
#it.body \
#it.depth \
#it.fields()

#let dict = (greet: "Hello")
#dict.greet \
#emoji.face
```

Heading
1
`(depth: 1, body: [Heading])`
Hello
😀

3.3.7 Methods

A *method call* is a convenient way to call a function that is scoped to a value's type[3.5.31]. For example, we can call the `str.len`[3.5.27/4.1] function in the following two equivalent ways:

```
#str.len("abc") is the same as
#"abc".len()
```

3 is the same as 3

The structure of a method call is `value.method(..args)` and its equivalent full function call is `type(value).method(value, ..args)`. The documentation of each type lists its scoped functions. You cannot currently define your own methods.

```
#let values = (1, 2, 3, 4)
#values.pop() \
#values.len() \

#("a, b, c"
    .split(", ")
    .join[ --- ])
```

4
3
a — b — c
3 is the same as 3

```
#"abc".len() is the same as
#str.len("abc")
```

There are a few special functions that modify the value they are called on (e.g. `array.push`[3.5.2/3.5]). These functions *must* be called in method form. In some cases, when the method is only called for its side effect, its return value should be ignored (and not participate in joining). The canonical way to discard a value is with a let binding: `let _ = array.remove(1)`.

3.3.8 Modules

You can split up your Typst projects into multiple files called *modules*. A module can refer to the content and definitions of another module in multiple ways:

- **Including:** `include "bar.typ"`
  Evaluates the file at the path[3.5.21] `bar.typ` and returns the resulting content[3.5.8].
- **Import:** `import "bar.typ"`
  Evaluates the file at the path[3.5.21] `bar.typ` and inserts the resulting module[3.5.18] into the current scope as `bar` (filename without extension). You can use the `as` keyword to rename the imported module: `import "bar.typ" as baz`. You can import nested items using dot notation: `import "bar.typ": baz.a`.
- **Import items:** `import "bar.typ": a, b`
  Evaluates the file at the path[3.5.21] `bar.typ`, extracts the values of the variables `a` and `b` (that need to be defined in `bar.typ`, e.g. through `let` bindings) and defines them in the current file. Replacing `a, b` with `*` loads all variables defined in a module. You can use the `as` keyword to rename the individual items: `import "bar.typ": a as one, b as two`

Instead of a string or path[3.5.21], you can also use a module value[3.5.18], as shown in the following example:

```
#import emoji: face
#face.grin
```

😀

3.3.9 Packages

To reuse building blocks across projects, you can also create and import Typst *packages*. A package import is specified as a triple of a namespace, a name, and a version.

```
#import "@preview/example:0.1.0": add
#add(2, 7)
```

9

The `preview` namespace contains packages shared by the community. You can find all available community packages on Typst Universe[A].

If you are using Typst locally, you can also create your own system-local packages. For more details on this, see the package repository[B].

3.3.10 Operators

The following table lists all available unary and binary operators with effect, arity (unary, binary) and precedence level (higher binds stronger). Some operations, such as modulus[3.5.7/1.39], do not have a special syntax and can be achieved using functions from the `calc`[3.5.7] module.

Operator	Effect	Arity	Precedence
`-`	Negation	Unary	7

[A] https://typst.app/universe
[B] https://github.com/typst/packages

Operator	Effect	Arity	Precedence
`+`	No effect (exists for symmetry)	Unary	7
`*`	Multiplication	Binary	6
`/`	Division	Binary	6
`+`	Addition	Binary	5
`-`	Subtraction	Binary	5
`==`	Check equality	Binary	4
`!=`	Check inequality	Binary	4
`<`	Check less-than	Binary	4
`<=`	Check less-than or equal	Binary	4
`>`	Check greater-than	Binary	4
`>=`	Check greater-than or equal	Binary	4
`in`	Check if in collection	Binary	4
`not in`	Check if not in collection	Binary	4
`not`	Logical “not”	Unary	3
`and`	Short-circuiting logical “and”	Binary	3
`or`	Short-circuiting logical “or”	Binary	2
`=`	Assignment	Binary	1
`+=`	Add-Assignment	Binary	1
`-=`	Subtraction-Assignment	Binary	1
`*=`	Multiplication-Assignment	Binary	1
`/=`	Division-Assignment	Binary	1

3.4 Context

Sometimes, we want to create content that reacts to its location in the document. This could be a localized phrase that depends on the configured text language or something as simple as a heading number which prints the right value based on how many headings came before it. However, Typst code isn't directly aware of its location in the document. Some code at the beginning of the source text could yield content that ends up at the back of the document.

To produce content that is reactive to its surroundings, we must thus specifically instruct Typst: We do this with the `context` keyword, which precedes an expression and ensures that it is computed with knowledge of its environment. In return, the context expression itself ends up opaque. We cannot directly access whatever results from it in our code, precisely because it is contextual: There is no one correct result, there may be multiple results in different places of the document. For this reason, everything that depends on the contextual data must happen inside of the context expression.

Aside from explicit context expressions, context is also established implicitly in some places that are also aware of their location in the document: Show rules[3.2.2] provide context[A] and numberings in the outline, for instance, also provide the proper context to resolve counters.

3.4.1 Style context

With set rules, we can adjust style properties for parts or the whole of our document. We cannot access these without a known context, as they may change throughout the course of the document. When context is available, we can retrieve them simply by accessing them as fields on the respective element function.

```
#set text(lang: "de")
#context text.lang
```

de

As explained above, a context expression is reactive to the different environments it is placed into. In the example below, we create a single context expression, store it in the `value` variable and use it multiple times. Each use properly reacts to the current surroundings.

```
#let value = context text.lang
#value

#set text(lang: "de")
#value

#set text(lang: "fr")
#value
```

en

de

fr

Crucially, upon creation, `value` becomes opaque content[3.5.8] that we cannot peek into. It can only be resolved when placed somewhere because only then the context is known. The body of a context expression may be evaluated zero, one, or multiple times, depending on how many different places it is put into.

3.4.2 Location context

We've already seen that context gives us access to set rule values. But it can do more: It also lets us know *where* in the document we currently are, relative to other elements, and absolutely on the pages. We can use this information to create very flexible interactions between different document parts. This underpins features like heading numbering, the table of contents, or page headers dependent on section headings.

Some functions like `counter.get`[3.12.1/11.1] implicitly access the current location. In the example below, we want to retrieve the value of the heading counter. Since it changes throughout the document, we need to first enter a context expression. Then, we use `get` to retrieve the counter's current value. This function accesses the current location from the context to resolve the counter value. Counters have multiple levels and `get` returns an array with the resolved numbers. Thus, we get the following result:

[A] Currently, all show rules provide style context, but only show rules on locatable elements provide a location context.

```
#set heading(numbering: "1.")

= Introduction
#lorem(5)

#context counter(heading).get()

= Background
#lorem(5)

#context counter(heading).get()
```

1. Introduction
Lorem ipsum dolor sit amet.

(1,)

2. Background
Lorem ipsum dolor sit amet.

(2,)

For more flexibility, we can also use the here[3.12.2] function to directly extract the current location[3.12.4] from the context. The example below demonstrates this:

- We first have `counter(heading).get()`, which resolves to `(2,)` as before.
- We then use the more powerful `counter.at`[3.12.1/11.3] with here[3.12.2], which in combination is equivalent to `get`, and thus get `(2,)`.
- Finally, we use `at` with a label[3.5.17] to retrieve the value of the counter at a *different* location in the document, in our case that of the introduction heading. This yields `(1,)`. Typst's context system gives us time travel abilities and lets us retrieve the values of any counters and states at *any* location in the document.

```
#set heading(numbering: "1.")

= Introduction <intro>
#lorem(5)

= Background <back>
#lorem(5)

#context [
  #counter(heading).get() \
  #counter(heading).at(here()) \
  #counter(heading).at(<intro>)
]
```

1. Introduction
Lorem ipsum dolor sit amet.

2. Background
Lorem ipsum dolor sit amet.

(2,)
(2,)
(1,)

As mentioned before, we can also use context to get the physical position of elements on the pages. We do this with the `locate`[3.12.3] function, which works similarly to `counter.at`: It takes a location or other selector[3.5.25] that resolves to a unique element (could also be a label) and returns the position on the pages for that element.

```
Background is at: \
#context locate(<back>).position()

= Introduction <intro>
#lorem(5)
#pagebreak()

= Background <back>
#lorem(5)
```

Background is at:
(page: 2, x: 15pt, y: 15pt)

Introduction
Lorem ipsum dolor sit amet.

Background
Lorem ipsum dolor sit amet.

There are other functions that make use of the location context, most prominently query[3.12.6]. Take a look at the introspection[3.12] category for more details on those.

3.4.3 Nested contexts

Context is also accessible from within function calls nested in context blocks. In the example below, `foo` itself becomes a contextual function, just like `to-absolute`[3.10.13/3.5] is.

```
#let foo() = 1em.to-absolute()
#context {
  foo() == text.size
}
```

```
true
```

Context blocks can be nested. Contextual code will then always access the innermost context. The example below demonstrates this: The first `text.lang` will access the outer context block's styles and as such, it will **not** see the effect of `set text(lang: "fr")`. The nested context block around the second `text.lang`, however, starts after the set rule and will thus show its effect.

```
#set text(lang: "de")
#context [
  #set text(lang: "fr")
  #text.lang \
  #context text.lang
]
```

```
de
fr
```

You might wonder why Typst ignores the French set rule when computing the first `text.lang` in the example above. The reason is that, in the general case, Typst cannot know all the styles that will apply as set rules can be applied to content after it has been constructed. Below, `text.lang` is already computed when the template function is applied. As such, it cannot possibly be aware of the language change to French in the template.

```
#let template(body) = {
  set text(lang: "fr")
  upper(body)
}

#set text(lang: "de")
#context [
  #show: template
  #text.lang \
  #context text.lang
]
```

```
DE
FR
```

The second `text.lang`, however, *does* react to the language change because evaluation of its surrounding context block is deferred until the styles for it are known. This illustrates the importance of picking the right insertion point for a context to get access to precisely the right styles.

The same also holds true for the location context. Below, the first `c.display()` call will access the outer context block and will thus not see the effect of `c.update(2)` while the second `c.display()` accesses the inner context and will thus see it.

```
#let c = counter("mycounter")
#c.update(1)
#context [
  #c.update(2)
  #c.display() \
  #context c.display()
]
```

```
1
2
```

3.4.4 Compiler iterations

To resolve contextual interactions, the Typst compiler processes your document multiple times. For instance, to resolve a `locate` call, Typst first provides a placeholder position, layouts your document and

then recompiles with the known position from the finished layout. The same approach is taken to resolve counters, states, and queries. In certain cases, Typst may even need more than two iterations to resolve everything. While that's sometimes a necessity, it may also be a sign of misuse of contextual functions (e.g. of state[3.12.7/4]). If Typst cannot resolve everything within five attempts, it will stop and output the warning "document did not converge within five attempts."

A very careful reader might have noticed that not all of the functions presented above actually make use of the current location. While `counter(heading).get()` definitely depends on it, `counter(heading).at(<intro>)`, for instance, does not. However, it still requires context. While its value is always the same *within* one compilation iteration, it may change over the course of multiple compiler iterations. If one could call it directly at the top level of a module, the whole module and its exports could change over the course of multiple compiler iterations, which would not be desirable.

3.5 Foundations

Foundational types and functions.

Here, you'll find documentation for basic data types like integers[3.5.16] and strings[3.5.27] as well as details about core computational functions.

Definitions

3.5.1 Arguments Captured arguments to a function. . . . 57
3.5.2 Array A sequence of values. . . . 60
3.5.3 Assert Ensures that a condition is fulfilled. . . . 70
3.5.4 Auto A value that indicates a smart default. . . . 72
3.5.5 Boolean A type with two states. . . . 73
3.5.6 Bytes A sequence of bytes. . . . 74
3.5.7 Calculation Module for calculations and processing of numeric values. . . . 76
3.5.8 Content A piece of document content. . . . 91
3.5.9 Datetime Represents a date, a time, or a combination of both. . . . 93
3.5.10 Decimal A fixed-point decimal number type. . . . 98
3.5.11 Dictionary A map from string keys to values. . . . 100
3.5.12 Duration Represents a positive or negative span of time. . . . 103
3.5.13 Evaluate Evaluates a string as Typst code. . . . 105
3.5.14 Float A floating-point number. . . . 106
3.5.15 Function A mapping from argument values to a return value. . . . 109
3.5.16 Integer An integer: a positive whole number, a negative whole number, or zero. . . . 112
3.5.17 Label A label for an element. . . . 117
3.5.18 Module A collection of variables and functions that are commonly related to a single theme. . . . 118
3.5.19 None A value that indicates the absence of any other value. . . . 119
3.5.20 Panic Fails with an error. . . . 120
3.5.21 Path A file system path. . . . 121
3.5.22 Plugin Loads a WebAssembly module. . . . 123
3.5.23 Regex A regular expression. . . . 126
3.5.24 Representation Returns the string representation of a value. . . . 127
3.5.25 Selector A filter for selecting elements within the document. . . . 128
3.5.26 Standard Library A module that contains all globally accessible items. . . . 131
3.5.27 String A sequence of Unicode codepoints. . . . 132
3.5.28 Symbol A Unicode symbol. . . . 139
3.5.29 System Module for system interactions. . . . 140
3.5.30 Target Returns the current export target. . . . 141
3.5.31 Type Describes a kind of value. . . . 142
3.5.32 Version A version with an arbitrary number of components. . . . 143

3.5.1 `arguments`

Captured arguments to a function.

Arguments are either *positional* or *named,* and can be accessed through the `pos`[3.5.1/4.3], `named`[3.5.1/4.4], and `at`[3.5.1/4.2] methods.

Additionally, named arguments can be accessed with field syntax[3.5.1/4.2] similar to dictionaries[3.5.11].

/1 Argument Sinks

Like built-in functions, custom functions can also take a variable number of arguments. You can specify an *argument sink* which collects all excess arguments as `..sink`. The resulting `sink` value is of the `arguments` type. It exposes methods to access the positional and named arguments.

```
#let format(title, ..authors) = {
  let by = authors
    .pos()
    .join(", ", last: " and ")

  [*#title* \ _Written by #by;_]
}

#format("ArtosFlow", "Jane", "Joe")
```

ArtosFlow
Written by Jane and Joe

/2 Spreading

Inversely to an argument sink, you can *spread* arguments, arrays and dictionaries into a function call with the `..spread` operator:

```
#let array = (2, 3, 5)
#calc.min(..array)
#let dict = (fill: blue)
#text(..dict)[Hello]
```

2 Hello

/3 Constructor

Construct spreadable arguments in place.

This function behaves like `let args(..sink) = sink`.

```
#let args = arguments(stroke: red, inset: 1em, [Body])
#box(..args)
```

Body

```
arguments(.. any ) → arguments
```

/3.1 **`arguments`** `any` *Required* *Positional* *Variadic*

The arguments to construct.

/4 Definitions

/4.1 `len`

The number of arguments, positional or named.

```
self.len() → int
```

/4.2 at

Returns the positional argument at the specified index, or the named argument with the specified name.

If the key is an integer[3.5.16], this is equivalent to first calling pos[3.5.1/4.3] and then array.at[3.5.2/3.4]. If it is a string[3.5.27], this is equivalent to first calling named[3.5.1/4.4] and then dictionary.at[3.5.11/3.2].

Named arguments can also be accessed with field syntax (e.g. arguments(key: 42).key) if no default is needed. Unlike dictionaries[3.5.11], fields on arguments cannot be modified.

```
self.at(
  int str,
  default: any,
) → any
```

/4.2.1 key int or str *Required* *Positional*

The index or name of the argument to get.

/4.2.2 default any

A default value to return if the key is invalid.

/4.3 pos

Returns the captured positional arguments as an array.

```
self.pos() → array
```

/4.4 named

Returns the captured named arguments as a dictionary.

```
self.named() → dictionary
```

/4.5 filter

Produces a new arguments with only the arguments for which the value passes the test.

```
#{
  arguments(-1, a: 0, b: 1, 2)
    .filter(v => v > 0)
}
```

```
arguments(b: 1, 2)
```

```
self.filter(function) → arguments
```

/4.5.1 test function *Required* *Positional*

The function to apply to each value. Must return a boolean.

/4.6 map

Produces a new arguments by transforming each argument value with the passed function.

```
#{
  arguments(0, a: 1, 2)
    .map(v => v + 1)
}
```

```
arguments(1, a: 2, 3)
```

```
self.map(function) → arguments
```

/4.6.1 **mapper** `function` *Required* *Positional*

The function to apply to each value.

3.5.2 `array`

A sequence of values.

You can construct an array by enclosing a comma-separated sequence of values in parentheses. The values do not have to be of the same type.

You can access and update array items with the `.at()` method. Indices are zero-based and negative indices wrap around to the end of the array. You can iterate over an array using a for loop[3.3.5]. Arrays can be added together with the + operator, joined together[3.3.2] and multiplied with integers.

Note: An array of length one needs a trailing comma, as in `(1,)`. This is to disambiguate from a simple parenthesized expressions like `(1 + 2) * 3`. An empty array is written as `()`.

/1 Example

```
#let values = (1, 7, 4, -3, 2)

#values.at(0) \
#(values.at(0) = 3)
#values.at(-1) \
#values.find(calc.even) \
#values.filter(calc.odd) \
#values.map(calc.abs) \
#values.rev() \
#(1, (2, 3)).flatten() \
#(("A", "B", "C")
    .join(", ", last: " and "))
```

```
1
2
4
(3, 7, -3)
(3, 7, 4, 3, 2)
(2, -3, 4, 7, 3)
(1, 2, 3)
A, B and C
```

/2 Constructor

Converts a value to an array.

Note that this function is only intended for conversion of a collection-like value to an array, not for creation of an array from individual items. Use the array syntax `(1, 2, 3)` (or `(1,)` for a single-element array) instead.

```
#let hi = "Hello 😃"
#array(bytes(hi))
```

```
(72, 101, 108, 108, 111, 32, 240, 159,
152, 131)
```

```
array( bytes  array  version ) → array
```

/2.1 **`value`** `bytes` or `array` or `version` *Required* *Positional*

The value that should be converted to an array.

/3 Definitions

/3.1 `len`

The number of values in the array.

```
self.len() → int
```

/3.2 `first`

Returns the first item in the array. May be used on the left-hand side of an assignment. Returns the default value if the array is empty or fails with an error if no default value was specified.

```
self.first(default: any ) → any
```

/3.2.1 **default** any

A default value to return if the array is empty.

/3.3 last

Returns the last item in the array. May be used on the left-hand side of an assignment. Returns the default value if the array is empty or fails with an error if no default value was specified.

```
self.last(default: any ) → any
```

/3.3.1 **default** any

A default value to return if the array is empty.

/3.4 at

Returns the item at the specified index in the array. May be used on the left-hand side of an assignment. Returns the default value if the index is out of bounds or fails with an error if no default value was specified.

```
self.at(
  int ,
  default: any ,
) → any
```

/3.4.1 **index** int *Required* *Positional*

The index at which to retrieve the item. If negative, indexes from the back.

/3.4.2 **default** any

A default value to return if the index is out of bounds.

/3.5 push

Adds a value to the end of the array.

```
self.push( any ) → none
```

/3.5.1 **value** any *Required* *Positional*

The value to insert at the end of the array.

/3.6 pop

Removes the last item from the array and returns it. Fails with an error if the array is empty.

```
self.pop() → any
```

/3.7 insert

Inserts a value into the array at the specified index, shifting all subsequent elements to the right. Fails with an error if the index is out of bounds.

To replace an element of an array, use at[3.5.2/3.4].

```
self.insert(
  int ,
  any ,
) → none
```

/3.7.1 **index** `int` *Required* *Positional*

The index at which to insert the item. If negative, indexes from the back.

/3.7.2 **value** `any` *Required* *Positional*

The value to insert into the array.

/3.8 remove

Removes the value at the specified index from the array and return it.

```
self.remove(
  int,
  default: any,
) → any
```

/3.8.1 **index** `int` *Required* *Positional*

The index at which to remove the item. If negative, indexes from the back.

/3.8.2 **default** `any`

A default value to return if the index is out of bounds.

/3.9 slice

Extracts a subslice of the array. Fails with an error if the start or end index is out of bounds.

```
self.slice(
  int,
  none int,
  count: int,
) → array
```

/3.9.1 **start** `int` *Required* *Positional*

The start index (inclusive). If negative, indexes from the back.

/3.9.2 **end** `none` or `int` *Positional* Default: `none`

The end index (exclusive). If omitted, the whole slice until the end of the array is extracted. If negative, indexes from the back.

/3.9.3 **count** `int`

The number of items to extract. This is equivalent to passing `start + count` as the `end` position. Mutually exclusive with `end`.

/3.10 contains

Whether the array contains the specified value.

This method also has dedicated syntax: You can write `2 in (1, 2, 3)` instead of `(1, 2, 3).contains(2)`.

```
self.contains(any) → bool
```

/3.10.1 **value** `any` *Required* *Positional*

The value to search for.

/3.11 find

Searches for an item for which the given function returns `true` and returns the first match or `none` if there is no match.

```
self.find(function) → none any
```

/3.11.1 **searcher** function *Required* *Positional*

The function to apply to each item. Must return a boolean.

/3.12 **position**

Searches for an item for which the given function returns `true` and returns the index of the first match or `none` if there is no match.

```
self.position(function) → none int
```

/3.12.1 **searcher** function *Required* *Positional*

The function to apply to each item. Must return a boolean.

```
#let values = (1, 7, 4, 6, 9)
#values.position(x => calc.even(x)) \
// Or equivalently:
#values.position(calc.even)
```

```
2
2
```

/3.13 **range**

Create an array consisting of a sequence of numbers.

If you pass just one positional parameter, it is interpreted as the `end` of the range. If you pass two, they describe the `start` and `end` of the range.

This function is available both in the array function's scope and globally.

```
#range(5) \
#range(2, 5) \
#range(20, step: 4) \
#range(21, step: 4) \
#range(5, 2, step: -1)
```

```
(0, 1, 2, 3, 4)
(2, 3, 4)
(0, 4, 8, 12, 16)
(0, 4, 8, 12, 16, 20)
(5, 4, 3)
```

```
array.range(
  int,
  int,
  inclusive: bool,
  step: int,
) → array
```

/3.13.1 **start** int *Positional* Default: `0`

The start of the range (inclusive).

/3.13.2 **end** int *Required* *Positional*

The end of the range.

/3.13.3 **inclusive** bool Default: `false`

Whether `end` is inclusive.

```
#range(0, inclusive: true) \
#range(7, 10, inclusive: true) \
#range(-8, -4, inclusive: true) \
#range(-6, step: -2, inclusive: true)
```

```
(0,)
(7, 8, 9, 10)
(-8, -7, -6, -5, -4)
(0, -2, -4, -6)
```

/3.13.4 **step** int Default: 1

The distance between the generated numbers.

/3.14 filter

Produces a new array with only the items from the original one for which the given function returns `true`.

```
self.filter(function) → array
```

/3.14.1 **test** function *Required* *Positional*

The function to apply to each item. Must return a boolean.

/3.15 map

Produces a new array in which all items from the original one were transformed with the given function.

```
self.map(function) → array
```

/3.15.1 **mapper** function *Required* *Positional*

The function to apply to each item.

/3.16 enumerate

Returns a new array with the values alongside their indices.

The returned array consists of `(index, value)` pairs in the form of length-2 arrays. These can be destructured[3.3.3] with a let binding or for loop.

```
#for (i, value) in ("A", "B", "C").enumerate() {
  [#i: #value \ ]
}

#("A", "B", "C").enumerate(start: 1)
```

0: A
1: B
2: C

```
((1, "A"), (2, "B"), (3, "C"))
```

```
self.enumerate(start: int) → array
```

/3.16.1 **start** int Default: 0

The index returned for the first pair of the returned list.

/3.17 zip

Zips the array with other arrays.

Returns an array of arrays, where the `i`th inner array contains all the `i`th elements from each original array.

If the arrays to be zipped have different lengths, they are zipped up to the last element of the shortest array and all remaining elements are ignored.

This function is variadic, meaning that you can zip multiple arrays together at once: `(1, 2).zip(("A", "B"), (10, 20))` yields `((1, "A", 10), (2, "B", 20))`.

```
self.zip(
  exact: bool,
```

```
  ..array,
) → array
```

/3.17.1 **exact** bool Default: false

Whether all arrays have to have the same length. For example, `(1, 2).zip((1, 2, 3), exact: true)` produces an error.

/3.17.2 **others** array *Required* *Positional* *Variadic*

The arrays to zip with.

/3.18 fold

Folds all items into a single value using an accumulator function.

```
#let array = (1, 2, 3, 4)
#array.fold(0, (acc, x) => acc + x)
```

10

```
self.fold(
  any,
  function,
) → any
```

/3.18.1 **init** any *Required* *Positional*

The initial value to start with.

/3.18.2 **folder** function *Required* *Positional*

The folding function. Must have two parameters: One for the accumulated value and one for an item.

/3.19 sum

Sums all items (works for all types that can be added).

```
self.sum(default: any) → any
```

/3.19.1 **default** any

What to return if the array is empty. Must be set if the array can be empty.

/3.20 product

Calculates the product of all items (works for all types that can be multiplied).

```
self.product(default: any) → any
```

/3.20.1 **default** any

What to return if the array is empty. Must be set if the array can be empty.

/3.21 any

Whether the given function returns `true` for any item in the array.

```
self.any(function) → bool
```

/3.21.1 **test** function *Required* *Positional*

The function to apply to each item. Must return a boolean.

/3.22 all

Whether the given function returns `true` for all items in the array.

```
self.all(function) → bool
```

/3.22.1 **test** function *Required* *Positional*

The function to apply to each item. Must return a boolean.

/3.23 flatten

Combine all nested arrays into a single flat one.

```
self.flatten() → array
```

/3.24 rev

Return a new array with the same items, but in reverse order.

```
self.rev() → array
```

/3.25 split

Split the array at occurrences of the specified value.

```
#(1, 1, 2, 3, 2, 4, 5).split(2)
```

```
((1, 1), (3,), (4, 5))
```

```
self.split(any) → array
```

/3.25.1 **at** any *Required* *Positional*

The value to split at.

/3.26 join

Combine all items in the array into one.

```
self.join(
  none any,
  last: any,
  default: none any,
) → any
```

/3.26.1 **separator** none or any *Positional* Default: none

A value to insert between each item of the array.

/3.26.2 **last** any

An alternative separator between the last two items.

/3.26.3 **default** none or any Default: none

What to return if the array is empty.

/3.27 intersperse

Returns an array with a copy of the separator value placed between adjacent elements.

```
#("A", "B", "C").intersperse("-")
```

```
("A", "-", "B", "-", "C")
```

```
self.intersperse( any ) → array
```

/3.27.1 **separator** any *Required* *Positional*

The value that will be placed between each adjacent element.

/3.28 chunks

Splits an array into non-overlapping chunks, starting at the beginning, ending with a single remainder chunk.

All chunks but the last have `chunk-size` elements. If `exact` is set to `true`, the remainder is dropped if it contains less than `chunk-size` elements.

```
#let array = (1, 2, 3, 4, 5, 6, 7, 8)
#array.chunks(3) \
#array.chunks(3, exact: true)
```

```
((1, 2, 3), (4, 5, 6), (7, 8))
((1, 2, 3), (4, 5, 6))
```

```
self.chunks(
  int ,
  exact: bool ,
) → array
```

/3.28.1 **chunk-size** int *Required* *Positional*

How many elements each chunk may at most contain.

/3.28.2 **exact** bool Default: `false`

Whether to discard the remainder if its size is less than `chunk-size`.

/3.29 windows

Returns sliding windows of `window-size` elements over an array.

If the array length is less than `window-size`, this will return an empty array.

```
#let array = (1, 2, 3, 4, 5, 6, 7, 8)
#array.windows(5)
```

```
(
  (1, 2, 3, 4, 5),
  (2, 3, 4, 5, 6),
  (3, 4, 5, 6, 7),
  (4, 5, 6, 7, 8),
)
```

```
self.windows( int ) → array
```

/3.29.1 **window-size** int *Required* *Positional*

How many elements each window will contain.

/3.30 sorted

Return a sorted version of this array, optionally by a given key function. The sorting algorithm used is stable.

Returns an error if a pair of values selected for comparison could not be compared, or if the key or comparison function (if given) yield an error.

To sort according to multiple criteria at once, e.g. in case of equality between some criteria, the key function can return an array. The results are in lexicographic order.

```
#let array = (
  (a: 2, b: 4),
  (a: 1, b: 5),
  (a: 2, b: 3),
)
#array.sorted(key: it => (it.a, it.b))
```

```
((a: 1, b: 5), (a: 2, b: 3), (a: 2, b: 4))
```

```
self.sorted(
  key: function,
  by: function,
) → array
```

/3.30.1 **key** `function`

If given, applies this function to each element in the array to determine the keys to sort by.

/3.30.2 **by** `function`

If given, uses this function to compare every two elements in the array.

The function will receive two elements in the array for comparison, and should return a boolean indicating their order: `true` indicates that the elements are in order, while `false` indicates that they should be swapped. To keep the sort stable, if the two elements are equal, the function should return `true`.

If this function does not order the elements properly (e.g., by returning `false` for both `(x, y)` and `(y, x)`, or for `(x, x)`), the resulting array will be in unspecified order.

When used together with `key`, `by` will be passed the keys instead of the elements.

```
#(
  "sorted",
  "by",
  "decreasing",
  "length",
).sorted(
  key: s => s.len(),
  by: (l, r) => l >= r,
)
```

```
("decreasing", "sorted", "length", "by")
```

/3.31 **dedup**

Deduplicates all items in the array.

Returns a new array with all duplicate items removed. Only the first element of each duplicate is kept.

```
#(3, 3, 1, 2, 3).dedup()
```

```
(3, 1, 2)
```

```
self.dedup(key: function) → array
```

/3.31.1 **key** `function`

If given, applies this function to each element in the array to determine the keys to deduplicate by.

```
#("apple", "banana", " apple ").dedup(key: s => s.trim())
```

```
("apple", "banana")
```

/3.32 to-dict

Converts an array of pairs into a dictionary. The first value of each pair is the key, the second the value.

If the same key occurs multiple times, the last value is selected.

```
#(
  ("apples", 2),
  ("peaches", 3),
  ("apples", 5),
).to-dict()
```

```
(apples: 5, peaches: 3)
```

```
self.to-dict() → dictionary
```

/3.33 reduce

Reduces the elements to a single one, by repeatedly applying a reducing operation.

If the array is empty, returns none, otherwise, returns the result of the reduction.

The reducing function is a closure with two arguments: an “accumulator”, and an element.

For arrays with at least one element, this is the same as array.fold[3.5.2/3.18] with the first element of the array as the initial accumulator value, folding every subsequent element into it.

```
#let array = (2, 1, 4, 3)
#array.reduce((acc, x) => calc.max(acc, x))
```

```
4
```

```
self.reduce(function) → any
```

/3.33.1 reducer function *Required* *Positional*

The reducing function. Must have two parameters: One for the accumulated value and one for an item.

3.5.3 assert

Ensures that a condition is fulfilled.

Fails with an error if the condition is not fulfilled. Does not produce any output in the document.

If you wish to test equality between two values, see `assert.eq`[3.5.3/3.1] and `assert.ne`[3.5.3/3.2].

/1 Example

```
#assert(1 < 2, message: "math broke")
```

/2 Parameters

```
assert(
  bool,
  message: str,
) → none
```

/2.1 condition `bool` *Required* *Positional*

The condition that must be true for the assertion to pass.

/2.2 message `str`

The error message when the assertion fails.

/3 Definitions

/3.1 eq

Ensures that two values are equal.

Fails with an error if the first value is not equal to the second. Does not produce any output in the document.

```
#assert.eq(10, 10)
```

```
assert.eq(
  any,
  any,
  message: str,
) → none
```

/3.1.1 left `any` *Required* *Positional*

The first value to compare.

/3.1.2 right `any` *Required* *Positional*

The second value to compare.

/3.1.3 message `str`

An optional message to display on error instead of the representations of the compared values.

/3.2 ne

Ensures that two values are not equal.

Fails with an error if the first value is equal to the second. Does not produce any output in the document.

```
#assert.ne(3, 4)
```

```
assert.ne(
  any,
  any,
  message: str,
) → none
```

/3.2.1 **left** `any` *Required* *Positional*

The first value to compare.

/3.2.2 **right** `any` *Required* *Positional*

The second value to compare.

/3.2.3 **message** `str`

An optional message to display on error instead of the representations of the compared values.

3.5.4 auto

A value that indicates a smart default.

The auto type has exactly one value: `auto`.

Parameters that support the `auto` value have some smart default or contextual behaviour. A good example is the text direction[3.7.12/2.19] parameter. Setting it to `auto` lets Typst automatically determine the direction from the text language[3.7.12/2.16].

3.5.5 `bool`

A type with two states.

The boolean type has two values: `true` and `false`. It denotes whether something is active or enabled.

/1 **Example**

```
#false \
#true \
#(1 < 2)
```

```
false
true
true
```

3.5.6 bytes

A sequence of bytes.

This is conceptually similar to an array of integers[3.5.16] between `0` and `255`, but represented much more efficiently. You can iterate over it using a for loop[3.3.5].

You can convert
- a string[3.5.27] or an array[3.5.2] of integers to bytes with the `bytes`[3.5.6] constructor
- bytes to a string with the `str`[3.5.27] constructor, with UTF-8 encoding
- bytes to an array of integers with the `array`[3.5.2] constructor

When reading[3.13.5] data from a file, you can decide whether to load it as a string or as raw bytes.

```
#bytes((123, 160, 22, 0)) \
#bytes("Hello 😃")

#let data = read(
  "rhino.png",
  encoding: none,
)

// Magic bytes.
#array(data.slice(0, 4)) \
#str(data.slice(1, 4))
```

```
bytes(4)
bytes(10)
```

```
(137, 80, 78, 71)
PNG
```

/1 Constructor

Converts a value to bytes.

- Strings are encoded in UTF-8.
- Arrays of integers between `0` and `255` are converted directly. The dedicated byte representation is much more efficient than the array representation and thus typically used for large byte buffers (e.g. image data).

```
#bytes("Hello 😃") \
#bytes((123, 160, 22, 0))
```

```
bytes(10)
bytes(4)
```

```
bytes( str  bytes  array ) → bytes
```

/1.1 value `str` or `bytes` or `array` *Required* *Positional*

The value that should be converted to bytes.

/2 Definitions

/2.1 len

The length in bytes.

```
self.len() → int
```

/2.2 at

Returns the byte at the specified index. Returns the default value if the index is out of bounds or fails with an error if no default value was specified.

```
self.at(
  int ,
```

```
  default: any,
) → any
```

/2.2.1 **index** `int` *Required* *Positional*

The index at which to retrieve the byte.

/2.2.2 **default** `any`

A default value to return if the index is out of bounds.

/2.3 slice

Extracts a subslice of the bytes. Fails with an error if the start or end index is out of bounds.

```
self.slice(
  int,
  none int,
  count: int,
) → bytes
```

/2.3.1 **start** `int` *Required* *Positional*

The start index (inclusive).

/2.3.2 **end** `none` or `int` *Positional* Default: `none`

The end index (exclusive). If omitted, the whole slice until the end is extracted.

/2.3.3 **count** `int`

The number of items to extract. This is equivalent to passing `start + count` as the `end` position. Mutually exclusive with `end`.

3.5.7 Calculation

Module for calculations and processing of numeric values.

These definitions are part of the `calc` module and not imported by default. In addition to the functions listed below, the `calc` module also defines the constants `pi`, `tau`, `e`, and `inf`.

/1 Functions

/1.1 abs

Calculates the absolute value of a numeric value.

```
#calc.abs(-5) \
#calc.abs(5pt - 2cm) \
#calc.abs(2fr) \
#calc.abs(decimal("-342.440"))
```

```
5
51.69pt
2fr
342.440
```

`calc.abs(` `int` `float` `length` `angle` `ratio` `fraction` `decimal` `) → any`

/1.1.1 **value** `int` or `float` or `length` or `angle` or `ratio` or `fraction` or `decimal` *Required* *Positional*

The value whose absolute value to calculate.

/1.2 pow

Raises a value to some exponent.

```
#calc.pow(2, 3) \
#calc.pow(decimal("2.5"), 2)
```

```
8
6.25
```

`calc.pow(`
`int` `float` `decimal`,
`int` `float`,
`) → int float decimal`

/1.2.1 **base** `int` or `float` or `decimal` *Required* *Positional*

The base of the power.

If this is a `decimal`[3.5.10], the exponent can only be an integer[3.5.16].

/1.2.2 **exponent** `int` or `float` *Required* *Positional*

The exponent of the power.

/1.3 exp

Raises a value to some exponent of e.

```
#calc.exp(1)
```

```
2.7182818284590455
```

`calc.exp(` `int` `float` `) → float`

/1.3.1 **exponent** `int` or `float` *Required* *Positional*

The exponent of the power.

/1.4 sqrt

Calculates the square root of a number.

```
#calc.sqrt(16) \
#calc.sqrt(2.5)
```

4
1.5811388300841898

```
calc.sqrt( int float ) → float
```

/1.4.1 value `int` or `float` *Required* *Positional*

The number whose square root to calculate. Must be non-negative.

/1.5 root

Calculates the real n^{th} root of a number.

If the number is negative, then n must be odd.

```
#calc.root(16.0, 4) \
#calc.root(27.0, 3)
```

2
3

```
calc.root(
  float ,
  int ,
) → float
```

/1.5.1 radicand `float` *Required* *Positional*

The expression to take the root of.

/1.5.2 index `int` *Required* *Positional*

The value of n.

/1.6 sin

Calculates the sine of an angle.

When called with an integer or a float, they will be interpreted as radians.

```
#calc.sin(1.5) \
#calc.sin(90deg)
```

0.9974949866040544
1

```
calc.sin( int float angle ) → float
```

/1.6.1 angle `int` or `float` or `angle` *Required* *Positional*

The angle whose sine to calculate.

/1.7 cos

Calculates the cosine of an angle.

When called with an integer or a float, they will be interpreted as radians.

```
#calc.cos(1.5) \
#calc.cos(90deg)
```

0.0707372016677029
0.000000000000000006123233995736766

```
calc.cos( int  float  angle ) → float
```

/1.7.1 **angle** `int` or `float` or `angle` *Required* *Positional*

The angle whose cosine to calculate.

/1.8 **tan**

Calculates the tangent of an angle.

When called with an integer or a float, they will be interpreted as radians.

```
#calc.tan(1.5) \
#calc.tan(90deg)
```

14.101419947171719
16331239353195370

```
calc.tan( int  float  angle ) → float
```

/1.8.1 **angle** `int` or `float` or `angle` *Required* *Positional*

The angle whose tangent to calculate.

/1.9 **asin**

Calculates the arcsine of a number.

```
#calc.asin(0) \
#calc.asin(1)
```

0deg
90deg

```
calc.asin( int  float ) → angle
```

/1.9.1 **value** `int` or `float` *Required* *Positional*

The number whose arcsine to calculate. Must be between -1 and 1.

/1.10 **acos**

Calculates the arccosine of a number.

```
#calc.acos(0) \
#calc.acos(1)
```

90deg
0deg

```
calc.acos( int  float ) → angle
```

/1.10.1 **value** `int` or `float` *Required* *Positional*

The number whose arccosine to calculate. Must be between -1 and 1.

/1.11 **atan**

Calculates the arctangent of a number.

```
#calc.atan(0) \
#calc.atan(1)
```

```
0deg
45deg
```

```
calc.atan( int float ) → angle
```

/1.11.1 **value** `int` or `float` *Required* *Positional*

The number whose arctangent to calculate.

/1.12 **atan2**

Calculates the four-quadrant arctangent of a coordinate.

The four-quadrant arctangent of (x, y) is defined as the argument of the complex number $x + iy$.

Returns an `angle`[3.10.3] between `-180deg` and `180deg`.

Note that this function accepts (x, y), not (y, x).

```
#calc.atan2(1, 1) \
#calc.atan2(-2, -3)
```

```
45deg
-123.69deg
```

```
calc.atan2(
  int float,
  int float,
) → angle
```

/1.12.1 **x** `int` or `float` *Required* *Positional*

The x coordinate.

/1.12.2 **y** `int` or `float` *Required* *Positional*

The y coordinate.

/1.13 **sinh**

Calculates the hyperbolic sine of a hyperbolic angle.

The hyperbolic sine of x is defined as follows:

$$\frac{e^x - e^{-x}}{2}$$

```
#calc.sinh(0) \
#calc.sinh(1.5)
```

```
0
2.1292794550948173
```

```
calc.sinh( float ) → float
```

/1.13.1 **value** `float` *Required* *Positional*

The hyperbolic angle whose hyperbolic sine to calculate.

/1.14 **cosh**

Calculates the hyperbolic cosine of a hyperbolic angle.

The hyperbolic cosine of x is defined as follows:

$$\frac{e^x + e^{-x}}{2}$$

```
#calc.cosh(0) \
#calc.cosh(1.5)
```

```
1
2.352409615243247
```

```
calc.cosh( float ) → float
```

/1.14.1 **value** `float` *Required* *Positional*

The hyperbolic angle whose hyperbolic cosine to calculate.

/1.15 **tanh**

Calculates the hyperbolic tangent of a hyperbolic angle.

The hyperbolic tangent of x is defined as follows:

$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

```
#calc.tanh(0) \
#calc.tanh(1.5)
```

```
0
0.9051482536448664
```

```
calc.tanh( float ) → float
```

/1.15.1 **value** `float` *Required* *Positional*

The hyperbolic angle whose hyperbolic tangent to calculate.

/1.16 **asinh**

Calculates the inverse hyperbolic sine of a number.

The inverse hyperbolic sine of x is defined as follows:

$$\ln\left(x + \sqrt{x^2 + 1}\right)$$

```
#calc.asinh(0) \
#calc.asinh(1)
```

```
0
0.881373587019543
```

```
calc.asinh( float ) → float
```

/1.16.1 **value** `float` *Required* *Positional*

The number whose inverse hyperbolic sine to calculate.

/1.17 **acosh**

Calculates the inverse hyperbolic cosine of a number.

The inverse hyperbolic cosine of x is defined as follows:

$$\ln\left(x + \sqrt{x^2 - 1}\right)$$

```
#calc.acosh(1) \
#calc.acosh(2.5)
```

0
1.566799236972411

```
calc.acosh( float ) → float
```

/1.17.1 **value** `float` *Required* *Positional*

The number whose inverse hyperbolic cosine to calculate. Must be greater than or equal to 1.

/1.18 **atanh**

Calculates the inverse hyperbolic tangent of a number.

The inverse hyperbolic tangent of x is defined as follows:

$$\frac{1}{2}\ln\left(\frac{1+x}{1-x}\right)$$

```
#calc.atanh(0) \
#calc.atanh(0.5)
```

0
0.5493061443340548

```
calc.atanh( float ) → float
```

/1.18.1 **value** `float` *Required* *Positional*

The number whose inverse hyperbolic tangent to calculate. Must be between -1 and 1 (exclusive).

/1.19 **log**

Calculates the logarithm of a number.

If the base is not specified, the logarithm is calculated in base ten.

```
#calc.log(100)
```

2

```
calc.log(
  int float ,
  base: float ,
) → float
```

/1.19.1 **value** `int` or `float` *Required* *Positional*

The number whose logarithm to calculate. Must be strictly positive.

/1.19.2 **base** `float` Default: `10.0`

The base of the logarithm. May not be zero.

/1.20 **ln**

Calculates the natural logarithm of a number.

```
#calc.ln(calc.e)
```

1

```
calc.ln( int  float ) → float
```

/1.20.1 **value** `int` or `float` *Required* *Positional*

The number whose logarithm to calculate. Must be strictly positive.

/1.21 **erf**

Applies the error function to a number.

The value of the error function at x is defined as follows:

$$\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, \mathrm{d}t$$

```
#calc.erf(0.2)
```

0.22270258921047847

```
calc.erf( float ) → float
```

/1.21.1 **value** `float` *Required* *Positional*

The number at which to calculate the error function.

/1.22 **fact**

Calculates the factorial of a number.

```
#calc.fact(5)
```

120

```
calc.fact( int ) → int
```

/1.22.1 **number** `int` *Required* *Positional*

The number whose factorial to calculate. Must be non-negative.

/1.23 **perm**

Calculates a permutation.

Returns the k-permutation of n, or the number of ways to choose k items from a set of n with regard to order, defined as follows:

$$\begin{cases} 0 & \text{if } k > n \\ \frac{n!}{(n-k)!} & \text{if } k \leq n \end{cases}$$

```
#calc.perm(5, 3)
```

60

```
calc.perm(
  int ,
  int ,
) → int
```

/1.23.1 **base** `int` *Required* *Positional*

The value of n: The number of items to choose from. Must be non-negative.

/1.23.2 **numbers** `int` *Required* *Positional*

The value of k: The number of items to choose. Must be non-negative.

/1.24 **binom**

Calculates a binomial coefficient.

Returns the k-combination of n, or the number of ways to choose k items from a set of n without regard to order, defined as follows:

$$\begin{cases} \frac{n!}{k!(n-k)!} & \text{if } 0 \leq k \leq n \\ 0 & \text{otherwise} \end{cases}$$

```
#calc.binom(10, 5)
```

252

```
calc.binom(
  int,
  int,
) → int
```

/1.24.1 **n** `int` *Required* *Positional*

The value of n: The numbers of items to choose from. Must be non-negative.

/1.24.2 **k** `int` *Required* *Positional*

The value of k: The number of items to choose. Must be non-negative.

/1.25 **gcd**

Calculates the greatest common divisor of two integers.

This will error if the result of integer division would be larger than the maximum 64-bit signed integer.

```
#calc.gcd(7, 42)
```

7

```
calc.gcd(
  int,
  int,
) → int
```

/1.25.1 **a** `int` *Required* *Positional*

The first integer.

/1.25.2 **b** `int` *Required* *Positional*

The second integer.

/1.26 **lcm**

Calculates the least common multiple of two integers.

```
#calc.lcm(96, 13)
```

1248

```
calc.lcm(
  int,
  int,
) → int
```

/1.26.1 **a** `int` *Required* *Positional*

The first integer.

/1.26.2 **b** `int` *Required* *Positional*

The second integer.

/1.27 floor

Rounds a number down to the nearest integer.

If the number is already an integer, it is returned unchanged.

Note that this function will always return an integer[3.5.16], and will error if the resulting `float`[3.5.14] or `decimal`[3.5.10] is larger than the maximum 64-bit signed integer or smaller than the minimum for that type.

```
#calc.floor(500.1)
#assert(calc.floor(3) == 3)
#assert(calc.floor(3.14) == 3)
#assert(calc.floor(decimal("-3.14")) == -4)
```

500

```
calc.floor(int float decimal) → int
```

/1.27.1 **value** `int` or `float` or `decimal` *Required* *Positional*

The number to round down.

/1.28 ceil

Rounds a number up to the nearest integer.

If the number is already an integer, it is returned unchanged.

Note that this function will always return an integer[3.5.16], and will error if the resulting `float`[3.5.14] or `decimal`[3.5.10] is larger than the maximum 64-bit signed integer or smaller than the minimum for that type.

```
#calc.ceil(500.1)
#assert(calc.ceil(3) == 3)
#assert(calc.ceil(3.14) == 4)
#assert(calc.ceil(decimal("-3.14")) == -3)
```

501

```
calc.ceil(int float decimal) → int
```

/1.28.1 **value** `int` or `float` or `decimal` *Required* *Positional*

The number to round up.

/1.29 trunc

Returns the integer part of a number.

If the number is already an integer, it is returned unchanged.

Note that this function will always return an integer[3.5.16], and will error if the resulting `float`[3.5.14] or `decimal`[3.5.10] is larger than the maximum 64-bit signed integer or smaller than the minimum for that type.

```
#calc.trunc(15.9)
#assert(calc.trunc(3) == 3)
#assert(calc.trunc(-3.7) == -3)
#assert(calc.trunc(decimal("8493.12949582390")) == 8493)
```

15

```
calc.trunc( int  float  decimal ) → int
```

/1.29.1 **value** `int` or `float` or `decimal` *Required* *Positional*

The number to truncate.

/1.30 **fract**

Returns the fractional part of a number.

If the number is an integer, returns `0`.

```
#calc.fract(-3.1)
#assert(calc.fract(3) == 0)
#assert(calc.fract(decimal("234.23949211")) == decimal("0.23949211"))
```

−0.10000000000000009

```
calc.fract( int  float  decimal ) → int  float  decimal
```

/1.30.1 **value** `int` or `float` or `decimal` *Required* *Positional*

The number to truncate.

/1.31 **round**

Rounds a number to the nearest integer.

Half-integers are rounded away from zero.

Optionally, a number of decimal places can be specified. If negative, its absolute value will indicate the amount of significant integer digits to remove before the decimal point.

Note that this function will return the same type as the operand. That is, applying `round` to a `float`[3.5.14] will return a `float`, and to a `decimal`[3.5.10], another `decimal`. You may explicitly convert the output of this function to an integer with `int`[3.5.16], but note that such a conversion will error if the `float` or `decimal` is larger than the maximum 64-bit signed integer or smaller than the minimum integer.

In addition, this function can error if there is an attempt to round beyond the maximum or minimum integer or `decimal`. If the number is a `float`, such an attempt will cause `float.inf` or `-float.inf` to be returned for maximum and minimum respectively.

```
#calc.round(3.1415, digits: 2)
#assert(calc.round(3) == 3)
#assert(calc.round(3.14) == 3)
#assert(calc.round(3.5) == 4.0)
#assert(calc.round(3333.45, digits: -2) == 3300.0)
```

```
#assert(calc.round(-48953.45, digits: -3) == -49000.0)
#assert(calc.round(3333, digits: -2) == 3300)
#assert(calc.round(-48953, digits: -3) == -49000)
#assert(calc.round(decimal("-6.5")) == decimal("-7"))
#assert(calc.round(decimal("7.123456789"), digits: 6) == decimal("7.123457"))
#assert(calc.round(decimal("3333.45"), digits: -2) == decimal("3300"))
#assert(calc.round(decimal("-48953.45"), digits: -3) == decimal("-49000"))
```

3.14

```
calc.round(
  int float decimal,
  digits: int,
) → int float decimal
```

/1.31.1 **value** `int` or `float` or `decimal` *Required* *Positional*

The number to round.

/1.31.2 **digits** `int` Default: `0`

If positive, the number of decimal places.

If negative, the number of significant integer digits that should be removed before the decimal point.

/1.32 clamp

Clamps a number between a minimum and maximum value.

```
#calc.clamp(5, 0, 4)
#assert(calc.clamp(5, 0, 10) == 5)
#assert(calc.clamp(5, 6, 10) == 6)
#assert(calc.clamp(decimal("5.45"), 2, decimal("45.9")) == decimal("5.45"))
#assert(calc.clamp(decimal("5.45"), decimal("6.75"), 12) == decimal("6.75"))
```

4

```
calc.clamp(
  int float decimal,
  int float decimal,
  int float decimal,
) → int float decimal
```

/1.32.1 **value** `int` or `float` or `decimal` *Required* *Positional*

The number to clamp.

/1.32.2 **min** `int` or `float` or `decimal` *Required* *Positional*

The inclusive minimum value.

/1.32.3 **max** `int` or `float` or `decimal` *Required* *Positional*

The inclusive maximum value.

/1.33 min

Determines the minimum of a sequence of values.

```
#calc.min(1, -3, -5, 20, 3, 6) \
#calc.min("typst", "is", "cool")
```

```
-5
cool
```

```
calc.min(.. any ) → any
```

/1.33.1 **values** `any` *Required* *Positional* *Variadic*

The sequence of values from which to extract the minimum. Must not be empty.

/1.34 **max**

Determines the maximum of a sequence of values.

```
#calc.max(1, -3, -5, 20, 3, 6) \
#calc.max("typst", "is", "cool")
```

```
20
typst
```

```
calc.max(.. any ) → any
```

/1.34.1 **values** `any` *Required* *Positional* *Variadic*

The sequence of values from which to extract the maximum. Must not be empty.

/1.35 **even**

Determines whether an integer is even.

```
#calc.even(4) \
#calc.even(5) \
#range(10).filter(calc.even)
```

```
true
false
(0, 2, 4, 6, 8)
```

```
calc.even( int ) → bool
```

/1.35.1 **value** `int` *Required* *Positional*

The number to check for evenness.

/1.36 **odd**

Determines whether an integer is odd.

```
#calc.odd(4) \
#calc.odd(5) \
#range(10).filter(calc.odd)
```

```
false
true
(1, 3, 5, 7, 9)
```

```
calc.odd( int ) → bool
```

/1.36.1 **value** `int` *Required* *Positional*

The number to check for oddness.

/1.37 **rem**

Calculates the remainder of two numbers.

The value `calc.rem(x, y)` always has the same sign as `x`, and is smaller in magnitude than `y`.

This can error if given a `decimal`[3.5.10] input and the dividend is too small in magnitude compared to the divisor.

```
#calc.rem(7, 3) \
#calc.rem(7, -3) \
#calc.rem(-7, 3) \
#calc.rem(-7, -3) \
#calc.rem(1.75, 0.5)
```

1
1
−1
−1
0.25

```
calc.rem(
  int float decimal,
  int float decimal,
) → int float decimal
```

/1.37.1 **dividend** `int` or `float` or `decimal` *Required* *Positional*

The dividend of the remainder.

/1.37.2 **divisor** `int` or `float` or `decimal` *Required* *Positional*

The divisor of the remainder.

/1.38 div-euclid

Performs euclidean division of two numbers.

The result of this computation is that of a division rounded to the integer `n` such that the dividend is greater than or equal to `n` times the divisor.

This can error if the resulting number is larger than the maximum value or smaller than the minimum value for its type.

```
#calc.div-euclid(7, 3) \
#calc.div-euclid(7, -3) \
#calc.div-euclid(-7, 3) \
#calc.div-euclid(-7, -3) \
#calc.div-euclid(1.75, 0.5) \
#calc.div-euclid(decimal("1.75"), decimal("0.5"))
```

2
−2
−3
3
3
3

```
calc.div-euclid(
  int float decimal,
  int float decimal,
) → int float decimal
```

/1.38.1 **dividend** `int` or `float` or `decimal` *Required* *Positional*

The dividend of the division.

/1.38.2 **divisor** `int` or `float` or `decimal` *Required* *Positional*

The divisor of the division.

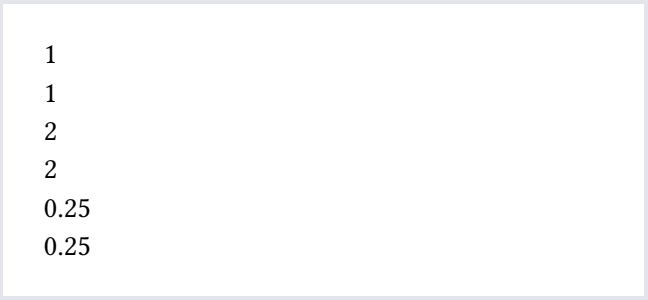
/1.39 **rem-euclid**

This calculates the least nonnegative remainder of a division.

Warning: Due to a floating point round-off error, the remainder may equal the absolute value of the divisor if the dividend is much smaller in magnitude than the divisor and the dividend is negative. This only applies for floating point inputs.

In addition, this can error if given a `decimal`[3.5.10] input and the dividend is too small in magnitude compared to the divisor.

```
#calc.rem-euclid(7, 3) \
#calc.rem-euclid(7, -3) \
#calc.rem-euclid(-7, 3) \
#calc.rem-euclid(-7, -3) \
#calc.rem-euclid(1.75, 0.5) \
#calc.rem-euclid(decimal("1.75"), decimal("0.5"))
```




```
calc.rem-euclid(
  int float decimal,
  int float decimal,
) → int float decimal
```

/1.39.1 **dividend** `int` or `float` or `decimal` *Required* *Positional*

The dividend of the remainder.

/1.39.2 **divisor** `int` or `float` or `decimal` *Required* *Positional*

The divisor of the remainder.

/1.40 **quo**

Calculates the quotient (floored division) of two numbers.

Note that this function will always return an integer[3.5.16], and will error if the resulting number is larger than the maximum 64-bit signed integer or smaller than the minimum for that type.

```
$ "quo"(a, b) &= floor(a/b) \
  "quo"(14, 5) &= #calc.quo(14, 5) \
  "quo"(3.46, 0.5) &= #calc.quo(3.46, 0.5) $
```

$$\operatorname{quo}(a, b) = \left\lfloor \frac{a}{b} \right\rfloor$$
$$\operatorname{quo}(14, 5) = 2$$
$$\operatorname{quo}(3.46, 0.5) = 6$$

```
calc.quo(
  int float decimal,
  int float decimal,
) → int
```

/1.40.1 **dividend** `int` or `float` or `decimal` *Required* *Positional*

The dividend of the quotient.

/1.40.2 **divisor** `int` or `float` or `decimal` *Required* *Positional*

The divisor of the quotient.

/1.41 norm

Calculates the p-norm of a sequence of values.

The p-norm of $x_1, ..., x_n$ is defined as follows:

$$\begin{cases} \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} & \text{if } 0 < p < +\infty \\ \max_{i=1}^{n} |x_i| & \text{if } p = +\infty \end{cases}$$

```
#calc.norm(1, 2, -3, 0.5) \
#calc.norm(p: 3, 1, 2)
```

3.774917217635375
2.080083823051904

```
calc.norm(
  p: float,
  ..float,
) → float
```

/1.41.1 **p** `float` Default: `2.0`

The value of p. Must be greater than zero.

The default value of `2.0` corresponds to the Euclidean norm:

$$\sqrt{\sum_{i=1}^{n} x_i^2}$$

/1.41.2 **values** `float` *Required* *Positional* *Variadic*

The sequence of values to calculate the p-norm of. Returns `0.0` if empty.

3.5.8 content

A piece of document content.

This type is at the heart of Typst. All markup you write and most functions[3.5.15] you call produce content values. You can create a content value by enclosing markup in square brackets. This is also how you pass content to functions.

/1 Example

```
Type of *Hello!* is
#type([*Hello!*])
```

Type of **Hello!** is content

Content can be added with the + operator, joined together[3.3.2] and multiplied with integers. Wherever content is expected, you can also pass a string[3.5.27] or none.

/2 Representation

Content consists of elements with fields. When constructing an element with its *element function,* you provide these fields as arguments and when you have a content value, you can access its fields with field access syntax[3.3.6].

Some fields are required: These must be provided when constructing an element and as a consequence, they are always available through field access on content of that type. Required fields are marked as such in the documentation.

Most fields are optional: Like required fields, they can be passed to the element function to configure them for a single element. However, these can also be configured with set rules[3.2.1] to apply them to all elements within a scope. Optional fields are only available with field access syntax when they were explicitly passed to the element function, not when they result from a set rule.

Each element has a default appearance. However, you can also completely customize its appearance with a show rule[3.2.2]. The show rule is passed the element. It can access the element's field and produce arbitrary content from it.

In the web app, you can hover over a content variable to see exactly which elements the content is composed of and what fields they have. Alternatively, you can inspect the output of the repr[3.5.24] function.

/3 Definitions

/3.1 func

The content's element function. This function can be used to create the element contained in this content. It can be used in set and show rules for the element. Can be compared with global functions to check whether you have a specific kind of element.

```
self.func() → function
```

/3.2 has

Whether the content has the specified field.

```
self.has(str) → bool
```

/3.2.1 **field** str *Required* *Positional*

The field to look for.

/3.3 at

Access the specified field on the content. Returns the default value if the field does not exist or fails with an error if no default value was specified.

```
self.at(
  str,
  default: any,
) → any
```

/3.3.1 **field** `str` *Required* *Positional*

The field to access.

/3.3.2 **default** `any`

A default value to return if the field does not exist.

/3.4 fields

Returns the fields of this content.

```
#rect(
  width: 10cm,
  height: 10cm,
).fields()
```

```
(width: 0% + 283.46pt, height: 0% +
283.46pt)
```

```
self.fields() → dictionary
```

/3.5 location

The location of the content. This is only available on content returned by query[3.12.6] or provided by a show rule[3.2.2], for other content it will be none. The resulting location can be used with counters[3.12.1], state[3.12.7] and queries[3.12.6].

```
self.location() → none location
```

3.5.9 `datetime`

Represents a date, a time, or a combination of both.

Can be created by either specifying a custom datetime using this type's constructor function or getting the current date with `datetime.today`[3.5.9/5.1].

/1 Example

```
#let date = datetime(
  year: 2020,
  month: 10,
  day: 4,
)

#date.display() \
#date.display(
  "y:[year repr:last_two]"
)

#let time = datetime(
  hour: 18,
  minute: 2,
  second: 23,
)

#time.display() \
#time.display(
  "h:[hour repr:12][period]"
)
```

2020-10-04
y:20

18:02:23
h:06PM

/2 Datetime and Duration

You can get a duration[3.5.12] by subtracting two datetime:

```
#let first-of-march = datetime(day: 1, month: 3, year: 2024)
#let first-of-jan = datetime(day: 1, month: 1, year: 2024)
#let distance = first-of-march - first-of-jan
#distance.hours()
```

1440

You can also add/subtract a datetime and a duration to retrieve a new, offset datetime:

```
#let date = datetime(day: 1, month: 3, year: 2024)
#let two-days = duration(days: 2)
#let two-days-earlier = date - two-days
#let two-days-later = date + two-days

#date.display() \
#two-days-earlier.display() \
#two-days-later.display()
```

```
2024-03-01
2024-02-28
2024-03-03
```

/3 ## Format

You can specify a customized formatting using the `display`[3.5.9/5.2] method. The format of a datetime is specified by providing *components* with a specified number of *modifiers*. A component represents a certain part of the datetime that you want to display, and with the help of modifiers you can define how you want to display that component. In order to display a component, you wrap the name of the component in square brackets (e.g. `[year]` will display the year). In order to add modifiers, you add a space after the component name followed by the name of the modifier, a colon and the value of the modifier (e.g. `[month repr:short]` will display the short representation of the month).

The possible combination of components and their respective modifiers is as follows:

- `year`: Displays the year of the datetime.
  - `base`: Can be either `calendar` or `iso_week`. Specifies whether the year is based on the Gregorian calendar or the ISO week number.
  - `padding`: Can be either `zero`, `space` or `none`. Specifies how the year is padded.
  - `repr` Can be either `full` in which case the full year is displayed or `last_two` in which case only the last two digits are displayed.
  - `sign`: Can be either `automatic` or `mandatory`. Specifies when the sign should be displayed.
- `month`: Displays the month of the datetime.
  - `padding`: Can be either `zero`, `space` or `none`. Specifies how the month is padded.
  - `repr`: Can be either `numerical`, `long` or `short`. Specifies if the month should be displayed as a number or a word. Unfortunately, when choosing the word representation, it can currently only display the English version. In the future, it is planned to support localization.
- `day`: Displays the day of the datetime.
  - `padding`: Can be either `zero`, `space` or `none`. Specifies how the day is padded.
- `week_number`: Displays the week number of the datetime.
  - `padding`: Can be either `zero`, `space` or `none`. Specifies how the week number is padded.
  - `repr`: Can be either `ISO`, `sunday` or `monday`. In the case of `ISO`, week numbers are between 1 and 53, while the other ones are between 0 and 53.
- `weekday`: Displays the weekday of the date.
  - `repr` Can be either `long`, `short`, `sunday` or `monday`. In the case of `long` and `short`, the corresponding English name will be displayed (same as for the month, other languages are currently not supported). In the case of `sunday` and `monday`, the numerical value will be displayed (assuming Sunday and Monday as the first day of the week, respectively).
  - `one_indexed`: Can be either `true` or `false`. Defines whether the numerical representation of the week starts with 0 or 1.
- `hour`: Displays the hour of the date.
  - `padding`: Can be either `zero`, `space` or `none`. Specifies how the hour is padded.
  - `repr`: Can be either `24` or `12`. Changes whether the hour is displayed in the 24-hour or 12-hour format.
- `period`: The AM/PM part of the hour
  - `case`: Can be `lower` to display it in lower case and `upper` to display it in upper case.
- `minute`: Displays the minute of the date.
  - `padding`: Can be either `zero`, `space` or `none`. Specifies how the minute is padded.
- `second`: Displays the second of the date.
  - `padding`: Can be either `zero`, `space` or `none`. Specifies how the second is padded.

See here[A] for more details on the supported syntax.

Keep in mind that not always all components can be used. For example, if you create a new datetime with `datetime(year: 2023, month: 10, day: 13)`, it will be stored as a plain date internally, meaning that

[A] https://time-rs.github.io/book/api/format-description.html#components

you cannot use components such as `hour` or `minute`, which would only work on datetimes that have a specified time.

/4 Constructor

Creates a new datetime.

You can specify the datetime[3.5.9] using a year, month, day, hour, minute, and second.

Note: Depending on which components of the datetime you specify, Typst will store it in one of the following three ways:
– If you specify year, month and day, Typst will store just a date.
– If you specify hour, minute and second, Typst will store just a time.
– If you specify all of year, month, day, hour, minute and second, Typst will store a full datetime.

Depending on how it is stored, the `display`[3.5.9/5.2] method will choose a different formatting by default.

```
#datetime(
  year: 2012,
  month: 8,
  day: 3,
).display()
```

2012-08-03

```
datetime(
  year: int,
  month: int,
  day: int,
  hour: int,
  minute: int,
  second: int,
) → datetime
```

/4.1 year int

The year of the datetime.

/4.2 month int

The month of the datetime.

/4.3 day int

The day of the datetime.

/4.4 hour int

The hour of the datetime.

/4.5 minute int

The minute of the datetime.

/4.6 second int

The second of the datetime.

/5 Definitions

/5.1 today

Returns the current date.

In the CLI, this can be overridden with the `--creation-timestamp` argument or by setting the `SOURCE_DATE_EPOCH`[A] environment variable. In both cases, the value should be given as a UNIX timestamp.

[A] https://reproducible-builds.org/specs/source-date-epoch/

```
Today's date is
#datetime.today().display().
```

Today's date is 1970-01-01.

```
datetime.today(offset: auto int duration) → datetime
```

/5.1.1 **offset** auto or int or duration Default: auto

An offset to apply to the current UTC date. If set to auto, the offset will be the local offset.

When an integer offset is given, it will be treated as a duration in hours.

/5.2 display

Displays the datetime in a specified format.

Depending on whether you have defined just a date, a time or both, the default format will be different. If you specified a date, it will be `[year]-[month]-[day]`. If you specified a time, it will be `[hour]:[minute]:[second]`. In the case of a datetime, it will be `[year]-[month]-[day] [hour]:[minute]:[second]`.

See the format syntax[3.5.9/3] for more information.

```
self.display(auto str) → str
```

/5.2.1 **pattern** auto or str *Positional* Default: auto

The format used to display the datetime.

/5.3 year

The year if it was specified, or none for times without a date.

```
self.year() → none int
```

/5.4 month

The month if it was specified, or none for times without a date.

```
self.month() → none int
```

/5.5 weekday

The weekday (counting Monday as 1) or none for times without a date.

```
self.weekday() → none int
```

/5.6 day

The day if it was specified, or none for times without a date.

```
self.day() → none int
```

/5.7 hour

The hour if it was specified, or none for dates without a time.

```
self.hour() → none int
```

/5.8 **minute**

The minute if it was specified, or none for dates without a time.

```
self.minute() → none int
```

/5.9 **second**

The second if it was specified, or none for dates without a time.

```
self.second() → none int
```

/5.10 **ordinal**

The ordinal (day of the year), or none for times without a date.

```
self.ordinal() → none int
```

3.5.10 `decimal`

A fixed-point decimal number type.

This type should be used for precise arithmetic operations on numbers represented in base 10. A typical use case is representing currency.

/1 Example

```
Decimal: #(decimal("0.1") + decimal("0.2")) \
Float: #(0.1 + 0.2)
```

Decimal: 0.3
Float: 0.30000000000000004

/2 Construction and casts

To create a decimal number, use the `decimal(string)` constructor, such as in `decimal("3.141592653")` *(note the double quotes)*. This constructor preserves all given fractional digits, provided they are representable as per the limits specified below (otherwise, an error is raised).

You can also convert any integer[3.5.16] to a decimal with the `decimal(int)` constructor, e.g. `decimal(59)`. However, note that constructing a decimal from a floating-point number[3.5.14], while supported, **is an imprecise conversion and therefore discouraged.** A warning will be raised if Typst detects that there was an accidental `float` to `decimal` cast through its constructor, e.g. if writing `decimal(3.14)` (note the lack of double quotes, indicating this is an accidental `float` cast and therefore imprecise). It is recommended to use strings for constant decimal values instead (e.g. `decimal("3.14")`).

The precision of a `float` to `decimal` cast can be slightly improved by rounding the result to 15 digits with `calc.round`[3.5.7/1.31], but there are still no precision guarantees for that kind of conversion.

/3 Operations

Basic arithmetic operations are supported on two decimals and on pairs of decimals and integers.

Built-in operations between `float` and `decimal` are not supported in order to guard against accidental loss of precision. They will raise an error instead.

Certain `calc` functions, such as trigonometric functions and power between two real numbers, are also only supported for `float` (although raising `decimal` to integer exponents is supported). You can opt into potentially imprecise operations with the `float(decimal)` constructor, which casts the `decimal` number into a `float`, allowing for operations without precision guarantees.

/4 Displaying decimals

To display a decimal, simply insert the value into the document. To only display a certain number of digits, round[3.5.7/1.31] the decimal first. Localized formatting of decimals and other numbers is not yet supported, but planned for the future.

You can convert decimals to strings using the `str`[3.5.27] constructor. This way, you can post-process the displayed representation, e.g. to replace the period with a comma (as a stand-in for proper built-in localization to languages that use the comma).

/5 Precision and limits

A `decimal` number has a limit of 28 to 29 significant base-10 digits. This includes the sum of digits before and after the decimal point. As such, numbers with more fractional digits have a smaller range. The maximum and minimum `decimal` numbers have a value of `79228162514264337593543950335` and `-79228162514264337593543950335` respectively. In contrast with `float`[3.5.14], this type does not support infinity or NaN, so overflowing or underflowing operations will raise an error.

Typical operations between `decimal` numbers, such as addition, multiplication, and power[3.5.7/1.2] to an integer, will be highly precise due to their fixed-point representation. Note, however, that multiplication and division may not preserve all digits in some edge cases: while they are considered precise, digits past the limits specified above are rounded off and lost, so some loss of precision beyond the maximum representable digits is possible. Note that this behavior can be observed not only when dividing, but also when multiplying by numbers between 0 and 1, as both operations can push a number's fractional digits beyond the limits described above, leading to rounding. When those two operations do not surpass the digit limits, they are fully precise.

/6 Constructor

Converts a value to a `decimal`.

It is recommended to use a string to construct the decimal number, or an integer[3.5.16] (if desired). The string must contain a number in the format `"3.14159"` (or `"-3.141519"` for negative numbers). The fractional digits are fully preserved; if that's not possible due to the limit of significant digits (around 28 to 29) having been reached, an error is raised as the given decimal number wouldn't be representable.

While this constructor can be used with floating-point numbers[3.5.14] to cast them to `decimal`, doing so is **discouraged** as **this cast is inherently imprecise.** It is easy to accidentally perform this cast by writing `decimal(1.234)` (note the lack of double quotes), which is why Typst will emit a warning in that case. Please write `decimal("1.234")` instead for that particular case (initialization of a constant decimal). Also note that floats that are NaN or infinite cannot be cast to decimals and will raise an error.

```
#decimal("1.222222222222222")
```

1.222222222222222

```
decimal( bool  int  float  decimal  str ) → decimal
```

/6.1 **value** `bool` or `int` or `float` or `decimal` or `str` *Required* *Positional*

The value that should be converted to a decimal.

3.5.11 dictionary

A map from string keys to values.

You can construct a dictionary by enclosing comma-separated `key: value` pairs in parentheses. The values do not have to be of the same type. Since empty parentheses already yield an empty array, you have to use the special `(:)` syntax to create an empty dictionary.

A dictionary is conceptually similar to an array[3.5.2], but it is indexed by strings instead of integers. You can access and create dictionary entries with the `.at()` method. If you know the key statically, you can alternatively use field access notation[3.3.6] (`.key`) to access the value. To check whether a key is present in the dictionary, use the `in` keyword.

You can iterate over the pairs in a dictionary using a for loop[3.3.5]. This will iterate in the order the pairs were inserted / declared initially.

Dictionaries can be added with the `+` operator and joined together[3.3.2]. They can also be spread[3.5.1/2] into a function call or another dictionary[A] with the `..spread` operator. In each case, if a key appears multiple times, the last value will override the others.

/1 Example

```
#let dict = (
  name: "Typst",
  born: 2019,
)

#dict.name \
#(dict.launch = 20)
#dict.len() \
#dict.keys() \
#dict.values() \
#dict.at("born") \
#dict.insert("city", "Berlin")
#("name" in dict)
```

Typst
3
("name", "born", "launch")
("Typst", 2019, 20)
2019
true

/2 Constructor

Converts a value into a dictionary.

Note that this function is only intended for conversion of a dictionary-like value to a dictionary, not for creation of a dictionary from individual pairs. Use the dictionary syntax `(key: value)` instead. Also see `array.to-dict`[3.5.2/3.32] for converting arrays to dictionaries.

```
#dictionary(sys).at("version")
```

0.15.1

```
dictionary(module) → dictionary
```

/2.1 value `module` *Required* *Positional*

The value that should be converted to a dictionary.

/3 Definitions

/3.1 len

The number of pairs in the dictionary.

[A] When spreading into a dictionary, if all items between the parentheses are spread, you have to begin the container with `(:`, as in `(: ..dict, ..other_dict)`. Otherwise the container is inferred to be an array and an error is raised.

```
self.len() → int
```

/3.2 at

Returns the value associated with the specified key in the dictionary.

May be used on the left-hand side of an assignment if the key is already present in the dictionary. Returns the default value if the key is not part of the dictionary or fails with an error if no default value was specified.

Values may also be accessed with field syntax (e.g. `(key: 42).key`) if no default is needed.

```
self.at(
  str,
  default: any,
) → any
```

/3.2.1 key `str` *Required* *Positional*

The key at which to retrieve the item.

/3.2.2 default `any`

A default value to return if the key is not part of the dictionary.

/3.3 insert

Inserts a new pair into the dictionary. If the dictionary already contains this key, the value is updated.

To insert multiple pairs at once, you can alternatively add another dictionary with the += operator.

```
self.insert(
  str,
  any,
) → none
```

/3.3.1 key `str` *Required* *Positional*

The key of the pair that should be inserted.

/3.3.2 value `any` *Required* *Positional*

The value of the pair that should be inserted.

/3.4 remove

Removes a pair from the dictionary by key and return the value.

```
self.remove(
  str,
  default: any,
) → any
```

/3.4.1 key `str` *Required* *Positional*

The key of the pair to remove.

/3.4.2 default `any`

A default value to return if the key does not exist.

/3.5 keys

Returns the keys of the dictionary as an array in insertion order.

```
self.keys() → array
```

/3.6 values

Returns the values of the dictionary as an array in insertion order.

```
self.values() → array
```

/3.7 pairs

Returns the keys and values of the dictionary as an array of pairs. Each pair is represented as an array of length two.

```
self.pairs() → array
```

/3.8 filter

Produces a new dictionary with only the pairs from the original one for which the given function returns `true`.

Example · Basic usage

```
#{
  (a: 0, b: 1, c: 2)
    .filter(v => v > 0)
}
```

```
(b: 1, c: 2)
```

Example · Filtering based on the key instead of the value

```
#{
  (a: 0, b: 1, c: 2)
    .pairs()
    .filter(((k, v)) => k != "a")
    .to-dict()
}
```

```
(b: 1, c: 2)
```

```
self.filter(function) → dictionary
```

/3.8.1 test `function` *Required* *Positional*

The function to apply to each value. Must return a boolean.

/3.9 map

Produces a new dictionary where the keys are the same, but the values are transformed with the given function.

```
#(a: 0, b: 1, c: 2).map(v => v + 1)
```

```
(a: 1, b: 2, c: 3)
```

```
self.map(function) → dictionary
```

/3.9.1 mapper `function` *Required* *Positional*

The function to apply to each value.

3.5.12 duration

Represents a positive or negative span of time.

/1 Constructor

Creates a new duration.

You can specify the duration[3.5.12] using weeks, days, hours, minutes and seconds. You can also get a duration by subtracting two datetimes[3.5.9].

```
#duration(
  days: 3,
  hours: 12,
).hours()
```

```
84
```

```
duration(
  seconds: int,
  minutes: int,
  hours: int,
  days: int,
  weeks: int,
) → duration
```

/1.1 seconds int

Default: 0

The number of seconds.

/1.2 minutes int

Default: 0

The number of minutes.

/1.3 hours int

Default: 0

The number of hours.

/1.4 days int

Default: 0

The number of days.

/1.5 weeks int

Default: 0

The number of weeks.

/2 Definitions

/2.1 seconds

The duration expressed in seconds.

This function returns the total duration represented in seconds as a floating-point number, rather than the seconds component of the duration.

```
self.seconds() → float
```

/2.2 minutes

The duration expressed in minutes.

This function returns the total duration represented in minutes as a floating-point number, rather than the minutes component of the duration.

```
self.minutes() → float
```

/2.3 hours

The duration expressed in hours.

This function returns the total duration represented in hours as a floating-point number, rather than the hours component of the duration.

```
self.hours() → float
```

/2.4 days

The duration expressed in days.

This function returns the total duration represented in days as a floating-point number, rather than the days component of the duration.

```
self.days() → float
```

/2.5 weeks

The duration expressed in weeks.

This function returns the total duration represented in weeks as a floating-point number, rather than the weeks component of the duration.

```
self.weeks() → float
```

3.5.13 eval

Evaluates a string as Typst code.

This function should only be used as a last resort.

/1 Example

```
#eval("1 + 1") \
#eval("(1, 2, 3, 4)").len() \
#eval("*Markup!*", mode: "markup") \
```

2
4
Markup!

/2 Parameters

```
eval(
  str,
  mode: str,
  scope: dictionary,
) → any
```

/2.1 source `str` *Required* *Positional*

A string of Typst code to evaluate.

/2.2 mode `str` Default: `"code"`

The syntactical mode[3.1.1] in which the string is parsed.

```
#eval("= Heading", mode: "markup")
#eval("1_2^3", mode: "math")
```

Heading
1_2^3

Variant	Details
`"markup"`	Evaluate as markup, as in a Typst file.
`"math"`	Evaluate as math, as in an equation.
`"code"`	Evaluate as code, as after a hash.

/2.3 scope `dictionary` Default: `(:)`

A scope of definitions that are made available.

```
#eval("x + 1", scope: (x: 2)) \
#eval(
  "abc/xyz",
  mode: "math",
  scope: (
    abc: $a + b + c$,
    xyz: $x + y + z$,
  ),
)
```

3
$\frac{a+b+c}{x+y+z}$

3.5.14 `float`

A floating-point number.

A limited-precision representation of a real number. Typst uses 64 bits to store floats. Wherever a float is expected, you can also pass an integer[3.5.16].

You can convert a value to a float with this type's constructor.

NaN and positive infinity are available as `float.nan` and `float.inf` respectively.

/1 Example

```
#3.14 \
#1e4 \
#(10 / 4)
```

3.14
10000
2.5

/2 Constructor

Converts a value to a float.

- Booleans are converted to `0.0` or `1.0`.
- Integers are converted to the closest 64-bit float. For integers with absolute value less than `calc.pow(2, 53)`, this conversion is exact.
- Ratios are divided by 100%.
- Strings are parsed in base 10 to the closest 64-bit float. Exponential notation is supported.

```
#float(false) \
#float(true) \
#float(4) \
#float(40%) \
#float("2.7") \
#float("1e5")
```

0
1
4
0.4
2.7
100000

`float(` `bool` `int` `float` `ratio` `decimal` `str` `) → float`

/2.1 **`value`** `bool` or `int` or `float` or `ratio` or `decimal` or `str` *Required* *Positional*

The value that should be converted to a float.

/3 Definitions

/3.1 `is-nan`

Checks if a float is not a number.

In IEEE 754, more than one bit pattern represents a NaN. This function returns `true` if the float is any of those bit patterns.

```
#float.is-nan(0) \
#float.is-nan(1) \
#float.is-nan(float.nan)
```

false
false
true

`self.is-nan() → bool`

/3.2 is-infinite

Checks if a float is infinite.

Floats can represent positive infinity and negative infinity. This function returns `true` if the float is an infinity.

```
#float.is-infinite(0) \
#float.is-infinite(1) \
#float.is-infinite(float.inf)
```

```
false
false
true
```

```
self.is-infinite() → bool
```

/3.3 signum

Calculates the sign of a floating point number.

- If the number is positive (including `+0.0`), returns `1.0`.
- If the number is negative (including `-0.0`), returns `-1.0`.
- If the number is NaN, returns `float.nan`.

```
#(5.0).signum() \
#(-5.0).signum() \
#(0.0).signum() \
#float.nan.signum()
```

```
1
−1
1
NaN
```

```
self.signum() → float
```

/3.4 from-bytes

Interprets bytes as a float.

```
#float.from-bytes(bytes((0, 0, 0, 0, 0, 0, 240, 63))) \
#float.from-bytes(bytes((63, 240, 0, 0, 0, 0, 0, 0)), endian: "big")
```

```
1
1
```

```
float.from-bytes(
  bytes,
  endian: str,
) → float
```

/3.4.1 bytes bytes *Required* *Positional*

The bytes that should be converted to a float.

Must have a length of either 4 or 8. The bytes are then interpreted in IEEE 754[A]'s binary32 (single-precision) or binary64 (double-precision) format depending on the length of the bytes.

[A] https://en.wikipedia.org/wiki/IEEE_754

/3.4.2 **endian** `str` Default: `"little"`

The endianness of the conversion.

Variant	Details
`"big"`	Big-endian byte order: The highest-value byte is at the beginning of the bytes.
`"little"`	Little-endian byte order: The lowest-value byte is at the beginning of the bytes.

/3.5 to-bytes

Converts a float to bytes.

```
#array(1.0.to-bytes(endian: "big")) \
#array(1.0.to-bytes())
```

```
(63, 240, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0, 240, 63)
```

```
self.to-bytes(
  endian: str,
  size: int,
) → bytes
```

/3.5.1 **endian** `str` Default: `"little"`

The endianness of the conversion.

Variant	Details
`"big"`	Big-endian byte order: The highest-value byte is at the beginning of the bytes.
`"little"`	Little-endian byte order: The lowest-value byte is at the beginning of the bytes.

/3.5.2 **size** `int` Default: 8

The size of the resulting bytes.

This must be either 4 or 8. The call will return the representation of this float in either IEEE 754[A]'s binary32 (single-precision) or binary64 (double-precision) format depending on the provided size.

[A] https://en.wikipedia.org/wiki/IEEE_754

3.5.15 `function`

A mapping from argument values to a return value.

You can call a function by writing a comma-separated list of function *arguments* enclosed in parentheses directly after the function name. Additionally, you can pass any number of trailing content block arguments to a function *after* the normal argument list. If the normal argument list would become empty, it can be omitted. Typst supports positional and named arguments. The former are identified by position and type, while the latter are written as `name: value`.

Within math mode, function calls have special behaviour. See the math documentation[3.8] for more details.

/1 Example

```
// Call a function.
#list([A], [B])

// Named arguments and trailing
// content blocks.
#enum(start: 2)[A][B]

// Version without parentheses.
#list[A][B]
```

- A
- B

2. A
3. B

- A
- B

Functions are a fundamental building block of Typst. Typst provides functions for a variety of typesetting tasks. Moreover, the markup you write is backed by functions and all styling happens through functions. This reference lists all available functions and how you can use them. Please also refer to the documentation about set[3.2.1] and show[3.2.2] rules to learn about additional ways you can work with functions in Typst.

/2 Element functions

Some functions are associated with *elements* like headings[3.6.10] or tables[3.6.20]. When called, these create an element of their respective kind. In contrast to normal functions, they can further be used in set rules[3.2.1], show rules[3.2.2], and selectors[3.5.25].

/3 Function scopes

Functions can hold related definitions in their own scope, similar to a module[3.3.8]. Examples of this are `assert.eq`[3.5.3/3.1] or `list.item`[3.6.3/4.1]. However, this feature is currently only available for built-in functions.

/4 Defining functions

You can define your own function with a let binding[3.3.3] that has a parameter list after the binding's name. The parameter list can contain mandatory positional parameters, named parameters with default values and argument sinks[3.5.1].

The right-hand side of a function binding is the function body, which can be a block or any other expression. It defines the function's return value and can depend on the parameters. If the function body is a code block[3.3.2], the return value is the result of joining the values of each expression in the block.

Within a function body, the `return` keyword can be used to exit early and optionally specify a return value. If no explicit return value is given, the body evaluates to the result of joining all expressions preceding the `return`.

Functions that don't return any meaningful value return `none`[3.5.19] instead. The return type of such functions is not explicitly specified in the documentation. (An example of this is `array.push`[3.5.2/3.5]).

```
#let alert(body, fill: red) = {
  set text(white)
  set align(center)
  rect(
    fill: fill,
    inset: 8pt,
    radius: 4pt,
    [*Warning:\ #body*],
  )
}

#alert[
  Danger is imminent!
]

#alert(fill: blue)[
  KEEP OFF TRACKS
]
```



/5 Importing functions

Functions can be imported from one file (`module`[3.3.8]) into another using `import`. For example, assume that we have defined the `alert` function from the previous example in a file called `foo.typ`. We can import it into another file by writing `import "foo.typ": alert`.

/6 Unnamed functions

You can also create an unnamed function without creating a binding by specifying a parameter list followed by => and the function body. If your function has just one parameter, the parentheses around the parameter list are optional. Unnamed functions are mainly useful for show rules, but also as arguments to other functions, like `array.position`[3.5.2/3.12].

```
#show "once?": it => [#it #it]
once?
```

once? once?

/7 Note on function purity

In Typst, all functions are *pure*. This means that for the same arguments, they always return the same result. They cannot "remember" things to produce another value when they are called a second time.

The only exception are built-in methods like `array.push(value)`[3.5.2/3.5]. These can modify the values they are called on.

/8 Definitions

/8.1 with

Returns a new function that has the given arguments pre-applied.

```
self.with(.. any) → function
```

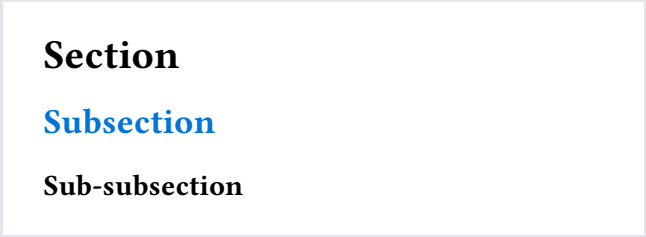
/8.1.1 arguments `any` *Required* *Positional* *Variadic*

The arguments to apply to the function.

/8.2 where

Returns a selector that filters for elements belonging to this function whose fields have the values of the given arguments.

```
#show heading.where(level: 2): set text(blue)
= Section
== Subsection
=== Sub-subsection
```




```
self.where(.. any ) → selector
```

/8.2.1 **fields** `any` *Required* *Positional* *Variadic*

The fields to filter for.

3.5.16 `int`

An integer: a positive whole number, a negative whole number, or zero.

Typst stores signed integers with the two's complement[A] representation in 64 bits. This allows storing numbers up to $2^{63} - 1$ or `9223372036854775807`, and down to -2^{63} or `-9223372036854775808`. These values are accessible as `int.max` and `int.min`.

Integers can also be specified as hexadecimal, octal, or binary by starting with the prefixes: `0x`, `0o`, or `0b`.

You can convert a value to an integer with this type's constructor.

/1 Example

```
#(1 + 2) \
#(2 - 5) \
#(3 + 4 < 8)

#0xff \
#0o10 \
#0b1001

#(int(3.8) + int("26"))

/ Max: #int.max
/ Min: #int.min
```

3
−3
true

255
8
9

29

Max 9223372036854775807
Min −9223372036854775808

/2 Constructor

Converts a value to an integer. Raises an error if there is an attempt to parse an invalid string or produce an integer that doesn't fit into a 64-bit signed integer.

- Booleans are converted to `0` or `1`.
- Floats and decimals are rounded to the next 64-bit integer towards zero.
- Strings are parsed in base 10 by default.

```
#int(false) \
#int(true) \
#int(2.7) \
#int(decimal("3.8")) \
#(int("27") + int("4")) \
#int("beef", base: 16)
```

0
1
2
3
31
48879

```
int(
  bool int float decimal str,
  base: int,
) → int
```

/2.1 **value** `bool` or `int` or `float` or `decimal` or `str` *Required* *Positional*

The value that should be converted to an integer.

/2.2 **base** `int` Default: `10`

The base (radix) for parsing strings, between 2 and 36.

[A] https://en.wikipedia.org/wiki/Two%27s_complement

/3 Definitions

/3.1 signum

Calculates the sign of an integer.

- If the number is positive, returns 1.
- If the number is negative, returns -1.
- If the number is zero, returns 0.

```
#(5).signum() \
#(-5).signum() \
#(0).signum()
```

```
1
-1
0
```

```
self.signum() → int
```

/3.2 bit-not

Calculates the bitwise NOT of an integer.

For the purposes of this function, the operand is treated as a signed integer of 64 bits.

```
#4.bit-not() \
#(-1).bit-not()
```

```
-5
0
```

```
self.bit-not() → int
```

/3.3 bit-and

Calculates the bitwise AND between two integers.

For the purposes of this function, the operands are treated as signed integers of 64 bits.

```
#128.bit-and(192)
```

```
128
```

```
self.bit-and(int) → int
```

/3.3.1 rhs int *Required* *Positional*

The right-hand operand of the bitwise AND.

/3.4 bit-or

Calculates the bitwise OR between two integers.

For the purposes of this function, the operands are treated as signed integers of 64 bits.

```
#64.bit-or(32)
```

```
96
```

```
self.bit-or(int) → int
```

/3.4.1 **rhs** `int` *Required* *Positional*

The right-hand operand of the bitwise OR.

/3.5 bit-xor

Calculates the bitwise XOR between two integers.

For the purposes of this function, the operands are treated as signed integers of 64 bits.

```
#64.bit-xor(96)
```

```
32
```

```
self.bit-xor( int ) → int
```

/3.5.1 **rhs** `int` *Required* *Positional*

The right-hand operand of the bitwise XOR.

/3.6 bit-lshift

Shifts the operand's bits to the left by the specified amount.

For the purposes of this function, the operand is treated as a signed integer of 64 bits. An error will occur if the result is too large to fit in a 64-bit integer.

```
#33.bit-lshift(2) \
#(-1).bit-lshift(3)
```

```
132
-8
```

```
self.bit-lshift( int ) → int
```

/3.6.1 **shift** `int` *Required* *Positional*

The amount of bits to shift. Must not be negative.

/3.7 bit-rshift

Shifts the operand's bits to the right by the specified amount. Performs an arithmetic shift by default (extends the sign bit to the left, such that negative numbers stay negative), but that can be changed by the `logical` parameter.

For the purposes of this function, the operand is treated as a signed integer of 64 bits.

```
#64.bit-rshift(2) \
#(-8).bit-rshift(2) \
#(-8).bit-rshift(2, logical: true)
```

```
16
-2
4611686018427387902
```

```
self.bit-rshift(
  int ,
  logical: bool ,
) → int
```

/3.7.1 **shift** `int` *Required* *Positional*

The amount of bits to shift. Must not be negative.

Shifts larger than 63 are allowed and will cause the return value to saturate. For non-negative numbers, the return value saturates at `0`, while, for negative numbers, it saturates at `-1` if `logical`

is set to `false`, or `0` if it is `true`. This behavior is consistent with just applying this operation multiple times. Therefore, the shift will always succeed.

/3.7.2 **logical** `bool` Default: `false`

Toggles whether a logical (unsigned) right shift should be performed instead of arithmetic right shift. If this is `true`, negative operands will not preserve their sign bit, and bits which appear to the left after the shift will be `0`. This parameter has no effect on non-negative operands.

/3.8 from-bytes

Converts bytes to an integer.

```
#int.from-bytes(bytes((0, 0, 0, 0, 0, 0, 0, 1))) \
#int.from-bytes(bytes((1, 0, 0, 0, 0, 0, 0, 0)), endian: "big")
```

```
72057594037927936
72057594037927936
```

```
int.from-bytes(
  bytes,
  endian: str,
  signed: bool,
) → int
```

/3.8.1 **bytes** `bytes` *Required* *Positional*

The bytes that should be converted to an integer.

Must be of length at most 8 so that the result fits into a 64-bit signed integer.

/3.8.2 **endian** `str` Default: `"little"`

The endianness of the conversion.

Variant	Details
`"big"`	Big-endian byte order: The highest-value byte is at the beginning of the bytes.
`"little"`	Little-endian byte order: The lowest-value byte is at the beginning of the bytes.

/3.8.3 **signed** `bool` Default: `true`

Whether the bytes should be treated as a signed integer. If this is `true` and the most significant bit is set, the resulting number will negative.

/3.9 to-bytes

Converts an integer to bytes.

```
#array(10000.to-bytes(endian: "big")) \
#array(10000.to-bytes(size: 4))
```

```
(0, 0, 0, 0, 0, 0, 39, 16)
(16, 39, 0, 0)
```

```
self.to-bytes(
  endian: str,
  size: int,
) → bytes
```

3.5.16 `int`

/3.9.1 **`endian`** `str` Default: `"little"`

The endianness of the conversion.

Variant	Details
`"big"`	Big-endian byte order: The highest-value byte is at the beginning of the bytes.
`"little"`	Little-endian byte order: The lowest-value byte is at the beginning of the bytes.

/3.9.2 **`size`** `int` Default: 8

The size in bytes of the resulting bytes (must be at least zero). If the integer is too large to fit in the specified size, the conversion will truncate the remaining bytes based on the endianness. To keep the same resulting value, if the endianness is big-endian, the truncation will happen at the rightmost bytes. Otherwise, if the endianness is little-endian, the truncation will happen at the leftmost bytes.

Be aware that if the integer is negative and the size is not enough to make the number fit, when passing the resulting bytes to `int.from-bytes`, the resulting number might be positive, as the most significant bit might not be set to 1.

3.5.17 label

A label for an element.

Inserting a label into content attaches it to the closest preceding element that is not a space. The preceding element must be in the same scope as the label, which means that `Hello #[<label>]`, for instance, wouldn't work.

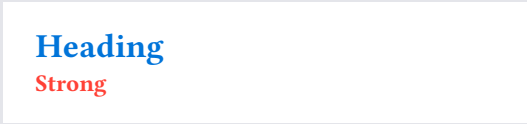
A labelled element can be referenced[3.6.18], queried[3.12.6] for, and styled[3.2] through its label.

Once constructed, you can get the name of a label using `str`[3.5.27/3].

/1 Example

```
#show <a>: set text(blue)
#show label("b"): set text(red)

= Heading <a>
*Strong* #label("b")
```



/2 Syntax

This function also has dedicated syntax: You can create a label by enclosing its name in angle brackets. This works both in markup and code. A label's name can contain letters, numbers, `_`, `-`, `:`, and `.`. A label cannot be empty.

Note that there is a syntactical difference when using the dedicated syntax for this function. In the code below, the `<a>` terminates the heading and thus attaches to the heading itself, whereas the `#label("b")` is part of the heading and thus attaches to the heading's text.

```
// Equivalent to `#heading[Introduction] <a>`.
= Introduction <a>

// Equivalent to `#heading[Conclusion #label("b")]`.
= Conclusion #label("b")
```

Currently, labels can only be attached to elements in markup mode, not in code mode. This might change in the future.

/3 Constructor

Creates a label from a string.

```
label( str ) → label
```

/3.1 name `str` *Required* *Positional*

The name of the label.

Unlike the dedicated syntax[3.5.17/2], this constructor accepts any non-empty string, including names with special characters.

3.5.18 `module`

A collection of variables and functions that are commonly related to a single theme.

A module can
– be built-in
– stem from a file import[3.3.8]
– stem from a package import[3.3.9] (and thus indirectly its entrypoint file)
– result from a call to the plugin[3.5.22] function

You can access definitions from the module using field access notation[3.3.6] and interact with it using the import and include syntaxes[3.3.8].

```
#import "utils.typ"
#utils.add(2, 5)

#import utils: sub
#sub(1, 4)
```

```
7
−3
```

You can check whether a definition is present in a module using the `in` operator, with a string on the left-hand side. This can be useful to conditionally access[3.5.26/2] definitions in a module.

```
#("table" in std) \
#("nope" in std)
```

```
true
false
```

Alternatively, it is possible to convert a module to a dictionary, and therefore access its contents dynamically, using the dictionary constructor[3.5.11/2].

3.5.19 none

A value that indicates the absence of any other value.

The none type has exactly one value: `none`.

When inserted into the document, it is not visible. This is also the value that is produced by empty code blocks. It can be joined[3.3.2] with any value, yielding the other value.

/1 Example

```
Not visible: #none
```

Not visible:

3.5.20 `panic`

Fails with an error.

Arguments are displayed to the user (not rendered in the document) as strings, converting with `repr` if necessary.

/1 Example

The code below produces the error `panicked with: this is wrong`.

```
#panic("this is wrong")
```

/2 Parameters

```
panic(..any)
```

/2.1 `values` `any` *Required* *Positional* *Variadic*

The values to panic with and display to the user.

3.5.21 path

A file system path.

When splitting up your project or package across multiple files, or referencing resources such as images or bibliographies, you'll need to interact with *paths*.

/1 Path strings

Commonly, paths are simply expressed as strings[3.5.27]. Built-in functions that expect paths typically also accept strings. For instance, you can write:

```
#figure(
  // Path to an image
  image("tiger.jpg"),
  caption: [A tiger],
)

// Path to a Typst file
#include "chapter.typ"
```

There are two kinds of such path strings: Relative and absolute.

– A **relative path** resolves in relation to the parent directory of the Typst file where the function is called. While this is the default, a path can also be explicitly specified as being relative by starting it with `./`.

```
#image("images/logo.png")
#image("./images/logo.png") // This is equivalent
```

– An **absolute path** always resolves relative to the *root*[3.5.21/3.1] of the project. Such a path is indicated by a leading `/`:

```
#image("/assets/logo.png")
```

Paths consist of segments that are separated by forward slashes, with interior segments indicating directories and the final one a file or a directory. There are two path components that are treated specially:

– The segment `.` refers to the *current* directory. This is why `"./image.png"` and `"image.png"` are equivalent.

– The segment `..` refers to the *parent* directory. If you have three files `main.typ`, `utils.typ`, and `text/chapter1.typ`, then you can reference your utility file from chapter 1 through the path `"../utils.typ"`.

/2 The path type

For most typical usage of paths, strings are all you need. However, sometimes you need a bit more control. For instance, you may want to resolve a path relative to the file you are currently writing in, but then pass it to a package and let the package read from the path. This is where the path type comes in.

With it, you can fully resolve a path string relative to the file where you construct it. Any following operations performed with the path (such as a file read or an image load), will then behave the same regardless of where in the code they occur.

Here's an example of how we could have a `main.typ` with a `data.json` file directly next to it and still let a package we've built read that file.

```
// This is main.typ, with data.json next to it.
#import "@local/my-pkg:0.1.0": process
#let data-path = path("data.json")
#process(data-path)
```

/3 Roots

/3.1 The project root

For security and reproducibility reasons, Typst encapsulates file access. A Typst project can only access paths within its *project root*. If you try to create or access a path outside of this root, you'll get an error:

```
// ❌ Error: path `"../secret.txt"` would escape the project root
#path("../secret.txt")
```

By default, the project root is the parent directory of the main Typst file. If you wish to use another folder as the root of your project, you can use the `--root` flag:

```
typst compile --root .. file.typ
```

Make sure that the main file is contained in the folder's subtree, so that Typst can access it.

In the web app, the project itself is the root directory. You can always read all files within it, no matter which one is previewed (via the eye toggle next to each Typst file in the file panel).

/3.2 Package roots

Just like the project, each package you import has its own root. Within a package, absolute paths point to the package root rather than the project root. On its own, code in a package cannot construct a path that lives in the project or another package.

If you need to provide a package with resources from the project (such as a logo image), you can do so by explicitly creating a path to the resource in your code with the path constructor[3.5.21/5]. You can then pass the resulting path to the package. An example of this is shown in the section "The path type"[3.5.21/2] above.

Alternatively, you can perform the path operation in your code and pass the result to the package. This could, for example, be the result of a `read`[3.13.5] call or a complete image (e.g. as a named parameter `logo: image("mylogo.svg")`). Note that if you pass an image to a package like this, you can still customize the image's appearance with a set rule within the package.

/4 Further operations

For now, the path type's purpose is limited to correctly handling and transferring paths across files in your project and packages. In the future, it may enable additional capabilities like checking for the existence of a file or enumerating files in a directory.

/5 Constructor

Creates a path from a string.

```
// A relative path without a leading slash.
// May optionally start with `./`.
#path("relative/path/to/file.typ")
#path("./relative/path/to/file.typ")

// An absolute path with a leading slash.
#path("/absolute/path/to/file.typ")
```

```
path( str  path ) → path
```

/5.1 path `str` or `path` *Required* *Positional*

Converts a string or path to a path.

If this is a path string[3.5.21/1]:
- If the path is absolute, it is resolved relative to the root of the project or package in which this function is called.
- If the path is relative, it is resolved relative to the file where this function is called.

If this is already a `path`, it is returned unchanged.

3.5.22 `plugin`

Loads a WebAssembly module.

The resulting module[3.5.18] will contain one Typst function[3.5.15] for each function export of the loaded WebAssembly module.

Typst WebAssembly plugins need to follow a specific protocol[3.5.22/5]. To run as a plugin, a program needs to be compiled to a 32-bit shared WebAssembly library. Plugin functions may accept multiple byte buffers[3.5.6] as arguments and return a single byte buffer. They should typically be wrapped in idiomatic Typst functions that perform the necessary conversions between native Typst types and bytes by leveraging `str`[3.5.27/3], `bytes`[3.5.6/1], and data loading functions[3.13].

For security reasons, plugins run in isolation from your system. This means that printing, reading files, or similar things are not supported.

/1 Example

```
#let myplugin = plugin("hello.wasm")
#let concat(a, b) = str(
  myplugin.concatenate(
    bytes(a),
    bytes(b),
  )
)

#concat("hello", "world")
```

hello*world

Since the plugin function returns a module, it can be used with import syntax:

```
#import plugin("hello.wasm"): concatenate
```

/2 Purity

Plugin functions **must be pure:** A plugin function call must not have any observable side effects on future plugin calls and given the same arguments, it must always return the same value.

The reason for this is that Typst functions must be pure (which is quite fundamental to the language design) and, since Typst function can call plugin functions, this requirement is inherited. In particular, if a plugin function is called twice with the same arguments, Typst might cache the results and call your function only once. Moreover, Typst may run multiple instances of your plugin in multiple threads, with no state shared between them.

Typst does not enforce plugin function purity (for efficiency reasons), but calling an impure function will lead to unpredictable and irreproducible results and must be avoided.

That said, mutable operations *can be* useful for plugins that require costly runtime initialization. Due to the purity requirement, such initialization cannot be performed through a normal function call. Instead, Typst exposes a plugin transition API[3.5.22/8.1], which executes a function call and then creates a derived module with new functions which will observe the side effects produced by the transition call. The original plugin remains unaffected.

/3 Plugins and Packages

Any Typst code can make use of a plugin simply by including a WebAssembly file and loading it. However, because the byte-based plugin interface is quite low-level, plugins are typically exposed through a package containing the plugin and idiomatic wrapper functions.

[A] https://wasi.dev/

/4 WASI

Many compilers will use the WASI ABI[A] by default or as their only option (e.g. emscripten), which allows printing, reading files, etc. This ABI will not directly work with Typst. You will either need to compile to a different target or stub all functions[B].

/5 Protocol

To be used as a plugin, a WebAssembly module must conform to the following protocol:

/5.1 Exports

A plugin module can export functions to make them callable from Typst. To conform to the protocol, an exported function should:

- Take `n` 32-bit integer arguments `a_1`, `a_2`, ..., `a_n` (interpreted as lengths, so `usize/size_t` may be preferable), and return one 32-bit integer.
- The function should first allocate a buffer `buf` of length `a_1 + a_2 + ... + a_n`, and then call `wasm_minimal_protocol_write_args_to_buffer(buf.ptr)`.
- The `a_1` first bytes of the buffer now constitute the first argument, the `a_2` next bytes the second argument, and so on.
- The function can now do its job with the arguments and produce an output buffer. Before returning, it should call `wasm_minimal_protocol_send_result_to_host` to send its result back to the host.
- To signal success, the function should return `0`.
- To signal an error, the function should return `1`. The written buffer is then interpreted as an UTF-8 encoded error message.

/5.2 Imports

Plugin modules need to import two functions that are provided by the runtime. (Types and functions are described using WAT syntax.)

- `(import "typst_env" "wasm_minimal_protocol_write_args_to_buffer" (func (param i32)))`

  Writes the arguments for the current function into a plugin-allocated buffer. When a plugin function is called, it receives the lengths[3.5.22/5.1] of its input buffers as arguments. It should then allocate a buffer whose capacity is at least the sum of these lengths. It should then call this function with a `ptr` to the buffer to fill it with the arguments, one after another.

- `(import "typst_env" "wasm_minimal_protocol_send_result_to_host" (func (param i32 i32)))`

  Sends the output of the current function to the host (Typst). The first parameter shall be a pointer to a buffer (`ptr`), while the second is the length of that buffer (`len`). The memory pointed at by `ptr` can be freed immediately after this function returns. If the message should be interpreted as an error message, it should be encoded as UTF-8.

/6 Resources

For more resources, check out the wasm-minimal-protocol repository[C]. It contains:

- A list of example plugin implementations and a test runner for these examples
- Wrappers to help you write your plugin in Rust
- A stubber for WASI

/7 Parameters

```
plugin( str path bytes ) → module
```

/7.1 source `str` or `path` or `bytes` *Required* *Positional*

A path to a WebAssembly file or raw WebAssembly bytes.

[B] https://github.com/typst-community/wasm-minimal-protocol/tree/main/crates/wasi-stub
[C] https://github.com/typst-community/wasm-minimal-protocol

/8 Definitions

/8.1 transition

Calls a plugin function that has side effects and returns a new module with plugin functions that are guaranteed to have observed the results of the mutable call.

Note that calling an impure function through a normal function call (without use of the transition API) is forbidden and leads to unpredictable behaviour. Read the section on purity[3.5.22/2] for more details.

In the example below, we load the plugin `hello-mut.wasm` which exports two functions: The `get()` function retrieves a global array as a string. The `add(value)` function adds a value to the global array.

We call `add` via the transition API. The call `mutated.get()` on the derived module will observe the addition. Meanwhile the original module remains untouched as demonstrated by the `base.get()` call.

Note: Due to limitations in the internal WebAssembly implementation, the transition API can only guarantee to reflect changes in the plugin's memory, not in WebAssembly globals. If your plugin relies on changes to globals being visible after transition, you might want to avoid use of the transition API for now. We hope to lift this limitation in the future.

```
#let base = plugin("hello-mut.wasm")
#assert.eq(base.get(), "[]")

#let mutated = plugin.transition(base.add, "hello")
#assert.eq(base.get(), "[]")
#assert.eq(mutated.get(), "[hello]")
```

```
plugin.transition(
  function,
  ..bytes,
) → module
```

/8.1.1 func `function` *Required* *Positional*

The plugin function to call.

/8.1.2 arguments `bytes` *Required* *Positional* *Variadic*

The byte buffers to call the function with.

3.5.23 `regex`

A regular expression.

Can be used as a show rule selector[3.2.2] and with string methods[3.5.27] like `find`, `split`, `replace`, and `match`.

See here[A] for a specification of the supported syntax.

/1 Example

```
// Works with string methods.
#"a,b;c".split(regex("[,;]"))

// Works with show rules.
#show regex("\\d+"): set text(red)

The numbers 1 to 10.
```

```
("a", "b", "c")
```

The numbers 1 to 10.

/2 Constructor

Create a regular expression from a string.

```
regex( str ) → regex
```

/2.1 `regex` `str` *Required* *Positional*

The regular expression as a string.

Both Typst strings and regular expressions use backslashes for escaping. To produce a regex escape sequence that is also valid in Typst, you need to escape the backslash itself (e.g., writing `regex("\\\\")` for the regex `\\`). Regex escape sequences that are not valid Typst escape sequences (e.g., `\d` and `\b`) can be entered into strings directly, but it's good practice to still escape them to avoid ambiguity (i.e., `regex("\\b\\d")`). See the list of valid string escape sequences[3.5.27/2].

If you need many escape sequences, you can also create a raw element and extract its text to use it for your regular expressions: `regex(`\d+\.\d+\.\d+`.text)`.

[A] https://docs.rs/regex/latest/regex/#syntax

3.5.24 repr

Returns the string representation of a value.

When inserted into content, most values are displayed as this representation in monospace with syntax-highlighting. The exceptions are `none`, integers, floats, strings, content, and functions.

/1 Example

```
#none vs #repr(none) \
#"hello" vs #repr("hello") \
#(1, 2) vs #repr((1, 2)) \
#[*Hi*] vs #repr([*Hi*])
```

vs none
hello vs "hello"
(1, 2) vs (1, 2)
Hi vs strong(body: [Hi])

/2 For debugging purposes only

This function is for debugging purposes. Its output should not be considered stable and may change at any time.

To be specific, having the same `repr` does not guarantee that values are equivalent, and `repr` is not a strict inverse of `eval`[3.5.13]. In the following example, for readability, the `length`[3.10.13] is rounded to two significant digits and the parameter list and body of the unnamed `function`[3.5.15/6] are omitted.

```
#assert(2pt / 3 < 0.67pt)
#repr(2pt / 3)

#repr(x => x + 1)
```

0.67pt

(..) => ..

/3 Parameters

```
repr( any ) → str
```

/3.1 value `any` *Required* *Positional*

The value whose string representation to produce.

3.5.25 `selector`

A filter for selecting elements within the document.

To construct a selector you can:
- use an element function[3.5.15/2]
- filter for an element function with specific fields[3.5.15/8.2]
- use a string[3.5.27] or regular expression[3.5.23]
- use a `<label>`[3.5.17]
- use a `location`[3.12.4]
- call the `selector`[3.5.25] constructor to convert any of the above types into a selector value and use the methods below to refine it

Selectors are used to apply styling rules[3.2.2] to elements. You can also use selectors to query[3.12.6] the document for certain types of elements.

Furthermore, you can pass a selector to several of Typst's built-in functions to configure their behaviour. One such example is the outline[3.6.14] where it can be used to change which elements are listed within the outline.

Multiple selectors can be combined using the methods shown below. However, not all kinds of selectors are supported in all places, at the moment.

/1 Example

```
#context query(
  heading.where(level: 1)
    .or(heading.where(level: 2))
)

= This will be found
== So will this
=== But this will not.
```

```
(
  heading(
    level: 1,
    depth: 1,
    offset: 0,
    numbering: none,
    supplement: [Section],
    outlined: true,
    bookmarked: auto,
    hanging-indent: auto,
    body: [This will be found],
  ),
  heading(
    level: 2,
    depth: 2,
    offset: 0,
    numbering: none,
    supplement: [Section],
    outlined: true,
    bookmarked: auto,
    hanging-indent: auto,
    body: [So will this],
  ),
)
```

This will be found

So will this

But this will not.

/2 Constructor

Turns a value into a selector. The following values are accepted:
- An element function like a `heading` or `figure`.
- A string[3.5.27] or regular expression[3.5.23].
- A `<label>`.
- A `location`[3.12.4].

– A more complex selector like heading.where(level: 1).

```
selector( str regex label selector location function ) → selector
```

/2.1 **target** str or regex or label or selector or location or function *Required* *Positional*

Can be an element function like a heading or figure, a <label> or a more complex selector like heading.where(level: 1).

/3 Definitions

/3.1 or

Selects all elements that match this or any of the other selectors.

```
self.or(.. str regex label selector location function ) → selector
```

/3.1.1 **others** str or regex or label or selector or location or function *Required* *Positional* *Variadic*

The other selectors to match on.

/3.2 and

Selects all elements that match this and all of the other selectors.

```
self.and(.. str regex label selector location function ) → selector
```

/3.2.1 **others** str or regex or label or selector or location or function *Required* *Positional* *Variadic*

The other selectors to match on.

/3.3 before

Returns a modified selector that will only match elements that occur before the first match of end.

Note: This selector is currently only supported with introspection functions, not in show rules.

```
self.before(
  label selector location function,
  inclusive: bool,
) → selector
```

/3.3.1 **end** label or selector or location or function *Required* *Positional*

The original selection will end at the first match of end.

/3.3.2 **inclusive** bool Default: true

Whether end itself should match or not. This is only relevant if both selectors match the same type of element. Defaults to true.

/3.4 after

Returns a modified selector that will only match elements that occur after the first match of start.

Note: This selector is currently only supported with introspection functions, not in show rules.

```
self.after(
  label selector location function,
  inclusive: bool,
) → selector
```

/3.4.1 **start** label or selector or location or function *Required* *Positional*

The original selection will start at the first match of `start`.

/3.4.2 **inclusive** bool Default: true

Whether `start` itself should match or not. This is only relevant if both selectors match the same type of element. Defaults to `true`.

/3.5 within

Returns a modified selector that will only match elements that are contained within any elements matching the `ancestor` selector.

Example · Finding strong elements in lists

```
*Strong emphasis* that does not count.

- An *important* word
- Another *key* word

Strong elements in lists:
#context {
  query(selector(strong).within(list))
    .map(it => it.body)
    .join[, ]
}
```

Strong emphasis that does not count.

- An **important** word
- Another **key** word

Strong elements in lists: important, key

This can also be used in combination with here[3.12.2] to find all matches of a selector within a context[3.4] expression. This can be quite useful to have an introspection return results local to some component you are building.

Example · Counting elements locally in a context block

```
#let count(sel, body) = context {
  let n = query(selector(sel).within(here())).len()
  [#body (#n matches)]
}

- #count(emph)[Has _two_ matching _elements_]
- #count(strong)[Has *one* matching element]
```

- Has *two* matching *elements* (2 matches)
- Has **one** matching element (1 matches)

Note: This selector is currently only supported with introspection functions, not in show rules.

`self.within(label selector location function) → selector`

/3.5.1 **ancestor** label or selector or location or function *Required* *Positional*

Only matches of `self` that are descendants of any element matching this selector will be included in the output.

An element is not considered its own ancestor.

3.5.26 Standard Library

A module that contains all globally accessible items.

/1 Using “shadowed” definitions

The `std` module is useful whenever you overrode a name from the global scope (this is called *shadowing*). For instance, you might have used the name `text` for a parameter. To still access the `text` element, write `std.text`.

```
#let par = [My special paragraph.]
#let special(text) = {
  set std.text(style: "italic")
  set std.par.line(numbering: "1")
  text
}

#special(par)

#lorem(10)
```

1 *My special paragraph.*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

/2 Conditional access

You can also use this in combination with the dictionary constructor[3.5.11/2] to conditionally access global definitions. This can, for instance, be useful to use new or experimental functionality when it is available, while falling back to an alternative implementation if used on an older Typst version. In particular, this allows us to create polyfills[A].

This can be as simple as creating an alias to prevent warning messages, for example, conditionally using `pattern` in Typst version 0.12, but using `tiling`[3.11.13] in newer versions. Since the parameters accepted by the `tiling` function match those of the older `pattern` function, using the `tiling` function when available and falling back to `pattern` otherwise will unify the usage across all versions. Note that, when creating a polyfill, `sys.version`[3.5.29] can also be very useful.

```
#let tiling = if "tiling" in std { tiling } else { pattern }

...
```

[A] https://en.wikipedia.org/wiki/Polyfill_(programming)

3.5.27 `str`

A sequence of Unicode codepoints.

You can iterate over the grapheme clusters of the string using a for loop[3.3.5]. Grapheme clusters are basically characters but keep together things that belong together, e.g. multiple codepoints that together form a flag emoji. Strings can be added with the + operator, joined together[3.3.2] and multiplied with integers.

Typst provides utility methods for string manipulation. Many of these methods (e.g., `split`[3.5.27/4.20], `trim`[3.5.27/4.19] and `replace`[3.5.27/4.18]) operate on *patterns:* A pattern can be either a string or a regular expression[3.5.23]. This makes the methods quite versatile.

All lengths and indices are expressed in terms of UTF-8 bytes. Indices are zero-based and negative indices wrap around to the end of the string.

You can convert a value to a string with the `str` constructor.

/1 Example

```
#"hello world!" \
#"\"hello\n  world\"!" \
#"1 2 3".split() \
#"1,2;3".split(regex("[,;]")) \
#(regex("\\d+") in "ten euros") \
#(regex("\\d+") in "10 euros")
```

```
hello world!
"hello
 world"!
("1", "2", "3")
("1", "2", "3")
false
true
```

/2 Escape sequences

Just like in markup, you can escape a few symbols in strings:
- `\\` for a backslash
- `\"` for a quote
- `\n` for a newline
- `\r` for a carriage return
- `\t` for a tab
- `\u{1f600}` for a hexadecimal Unicode escape sequence

/3 Constructor

Converts a value to a string.

- Integers are formatted in base 10. This can be overridden with the optional `base` parameter.
- Floats are formatted in base 10 and never in exponential notation.
- Negative integers and floats are formatted with the Unicode minus sign (“−” U+2212) instead of the ASCII minus sign (“-” U+002D).
- From labels the name is extracted.
- Bytes are decoded as UTF-8.

If you wish to convert from and to Unicode code points, see the `to-unicode`[3.5.27/4.8] and `from-unicode`[3.5.27/4.9] functions.

```
#str(10) \
#str(4000, base: 16) \
#str(2.7) \
#str(1e8) \
#str(<intro>)
```

```
10
fa0
2.7
100000000
intro
```

```
str(
  int float decimal str label bytes version type,
  base: int,
) → str
```

/3.1 **value** int or float or decimal or str or label or bytes or version or type
Required *Positional*

The value that should be converted to a string.

/3.2 **base** int Default: 10

The base (radix) to display integers in, between 2 and 36.

/4 Definitions

/4.1 len

The length of the string in UTF-8 encoded bytes.

```
self.len() → int
```

/4.2 first

Extracts the first grapheme cluster of the string.

Returns the provided default value if the string is empty or fails with an error if no default value was specified.

```
self.first(default: str) → str
```

/4.2.1 **default** str

A default value to return if the string is empty.

/4.3 last

Extracts the last grapheme cluster of the string.

Returns the provided default value if the string is empty or fails with an error if no default value was specified.

```
self.last(default: str) → str
```

/4.3.1 **default** str

A default value to return if the string is empty.

/4.4 at

Extracts the first grapheme cluster after the specified index. Returns the default value if the index is out of bounds or fails with an error if no default value was specified.

```
self.at(
  int,
  default: any,
) → any
```

/4.4.1 **index** int *Required* *Positional*

The byte index. If negative, indexes from the back.

/4.4.2 **default** any

A default value to return if the index is out of bounds.

/4.5 slice

Extracts a substring of the string. Fails with an error if the start or end index is out of bounds.

```
self.slice(
  int,
  none int,
  count: int,
) → str
```

/4.5.1 start int *Required* *Positional*

The start byte index (inclusive). If negative, indexes from the back.

/4.5.2 end none or int *Positional* Default: none

The end byte index (exclusive). If omitted, the whole slice until the end of the string is extracted. If negative, indexes from the back.

/4.5.3 count int

The number of bytes to extract. This is equivalent to passing `start + count` as the `end` position. Mutually exclusive with `end`.

/4.6 clusters

Returns the grapheme clusters of the string as an array of substrings.

```
self.clusters() → array
```

/4.7 codepoints

Returns the Unicode codepoints of the string as an array of substrings.

```
self.codepoints() → array
```

/4.8 to-unicode

Converts a character into its corresponding code point.

```
#"a".to-unicode() \
#("a\u{0300}"
   .codepoints()
   .map(str.to-unicode))
```

```
97
(97, 768)
```

```
str.to-unicode(str) → int
```

/4.8.1 character str *Required* *Positional*

The character that should be converted.

/4.9 from-unicode

Converts a unicode code point into its corresponding string.

```
#str.from-unicode(97)
```

```
a
```

```
str.from-unicode(int) → str
```

/4.9.1 **value** int *Required* *Positional*

The code point that should be converted.

/4.10 normalize

Normalizes the string to the given Unicode normal form.

This is useful when manipulating strings containing Unicode combining characters.

```
#assert.eq("é".normalize(form: "nfd"), "e\u{0301}")
#assert.eq("ﬁ".normalize(form: "nfkc"), "fi")
```

```
self.normalize(form: str) → str
```

/4.10.1 **form** str Default: "nfc"

Variant	Details
"nfc"	Canonical composition where e.g. accented letters are turned into a single Unicode codepoint.
"nfd"	Canonical decomposition where e.g. accented letters are split into a separate base and diacritic.
"nfkc"	Like NFC, but using the Unicode compatibility decompositions.
"nfkd"	Like NFD, but using the Unicode compatibility decompositions.

/4.11 contains

Whether the string contains the specified pattern.

This method also has dedicated syntax: You can write "bc" in "abcd" instead of "abcd".contains("bc").

```
self.contains(str regex) → bool
```

/4.11.1 **pattern** str or regex *Required* *Positional*

The pattern to search for.

/4.12 starts-with

Whether the string starts with the specified pattern.

```
self.starts-with(str regex) → bool
```

/4.12.1 **pattern** str or regex *Required* *Positional*

The pattern the string might start with.

/4.13 ends-with

Whether the string ends with the specified pattern.

```
self.ends-with(str regex) → bool
```

/4.13.1 **pattern** str or regex *Required* *Positional*

The pattern the string might end with.

/4.14 find

Searches for the specified pattern in the string and returns the first match as a string or none if there is no match.

```
self.find(str regex) → none str
```

/4.14.1 **pattern** `str` or `regex` *Required* *Positional*

The pattern to search for.

/4.15 **position**

Searches for the specified pattern in the string and returns the index of the first match as an integer or `none` if there is no match.

```
self.position(str regex) → none int
```

/4.15.1 **pattern** `str` or `regex` *Required* *Positional*

The pattern to search for.

/4.16 **match**

Searches for the specified pattern in the string and returns a dictionary with details about the first match or `none` if there is no match.

The returned dictionary has the following keys:
- `start`: The start offset of the match
- `end`: The end offset of the match
- `text`: The text that matched.
- `captures`: An array containing a string for each matched capturing group. The first item of the array contains the first matched capturing, not the whole match! This is empty unless the `pattern` was a regex with capturing groups.

Example · Shape of the returned dictionary

```
#let pat = regex("not (a|an) (apple|cat)")
#"I'm a doctor, not an apple.".match(pat) \
#"I am not a cat!".match(pat)
```

```
(
  start: 14,
  end: 26,
  text: "not an apple",
  captures: ("an", "apple"),
)
(
  start: 5,
  end: 14,
  text: "not a cat",
  captures: ("a", "cat"),
)
```

Example · Different kinds of patterns

```
#assert.eq("Is there a".match("for this?"), none)
#"The time of my life.".match(regex("[mit]+e"))
```

```
(start: 4, end: 8, text: "time",
captures: ())
```

```
self.match(str regex) → none dictionary
```

/4.16.1 **pattern** str or regex *Required* *Positional*

The pattern to search for.

/4.17 matches

Searches for the specified pattern in the string and returns an array of dictionaries with details about all matches. For details about the returned dictionaries, see above[3.5.27/4.16].

```
#"Day by Day.".matches("Day")
```

```
(
  (start: 0, end: 3, text: "Day",
captures: ()),
  (start: 7, end: 10, text: "Day",
captures: ()),
)
```

```
self.matches( str regex ) → array
```

/4.17.1 **pattern** str or regex *Required* *Positional*

The pattern to search for.

/4.18 replace

Replace at most `count` occurrences of the given pattern with a replacement string or function (beginning from the start). If no count is given, all occurrences are replaced.

```
self.replace(
  str regex ,
  str function ,
  count: int ,
) → str
```

/4.18.1 **pattern** str or regex *Required* *Positional*

The pattern to search for.

/4.18.2 **replacement** str or function *Required* *Positional*

The string to replace the matches with or a function that gets a dictionary for each match and can return individual replacement strings.

The dictionary passed to the function has the same shape as the dictionary returned by `match`[3.5.27/4.16].

/4.18.3 **count** int

If given, only the first `count` matches of the pattern are replaced.

/4.19 trim

Removes matches of a pattern from one or both sides of the string, once or repeatedly and returns the resulting string.

```
self.trim(
  none str regex ,
  at: alignment ,
  repeat: bool ,
) → str
```

/4.19.1 **pattern** none or str or regex *Positional* Default: none

The pattern to search for. If none, trims white spaces.

/4.19.2 **at** alignment

Can be start or end to only trim the start or end of the string. If omitted, both sides are trimmed.

/4.19.3 **repeat** bool Default: true

Whether to repeatedly removes matches of the pattern or just once. Defaults to true.

/4.20 split

Splits a string at matches of a specified pattern and returns an array of the resulting parts.

When the empty string is used as a separator, it separates every character (i.e., Unicode code point) in the string, along with the beginning and end of the string. In practice, this means that the resulting list of parts will contain the empty string at the start and end of the list.

```
self.split(none str regex) → array
```

/4.20.1 **pattern** none or str or regex *Positional* Default: none

The pattern to split at. Defaults to whitespace.

/4.21 rev

Reverses the string.

More specifically, this returns a string with the same grapheme clusters, in reversed order.

```
#"Pirate flag: ☠".rev()
```

☠:galf etariP

```
self.rev() → str
```

3.5.28 `symbol`

A Unicode symbol.

Typst defines common symbols so that they can easily be written with standard keyboards. The symbols are defined in modules, from which they can be accessed using field access notation[3.3.6]:

- General symbols are defined in the `sym` module[3.9.2] and are accessible without the `sym.` prefix in math mode.
- Emoji are defined in the `emoji` module[3.9.3]

Moreover, you can define custom symbols with this type's constructor function.

```
#sym.arrow.r \
#sym.gt.eq.not \
$gt.eq.not$ \
#emoji.face.halo
```

→
≱
😇

Many symbols have different variants, which can be selected by appending the modifiers with dot notation. The order of the modifiers is not relevant. Visit the documentation pages of the symbol modules and click on a symbol to see its available variants.

```
$arrow.l$ \
$arrow.r$ \
$arrow.t.quad$
```

←
→
⟰

/1 Constructor

Create a custom symbol with modifiers.

```
#let envelope = symbol(
  "🖂",
  ("stamped", "🖃"),
  ("stamped.pen", "🖆"),
  ("lightning", "🖄"),
  ("fly", "🖅"),
)

#envelope
#envelope.stamped
#envelope.stamped.pen
#envelope.lightning
#envelope.fly
```

🖂 🖃 🖆 🖄 🖅

`symbol(.. str array) → symbol`

/1.1 `variants` `str` or `array` *Required* *Positional* *Variadic*

The variants of the symbol.

Can be a just a string consisting of a single character for the modifierless variant or an array with two strings specifying the modifiers and the symbol. Individual modifiers should be separated by dots. When displaying a symbol, Typst selects the first from the variants that have all attached modifiers and the minimum number of other modifiers.

3.5.29 System

Module for system interactions.

This module defines the following items:

- The `sys.version` constant (of type `version`[3.5.32]) that specifies the currently active Typst compiler version.
- The `sys.inputs` dictionary[3.5.11], which makes external inputs available to the project. An input specified in the command line as `--input key=value` becomes available under `sys.inputs.key` as `"value"`. To include spaces in the value, it may be enclosed with single or double quotes.

  The value is always of type string[3.5.27]. More complex data may be parsed manually using functions like `json`[3.13.4].

3.5.30 `target` Contextual

Returns the current export target.

This function returns
- `"paged"` in PDF[3.14], PNG[3.16], and SVG[3.17] export, or within an HTML frame[3.15.4]
- `"html"` in HTML[3.15] export
- `"bundle"` in Bundle[3.18] export

/1 When to use it

This function allows you to format your document properly across the paged, HTML, and multi file export targets. It should primarily be used in templates and show rules, rather than directly in content. This way, the document's contents can be fully agnostic to the export target and content can be shared between different export targets.

/2 Varying targets

This function is contextual[3.4] as the target can vary within a single compilation: When exporting to HTML, the target will be `"paged"` while within an `html.frame`[3.15.4].

/3 Example

```
#let kbd(it) = context {
  if target() == "html" {
    html.elem("kbd", it)
  } else {
    set text(fill: rgb("#1f2328"))
    let r = 3pt
    box(
      fill: rgb("#f6f8fa"),
      stroke: rgb("#d1d9e0b3"),
      outset: (y: r),
      inset: (x: r),
      radius: r,
      raw(it)
    )
  }
}

Press #kbd("F1") for help.
```

Press `F1` for help.

/4 Parameters

```
target() → str
```

3.5.31 type

Describes a kind of value.

To style your document, you need to work with values of different kinds: Lengths specifying the size of your elements, colors for your text and shapes, and more. Typst categorizes these into clearly defined *types* and tells you where it expects which type of value.

Apart from basic types for numeric values and typical[3.5.16] types[3.5.14] known[3.5.27] from[3.5.2] programming[3.5.11] languages, Typst provides a special type for *content*.[3.5.8] A value of this type can hold anything that you can enter into your document: Text, elements like headings and shapes, and style information.

/1 Example

```
#let x = 10
#if type(x) == int [
  #x is an integer!
] else [
  #x is another value...
]

An image is of type
#type(image("glacier.jpg")).
```

10 is an integer!

An image is of type content.

The type of 10 is int. Now, what is the type of int or even type?

```
#type(int) \
#type(type)
```

type
type

Unlike other types like int, none[3.5.19] and auto[3.5.4] do not have a name representing them. To test if a value is one of these, compare your value to them directly, e.g:

```
#let val = none
#if val == none [
  Yep, it's none.
]
```

Yep, it's none.

Note that type will return content[3.5.8] for all document elements. To programmatically determine which kind of content you are dealing with, see content.func[3.5.8/3.1].

/2 Constructor

Determines a value's type.

```
#type(12) \
#type(14.7) \
#type("hello") \
#type(<glacier>) \
#type([Hi]) \
#type(x => x + 1) \
#type(type)
```

int
float
str
label
content
function
type

type(any) → type

/2.1 **value** any *Required* *Positional*

The value whose type's to determine.

3.5.32 `version`

A version with an arbitrary number of components.

The first three components have names that can be used as fields: `major`, `minor`, `patch`. All following components do not have names.

The list of components is semantically extended by an infinite list of zeros. This means that, for example, `0.8` is the same as `0.8.0`. As a special case, the empty version (that has no components at all) is the same as `0`, `0.0`, `0.0.0`, and so on.

The current version of the Typst compiler is available as `sys.version`.

You can convert a version to an array of explicitly given components using the `array`[3.5.2] constructor.

/1 Constructor

Creates a new version.

It can have any number of components (even zero).

Example · Constructing versions

```
#version() \
#version(1) \
#version(1, 2, 3, 4) \
#version((1, 2, 3, 4)) \
#version((1, 2), 3)
```

1
1.2.3.4
1.2.3.4
1.2.3

As a practical use case, this allows comparing the current version (`sys.version`[3.5.32]) to a specific one.

Example · Comparing with the current version

```
Current version: #sys.version \
#(sys.version >= version(0, 14, 0)) \
#(version(3, 2, 0) > version(4, 1, 0))
```

Current version: 0.15.1
true
false

```
version(.. int array) → version
```

/1.1 **components** `int` or `array` *Required* *Positional* *Variadic*

The components of the version (array arguments are flattened)

/2 Definitions

/2.1 **at**

Retrieves a component of a version.

The returned integer is always non-negative. Returns `0` if the version isn't specified to the necessary length.

```
self.at(int) → int
```

/2.1.1 **index** `int` *Required* *Positional*

The index at which to retrieve the component. If negative, indexes from the back of the explicitly given components.

3.6 Model

Document structuring.

Here, you can find functions to structure your document and interact with that structure. This includes section headings, figures, bibliography management, cross-referencing and more.

Definitions

3.6.1 Asset Adds a custom file to a bundle. . . . 145
3.6.2 Bibliography A bibliography / reference listing. . . . 146
3.6.3 Bullet List A bullet list. . . . 153
3.6.4 Cite Cite a work from the bibliography. . . . 156
3.6.5 Divider A thematic break that separates sections of content. . . . 161
3.6.6 Document Manages metadata and is used to add a document file to a bundle. . . . 162
3.6.7 Emphasis Emphasizes content by toggling italics. . . . 165
3.6.8 Figure A figure with an optional caption. . . . 166
3.6.9 Footnote A footnote. . . . 173
3.6.10 Heading A section heading. . . . 176
3.6.11 Link Links to a URL or a location in the document. . . . 180
3.6.12 Numbered List A numbered list. . . . 183
3.6.13 Numbering Applies a numbering to a sequence of numbers. . . . 187
3.6.14 Outline A table of contents, figures, or other elements. . . . 189
3.6.15 Paragraph A logical subdivison of textual content. . . . 195
3.6.16 Paragraph Break A paragraph break. . . . 203
3.6.17 Quote Displays a quote alongside an optional attribution. . . . 204
3.6.18 Reference A reference to a label or bibliography. . . . 208
3.6.19 Strong Emphasis Strongly emphasizes content by increasing the font weight. . . . 211
3.6.20 Table A table of items. . . . 212
3.6.21 Term List A list of terms and their descriptions. . . . 222
3.6.22 Title A document title. . . . 224

3.6.1 `asset` Element

Adds a custom file to a bundle.

This function creates a single file in a bundle[3.18], from raw byte data[3.5.6]. Unlike documents[3.6.6], assets will be emitted as-is without undergoing compilation.

The `asset` function can be combined with `read`[3.13.5] to copy a file from the project into the output bundle. The first argument to `asset` defines the output path for the asset in the bundle, while the path passed to `read` defines where in the project to read the data from.

```
// Copy the file `styles.css` into the bundle.
#asset("styles.css", read("styles.css"))
```

That said, `asset` is not tied to `read`. You can also generate bytes directly or use a function like `json.encode`[3.13.4/4.1] to emit serialized data.

```
// Emits a JSON file with the number
// of headings in the document.
#context {
  let headings = query(heading)
  let meta = (
    count: headings.len(),
  )
  asset("meta.json", json.encode(meta))
}

#document("doc.pdf")[
  = Introduction
  = Conclusion
]
```

This would emit a `meta.json` file with the following contents into the resulting bundle:

```
{
  "count": 2
}
```

This function may only be used in the bundle[3.18] target.

/1 Parameters

```
asset(
  str,
  str bytes,
) → content
```

/1.1 **path** `str` *Required* *Positional*

The path in the bundle at which the asset will be placed.

May contain interior slashes, in which case intermediate directories will be automatically created.

/1.2 **data** `str` or `bytes` *Required* *Positional*

The raw data that will be written into the file at the specified path.

If a string is given, it will be encoded using UTF-8.

3.6.2 `bibliography` Element

A bibliography / reference listing.

You can create a new bibliography by calling this function with a path to a bibliography file in either one of two formats:

- A Hayagriva `.yaml`/`.yml` file. Hayagriva is a new bibliography file format designed for use with Typst. Visit its documentation[A] for more details.
- A BibLaTeX `.bib` file.

As soon as you add a bibliography somewhere in your document, you can start citing things with reference syntax (`@key`) or explicit calls to the citation[3.6.4] function (`#cite(<key>)`). The bibliography will only show entries for works that were referenced in the document.

/1 Example

```
This was already noted by
pirates long ago. @arrgh

Multiple sources say ...
@arrgh @netwok.

#bibliography("works.bib")
```

This was already noted by pirates long ago. [1]

Multiple sources say ... [1], [2].

Bibliography

[1] P. T. Leeson, "The Pirate Organization."

[2] R. Astley and L. Morris, "At-scale impact of the Net Wok: A culinarically holistic investigation of distributed dumplings," *Armenian Journal of Proceedings*, vol. 61, pp. 192–219, 2020.

/2 Styles

Typst offers a wide selection of built-in citation and bibliography styles[3.6.2/4.4]. Beyond those, you can add and use custom CSL[B] (Citation Style Language) files. Wondering which style to use? Here are some good defaults based on what discipline you're working in:

Fields	Typical Styles
Engineering, IT	`"ieee"`
Psychology, Life Sciences	`"apa"`
Social sciences	`"chicago-author-date"`
Humanities	`"mla"`, `"chicago-notes"`, `"harvard-cite-them-right"`
Economics	`"harvard-cite-them-right"`
Physics	`"american-physics-society"`

/3 Multiple bibliographies

When a Typst document contains multiple bibliographies, each citation is assigned to one of them. By default, Typst will automatically pick a suitable bibliography (typically, the closest following one that contains the referenced citation key). This covers common cases like by-chapter or thematic bibliographies. For more fine-grained control, citations can be explicitly targeted by a bibliography through a `target`[3.6.2/4.5] selector.

[A] https://github.com/typst/hayagriva/blob/main/docs/file-format.md
[B] https://citationstyles.org/

/4 Parameters

```
bibliography(
  str path bytes array,
  title: none auto content,
  full: bool,
  style: str path bytes,
  target: auto label selector location function,
  group: none auto str,
) → content
```

/4.1 sources `str` or `path` or `bytes` or `array` *Required* *Positional*

One or multiple paths to or raw bytes for Hayagriva `.yaml` and/or BibLaTeX `.bib` files.

This can be a:
- A path string or path[3.5.21] to load a bibliography file from.
- Raw bytes from which the bibliography should be decoded.
- An array where each item is one of the above.

/4.2 title `none` or `auto` or `content` *Settable* Default: `auto`

The title of the bibliography.

- When set to `auto`, an appropriate title for the text language[3.7.12/2.16] will be used. This is the default.
- When set to `none`, the bibliography will not have a title.
- A custom title can be set by passing content.

The bibliography's heading will not be numbered by default, but you can force it to be with a show-set rule: `show bibliography: set heading(numbering: "1.")`

/4.3 full `bool` *Settable* Default: `false`

Whether to include all works from the given bibliography files, even those that weren't cited in the document.

To selectively add individual cited works without showing them, you can also use the `cite` function with `form`[3.6.4/3.3] set to `none`.

/4.4 style `str` or `path` or `bytes` *Settable* Default: `"ieee"`

The bibliography style.

This can be:
- A string with the name of one of the built-in styles (see below). Some of the styles listed below appear twice, once with their full name and once with a short alias.
- A path string or path[3.5.21] to a CSL file[A].
- Raw bytes from which a CSL style should be decoded.

Variant	Details
`"alphanumeric"`	Alphanumeric
`"american-anthropological-association"`	American Anthropological Association
`"american-chemical-society"`	ACS Guide 2022 revision
`"american-geophysical-union"`	American Geophysical Union
`"american-institute-of-aeronautics-and-astronautics"`	American Institute of Aeronautics and Astronautics
`"american-institute-of-physics"`	AIP Style Manual 4th edition
`"american-medical-association"`	AMA Manual of Style 11th edition
`"american-meteorological-society"`	American Meteorological Society
`"american-physics-society"`	American Physical Society

[A] https://citationstyles.org/

Variant	Details
`"american-physiological-society"`	American Physiological Society
`"american-political-science-association"`	APSA Style Manual revised 2018 edition
`"american-psychological-association"`	APA Style 7th edition
`"apa"`	A short alias of `american-psychological-association`
`"american-society-for-microbiology"`	American Society for Microbiology
`"american-society-of-civil-engineers"`	American Society of Civil Engineers
`"american-society-of-mechanical-engineers"`	American Society of Mechanical Engineers
`"american-sociological-association"`	ASA Style Guide 6th/7th edition
`"angewandte-chemie"`	Angewandte Chemie International Edition
`"annual-reviews"`	Annual Reviews (sorted by order of appearance)
`"annual-reviews-author-date"`	Annual Reviews (author-date)
`"associacao-brasileira-de-normas-tecnicas"`	Associação Brasileira de Normas Técnicas (Português - Brasil)
`"association-for-computing-machinery"`	Association for Computing Machinery
`"biomed-central"`	BioMed Central
`"bristol-university-press"`	Bristol University Press
`"british-medical-journal"`	BMJ
`"bmj"`	A short alias of `british-medical-journal`
`"cell"`	Cell
`"chicago-author-date"`	Chicago Manual of Style 18th edition (author-date)
`"chicago-notes"`	Chicago Manual of Style 18th edition (notes and bibliography)
`"chicago-fullnotes"`	A short alias of `chicago-notes`
`"chicago-shortened-notes"`	Chicago Manual of Style 18th edition (shortened notes and bibliography)
`"copernicus"`	Copernicus Publications
`"cse-citation-sequence-brackets-8th-edition"`	CSE Manual: Scientific Style and Format 8th edition (citation-sequence, brackets)
`"council-of-science-editors"`	A short alias of `cse-citation-sequence-brackets-8th-edition`
`"cse-name-year"`	CSE Manual: Scientific Style and Format 9th edition (name-year)
`"council-of-science-editors-author-date"`	A short alias of `cse-name-year`
`"current-opinion"`	Current Opinion journals
`"deutsche-gesellschaft-für-psychologie"`	Deutsche Gesellschaft für Psychologie 5. Auflage (Deutsch)
`"deutsche-sprache"`	Deutsche Sprache (Deutsch)
`"elsevier-harvard"`	Elsevier (author-date/Harvard, with titles)
`"elsevier-vancouver"`	Elsevier - NLM/Vancouver (citation-sequence)
`"elsevier-with-titles"`	Elsevier (numeric, with titles)
`"frontiers"`	Frontiers journals
`"future-medicine"`	Future Medicine journals
`"future-science"`	Future Science Group
`"gb-7714-2005-numeric"`	China National Standard GB/T 7714-2005 (numeric, 中文)
`"gb-7714-2015-author-date"`	China National Standard GB/T 7714-2015 (author-date, 中文)

Variant	Details
`"gb-7714-2015-note"`	China National Standard GB/T 7714-2015 (note, 中文)
`"gb-7714-2015-numeric"`	China National Standard GB/T 7714-2015 (numeric, 中文)
`"gost-r-705-2008-numeric"`	Russian GOST R 7.0.5-2008 (numeric)
`"harvard-cite-them-right"`	Cite Them Right 12th edition (author-date/ Harvard)
`"institute-of-electrical-and-electronics-engineers"`	IEEE Reference Guide version 11.29.2023
`"ieee"`	A short alias of `institute-of-electrical-and-electronics-engineers`
`"institute-of-physics-numeric"`	Institute of Physics (numeric)
`"iso-690-author-date"`	ISO-690 (author-date, English)
`"iso-690-numeric"`	ISO-690 (numeric, English)
`"karger"`	Karger journals
`"mary-ann-liebert-vancouver"`	Mary Ann Liebert - NLM/Vancouver
`"modern-humanities-research-association-notes"`	MHRA Style Guide 4th edition (notes)
`"modern-humanities-research-association"`	A short alias of `modern-humanities-research-association-notes`
`"modern-language-association"`	MLA Handbook 9th edition (in-text citations)
`"mla"`	A short alias of `modern-language-association`
`"modern-language-association-8"`	A short alias of `modern-language-association`
`"mla-8"`	A short alias of `modern-language-association`
`"multidisciplinary-digital-publishing-institute"`	Multidisciplinary Digital Publishing Institute
`"nature"`	Nature
`"nlm-citation-sequence"`	NLM/Vancouver: Citing Medicine 2nd edition (citation-sequence)
`"vancouver"`	A short alias of `nlm-citation-sequence`
`"nlm-citation-sequence-superscript"`	NLM/Vancouver: Citing Medicine 2nd edition (citation-sequence, superscript)
`"vancouver-superscript"`	A short alias of `nlm-citation-sequence-superscript`
`"pensoft"`	Pensoft Journals
`"public-library-of-science"`	Public Library of Science
`"plos"`	A short alias of `public-library-of-science`
`"royal-society-of-chemistry"`	Royal Society of Chemistry
`"sage-vancouver"`	SAGE - NLM/Vancouver
`"sist02"`	SIST02 (日本語)
`"spie"`	SPIE journals
`"springer-basic"`	Springer - Basic (numeric, brackets)
`"springer-basic-author-date"`	Springer - Basic (author-date)
`"springer-fachzeitschriften-medizin-psychologie"`	Springer - Fachzeitschriften Medizin Psychologie (Deutsch)
`"springer-humanities-author-date"`	Springer - Humanities (author-date)
`"springer-lecture-notes-in-computer-science"`	Springer - Lecture Notes in Computer Science
`"springer-mathphys"`	Springer - MathPhys (numeric, brackets)
`"springer-socpsych-author-date"`	Springer - SocPsych (author-date)

Variant	Details
`"springer-vancouver"`	Springer - NLM/Vancouver (citation-sequence, brackets)
`"taylor-and-francis-chicago-author-date"`	Taylor & Francis Journals Standard Reference Style Guide: Chicago author-date version 2.0
`"taylor-and-francis-national-library-of-medicine"`	Taylor & Francis - NLM/Vancouver (citation-sequence, brackets)
`"the-institution-of-engineering-and-technology"`	The Institution of Engineering and Technology
`"the-lancet"`	The Lancet
`"thieme"`	Thieme-German (Deutsch)
`"trends"`	Trends journals
`"turabian-author-date"`	Chicago Manual of Style 17th edition (author-date)
`"turabian-fullnote-8"`	Chicago Manual of Style 17th edition (notes and bibliography, subsequent author-title)

/4.5 **target** `auto` or `label` or `selector` or `location` or `function` *Settable* Default: `auto`

Defines which citations to include in the bibliography.

Typst will automatically assign each citation in the document to a bibliography. Concretely, a citation will be assigned to (in order of precedence)

1. the first bibliography that includes it in its `target` selector; or if no such bibliography exists
2. the closest *following* bibliography with `target: auto` that contains its key; or if no such bibliography follows
3. the closest *preceding* bibliography with `target: auto` that contains its key.

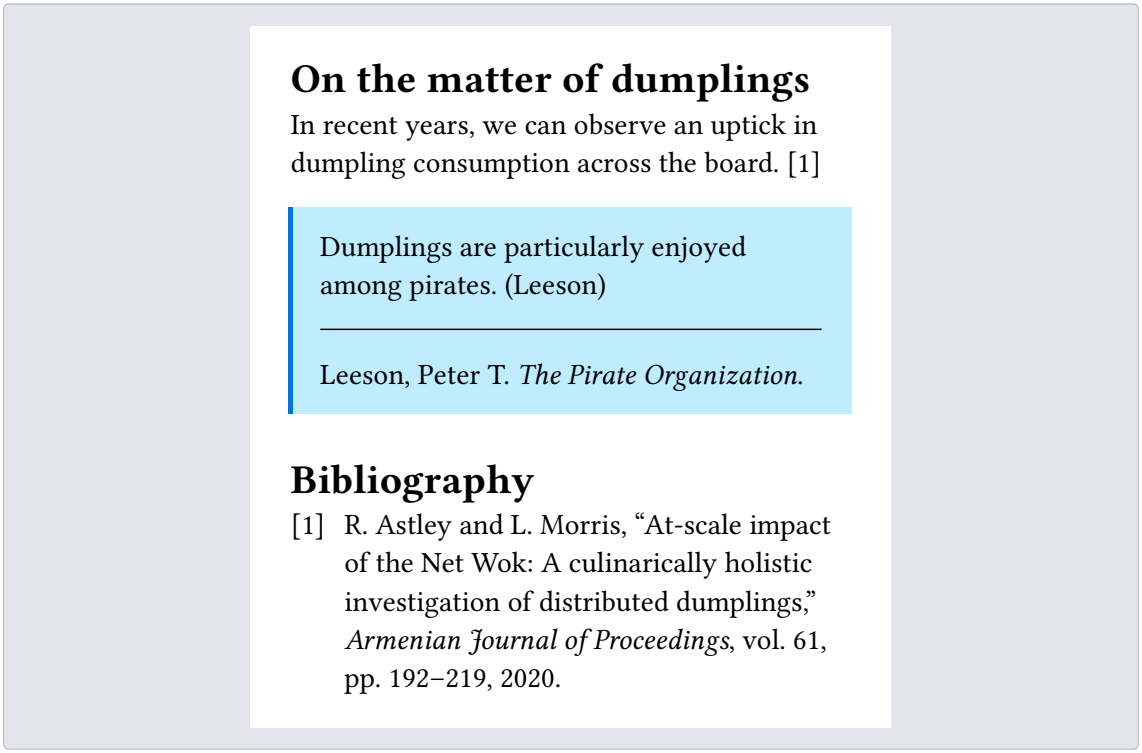
Example · Local bibliography

```
#let info(body) = block(
  stroke: (left: 1.5pt + blue),
  fill: aqua.lighten(50%),
  inset: 1em,
  context {
    body
    show divider: set block(spacing: 1.2em)
    divider()
    bibliography(
      "works.bib",
      title: none,
      target: selector(cite).within(here()),
      style: "mla",
    )
  }
)

= On the matter of dumplings
In recent years, we can observe an uptick in
dumpling consumption across the board. @netwok

#info[
  Dumplings are particularly enjoyed
  among pirates. @arrgh
]

#bibliography("works.bib")
```



On the matter of dumplings

In recent years, we can observe an uptick in dumpling consumption across the board. [1]

> Dumplings are particularly enjoyed among pirates. (Leeson)
>
> ---
>
> Leeson, Peter T. *The Pirate Organization.*

Bibliography

[1] R. Astley and L. Morris, "At-scale impact of the Net Wok: A culinarically holistic investigation of distributed dumplings," *Armenian Journal of Proceedings*, vol. 61, pp. 192–219, 2020.

/4.6 **group** `none` or `auto` or `str` *Settable* Default: `auto`

Conceptually groups this bibliography with other bibliographies for numbering purposes. Bibliographies in the same group will assign consecutive citation numbers.

This can be:
- `none`: The bibliography will be numbered in isolation.
- `auto`: The bibliography will be consecutively numbered with all other bibliographies in the `auto` group.
- A string[3.5.27]: The bibliography will be consecutively numbered with all other bibliographies with the same `group` value.

The `auto` group works just like any string group, but it is the canonical default group.

Example · Consecutive citation numbers

```
#show bibliography: set heading(
  offset: 1,
)

= First part
Starts at one: @netwok @arrgh
#bibliography(
  "works.bib",
  style: "ieee",
)

= Second part
Continues with three: @distress
#bibliography(
  "works.bib",
  style: "nlm-citation-sequence",
)
```

First part

Starts at one: [1], [2]

Bibliography

[1] R. Astley and L. Morris, "At-scale impact of the Net Wok: A culinarically holistic investigation of distributed dumplings," *Armenian Journal of Proceedings*, vol. 61, pp. 192–219, 2020.

[2] P. T. Leeson, "The Pirate Organization."

Second part

Continues with three: (3)

Bibliography

3. An Insight into Bibliographical Distress.

Example · Separate citation numbers

```
#show bibliography: set heading(
  offset: 1,
)
#set bibliography(group: none)

= First part
Starts at one: @netwok @arrgh
#bibliography(
  "works.bib",
  style: "ieee",
)

= Second part
Resets to one: @distress
#bibliography(
  "works.bib",
  style: "nlm-citation-sequence",
)
```

First part

Starts at one: [1], [2]

Bibliography

[1] R. Astley and L. Morris, "At-scale impact of the Net Wok: A culinarically holistic investigation of distributed dumplings," *Armenian Journal of Proceedings*, vol. 61, pp. 192–219, 2020.

[2] P. T. Leeson, "The Pirate Organization."

Second part

Resets to one: (1)

Bibliography

1. An Insight into Bibliographical Distress.

3.6.3 `list` Element

A bullet list.

Displays a sequence of items vertically, with each item introduced by a marker.

/1 Example

```
Normal list.
- Text
- Math
- Layout
- ...

Multiple lines.
- This list item spans multiple
  lines because it is indented.

Function call.
#list(
  [Foundations],
  [Calculate],
  [Construct],
  [Data Loading],
)
```

Normal list.
- Text
- Math
- Layout
- ...

Multiple lines.
- This list item spans multiple lines because it is indented.

Function call.
- Foundations
- Calculate
- Construct
- Data Loading

/2 Syntax

This function also has dedicated syntax: Start a line with a hyphen, followed by a space to create a list item. A list item can contain multiple paragraphs and other block-level content. All content that is indented more than an item's marker becomes part of that item.

/3 Parameters

```
list(
  tight: bool,
  marker: content array function,
  indent: length,
  body-indent: length,
  spacing: auto length,
  marker-align: alignment,
  ..content,
) → content
```

/3.1 `tight` `bool` *Settable* Default: `true`

Defines the default spacing[3.6.3/3.5] of the list. If it is `false`, the items are spaced apart with paragraph spacing[3.6.15/4.2]. If it is `true`, they use paragraph leading[3.6.15/4.1] instead. This makes the list more compact, which can look better if the items are short.

In markup mode, the value of this parameter is determined based on whether items are separated with a blank line. If items directly follow each other, this is set to `true`; if items are separated by a blank line, this is set to `false`. The markup-defined tightness cannot be overridden with set rules.

```
- If a list has a lot of text, and
  maybe other inline content, it
  should not be tight anymore.

- To make a list wide, simply insert
  a blank line between the items.
```

- If a list has a lot of text, and maybe other inline content, it should not be tight anymore.
- To make a list wide, simply insert a blank line between the items.

/3.2 **marker** `content` or `array` or `function` *Settable* Default: `([•], [‣], [–])`

The marker which introduces each item.

Instead of plain content, you can also pass an array with multiple markers that should be used for nested lists. If the list nesting depth exceeds the number of markers, the markers are cycled. For total control, you may pass a function that maps the list's nesting depth (starting from `0`) to a desired marker.

```
#set list(marker: [--])
- A more classic list
- With en-dashes

#set list(marker: ([•], [--]))
- Top-level
  - Nested
  - Items
- Items
```

- A more classic list
- With en-dashes

- Top-level
  - Nested
  - Items
- Items

/3.3 **indent** `length` *Settable* Default: `0pt`

The indent of each item.

/3.4 **body-indent** `length` *Settable* Default: `0.5em`

The spacing between the marker and the body of each item.

/3.5 **spacing** `auto` or `length` *Settable* Default: `auto`

The spacing between the items of the list.

If set to `auto`, uses paragraph `leading`[3.6.15/4.1] for tight lists and paragraph `spacing`[3.6.15/4.2] for wide (non-tight) lists.

/3.6 **marker-align** `alignment` *Settable* Default: `end`

Alignment to use for list markers.

Vertical alignment is always relative to the height of the list items they are attached to. By default, it is unspecified, which means that each marker will vertically align with the baseline of the item it is attached to (which is usually its first line of text, or otherwise its top).

Horizontal alignment, on the other hand, is relative to other markers at the same list level. By default, it is set to `end`, meaning that markers line up towards the end of the current text direction (`right` for LTR, `left` for RTL text). However, since markers at each level are usually identical, it is expected that horizontal alignment has no actual effect, most of the time. Regardless, it is still possible for it to make a difference in some cases, particularly if markers use counters or other forms of state to display different content for each item.

```
- Baseline \
  aligned

#set list(marker-align: horizon)

- Horizon \
  aligned
```

- Baseline
  aligned
- Horizon
  aligned

/3.7 **children** `content` *Required* *Positional* *Variadic*

The bullet list's children.

When using the list syntax, adjacent items are automatically collected into lists, even through constructs like for loops.

```
#for letter in "ABC" [
  - Letter #letter
]
```

- Letter A
- Letter B
- Letter C

/4 Definitions

/4.1 **item** Element

A bullet list item.

```
list.item(content) → content
```

/4.1.1 **body** `content` *Required* *Positional*

The item's body.

3.6.4 `cite` Element

Cite a work from the bibliography.

Before you starting citing, you need to add a bibliography[3.6.2] somewhere in your document.

/1 Example

```
This was already noted by
pirates long ago. @arrgh

Multiple sources say ...
@arrgh @netwok.

You can also call `cite`
explicitly. #cite(<arrgh>)

#bibliography("works.bib")
```

This was already noted by pirates long ago. [1]

Multiple sources say ... [1], [2].

You can also call `cite` explicitly. [1]

Bibliography

[1] P. T. Leeson, "The Pirate Organization."

[2] R. Astley and L. Morris, "At-scale impact of the Net Wok: A culinarically holistic investigation of distributed dumplings," *Armenian Journal of Proceedings*, vol. 61, pp. 192–219, 2020.

If your source name contains certain characters such as slashes, which are not recognized by the <> syntax, you can explicitly call `label` instead.

```
Computer Modern is an example of a modernist serif typeface.
#cite(label("DBLP:books/lib/Knuth86a")).
```

/2 Syntax

This function indirectly has dedicated syntax. References[3.6.18] can be used to cite works from the bibliography. The label then corresponds to the citation key.

/3 Parameters

```
cite(
  label,
  supplement: none content,
  form: none str,
  style: auto str path bytes,
) → content
```

/3.1 key `label` *Required* *Positional*

The citation key that identifies the entry in the bibliography that shall be cited, as a label.

```
// All the same
@netwok \
#cite(<netwok>) \
#cite(label("netwok"))
```

(Astley & Morris, 2020)
(Astley & Morris, 2020)
(Astley & Morris, 2020)

/3.2 supplement `none` or `content` *Settable* Default: `none`

A supplement for the citation such as page or chapter number.

In reference syntax, the supplement can be added in square brackets:

```
This has been proven. @distress[p.~7]

#bibliography("works.bib")
```

This has been proven. [1, p. 7]

Bibliography

[1] B. Aldrin, "An Insight into Bibliographical Distress."

/3.3 **form** `none` or `str` *Settable* Default: `"normal"`

The kind of citation to produce. Different forms are useful in different scenarios: A normal citation is useful as a source at the end of a sentence, while a "prose" citation is more suitable for inclusion in the flow of text.

If set to `none`, the cited work is included in the bibliography, but nothing will be displayed.

```
#cite(<netwok>, form: "prose")
show the outsized effects of
pirate life on the human psyche.
```

Astley & Morris (2020) show the outsized effects of pirate life on the human psyche.

Variant	Details
`"normal"`	Display in the standard way for the active style.
`"prose"`	Produces a citation that is suitable for inclusion in a sentence.
`"full"`	Mimics a bibliography entry, with full information about the cited work.
`"author"`	Shows only the cited work's author(s).
`"year"`	Shows only the cited work's year.

/3.4 **style** `auto` or `str` or `path` or `bytes` *Settable* Default: `auto`

The citation style.

This can be:
- `auto` to automatically use the bibliography's style[3.6.2/4.4] for citations.
- A string with the name of one of the built-in styles (see below). Some of the styles listed below appear twice, once with their full name and once with a short alias.
- A path string or `path`[3.5.21] to a CSL file[A].
- Raw bytes from which a CSL style should be decoded.

Variant	Details
`"alphanumeric"`	Alphanumeric
`"american-anthropological-association"`	American Anthropological Association
`"american-chemical-society"`	ACS Guide 2022 revision
`"american-geophysical-union"`	American Geophysical Union
`"american-institute-of-aeronautics-and-astronautics"`	American Institute of Aeronautics and Astronautics
`"american-institute-of-physics"`	AIP Style Manual 4th edition
`"american-medical-association"`	AMA Manual of Style 11th edition
`"american-meteorological-society"`	American Meteorological Society
`"american-physics-society"`	American Physical Society
`"american-physiological-society"`	American Physiological Society
`"american-political-science-association"`	APSA Style Manual revised 2018 edition
`"american-psychological-association"`	APA Style 7th edition

[A] https://citationstyles.org/

Variant	Details
`"apa"`	A short alias of `american-psychological-association`
`"american-society-for-microbiology"`	American Society for Microbiology
`"american-society-of-civil-engineers"`	American Society of Civil Engineers
`"american-society-of-mechanical-engineers"`	American Society of Mechanical Engineers
`"american-sociological-association"`	ASA Style Guide 6th/7th edition
`"angewandte-chemie"`	Angewandte Chemie International Edition
`"annual-reviews"`	Annual Reviews (sorted by order of appearance)
`"annual-reviews-author-date"`	Annual Reviews (author-date)
`"associacao-brasileira-de-normas-tecnicas"`	Associação Brasileira de Normas Técnicas (Português - Brasil)
`"association-for-computing-machinery"`	Association for Computing Machinery
`"biomed-central"`	BioMed Central
`"bristol-university-press"`	Bristol University Press
`"british-medical-journal"`	BMJ
`"bmj"`	A short alias of `british-medical-journal`
`"cell"`	Cell
`"chicago-author-date"`	Chicago Manual of Style 18th edition (author-date)
`"chicago-notes"`	Chicago Manual of Style 18th edition (notes and bibliography)
`"chicago-fullnotes"`	A short alias of `chicago-notes`
`"chicago-shortened-notes"`	Chicago Manual of Style 18th edition (shortened notes and bibliography)
`"copernicus"`	Copernicus Publications
`"cse-citation-sequence-brackets-8th-edition"`	CSE Manual: Scientific Style and Format 8th edition (citation-sequence, brackets)
`"council-of-science-editors"`	A short alias of `cse-citation-sequence-brackets-8th-edition`
`"cse-name-year"`	CSE Manual: Scientific Style and Format 9th edition (name-year)
`"council-of-science-editors-author-date"`	A short alias of `cse-name-year`
`"current-opinion"`	Current Opinion journals
`"deutsche-gesellschaft-für-psychologie"`	Deutsche Gesellschaft für Psychologie 5. Auflage (Deutsch)
`"deutsche-sprache"`	Deutsche Sprache (Deutsch)
`"elsevier-harvard"`	Elsevier (author-date/Harvard, with titles)
`"elsevier-vancouver"`	Elsevier - NLM/Vancouver (citation-sequence)
`"elsevier-with-titles"`	Elsevier (numeric, with titles)
`"frontiers"`	Frontiers journals
`"future-medicine"`	Future Medicine journals
`"future-science"`	Future Science Group
`"gb-7714-2005-numeric"`	China National Standard GB/T 7714-2005 (numeric, 中文)
`"gb-7714-2015-author-date"`	China National Standard GB/T 7714-2015 (author-date, 中文)
`"gb-7714-2015-note"`	China National Standard GB/T 7714-2015 (note, 中文)
`"gb-7714-2015-numeric"`	China National Standard GB/T 7714-2015 (numeric, 中文)

Variant	Details
`"gost-r-705-2008-numeric"`	Russian GOST R 7.0.5-2008 (numeric)
`"harvard-cite-them-right"`	Cite Them Right 12th edition (author-date/ Harvard)
`"institute-of-electrical-and-electronics-engineers"`	IEEE Reference Guide version 11.29.2023
`"ieee"`	A short alias of `institute-of-electrical-and-electronics-engineers`
`"institute-of-physics-numeric"`	Institute of Physics (numeric)
`"iso-690-author-date"`	ISO-690 (author-date, English)
`"iso-690-numeric"`	ISO-690 (numeric, English)
`"karger"`	Karger journals
`"mary-ann-liebert-vancouver"`	Mary Ann Liebert - NLM/Vancouver
`"modern-humanities-research-association-notes"`	MHRA Style Guide 4th edition (notes)
`"modern-humanities-research-association"`	A short alias of `modern-humanities-research-association-notes`
`"modern-language-association"`	MLA Handbook 9th edition (in-text citations)
`"mla"`	A short alias of `modern-language-association`
`"modern-language-association-8"`	A short alias of `modern-language-association`
`"mla-8"`	A short alias of `modern-language-association`
`"multidisciplinary-digital-publishing-institute"`	Multidisciplinary Digital Publishing Institute
`"nature"`	Nature
`"nlm-citation-sequence"`	NLM/Vancouver: Citing Medicine 2nd edition (citation-sequence)
`"vancouver"`	A short alias of `nlm-citation-sequence`
`"nlm-citation-sequence-superscript"`	NLM/Vancouver: Citing Medicine 2nd edition (citation-sequence, superscript)
`"vancouver-superscript"`	A short alias of `nlm-citation-sequence-superscript`
`"pensoft"`	Pensoft Journals
`"public-library-of-science"`	Public Library of Science
`"plos"`	A short alias of `public-library-of-science`
`"royal-society-of-chemistry"`	Royal Society of Chemistry
`"sage-vancouver"`	SAGE - NLM/Vancouver
`"sist02"`	SIST02 (日本語)
`"spie"`	SPIE journals
`"springer-basic"`	Springer - Basic (numeric, brackets)
`"springer-basic-author-date"`	Springer - Basic (author-date)
`"springer-fachzeitschriften-medizin-psychologie"`	Springer - Fachzeitschriften Medizin Psychologie (Deutsch)
`"springer-humanities-author-date"`	Springer - Humanities (author-date)
`"springer-lecture-notes-in-computer-science"`	Springer - Lecture Notes in Computer Science
`"springer-mathphys"`	Springer - MathPhys (numeric, brackets)
`"springer-socpsych-author-date"`	Springer - SocPsych (author-date)
`"springer-vancouver"`	Springer - NLM/Vancouver (citation-sequence, brackets)
`"taylor-and-francis-chicago-author-date"`	Taylor & Francis Journals Standard Reference Style Guide: Chicago author-date version 2.0

Variant	Details
`"taylor-and-francis-national-library-of-medicine"`	Taylor & Francis - NLM/Vancouver (citation-sequence, brackets)
`"the-institution-of-engineering-and-technology"`	The Institution of Engineering and Technology
`"the-lancet"`	The Lancet
`"thieme"`	Thieme-German (Deutsch)
`"trends"`	Trends journals
`"turabian-author-date"`	Chicago Manual of Style 17th edition (author-date)
`"turabian-fullnote-8"`	Chicago Manual of Style 17th edition (notes and bibliography, subsequent author-title)

3.6.5 `divider` Element

A thematic break that separates sections of content.

By default, it renders as a horizontal line, but it can be customized through show rules.

/1 Example

```
She left without a word.
#divider()
Three days later, she returned.
```

She left without a word.

Three days later, she returned.

/2 Styling

The divider can be styled through show rules.

Since the divider shows as a line by default, you can use a set rule to adjust the line's stroke:

```
#show divider: set line(stroke: 2pt + red)
First part
#divider()
Second part
```

First part

Second part

You can also fully replace the divider with custom content like a floral or asterisks, but then you should wrap it in a block to preserve spacing:

```
#show divider: set align(center)
#show divider: block[* * *]
Chapter 1
#divider()
Chapter 2
```

Chapter 1

* * *

Chapter 2

/3 Parameters

```
divider() → content
```

3.6.6 `document` Element

Manages metadata and is used to add a document file to a bundle.

/1 Metadata

The document element is the single source of truth for document metadata. With it, you can specify the document's title, authors, date, etc. in one place. Typically, the element is used with a set rule[3.2.1] like this:

```
#set document(title: [My doc])

Title is _not_ rendered, but
embedded in PDF metadata.
```

Title is *not* rendered, but embedded in PDF metadata.

By default, the metadata is embedded into the output, but not visibly rendered in the document. However, it becomes contextually available[3.4] to the full document and can be used by elements and templates. For instance, the built-in `title`[3.6.22] element automatically picks up the configured document title:

```
#set document(title: [My doc])

#title()
Title is now rendered _and_
embedded in PDF metadata.
```

My doc

Title is now rendered *and* embedded in PDF metadata.

In a similar fashion to the `title` element, you can also access metadata yourself using a context expression[3.4].

```
// In the document.
#set document(
  keywords: ("Typst", "Metadata")
)

// Somewhere in your template.
_Keywords:_
#context document.keywords.join(", ")
```

Keywords: Typst, Metadata

In single-document export formats, this function is only used with set rules[3.2.1]. Such set rules must only occur at the top level, not inside of any layout container. You can also explicitly create a `document` element, but this is only relevant in bundle export[3.6.6/2].

/1.1 Format-specific considerations

Metadata is embedded into the output to varying extents:

- PDF export supports the full range of metadata and emits it into the PDF *document information dictionary* as well as XMP metadata.
- HTML export only supports the `title`, `description`, `author`, and `keywords` properties. The `date` property is not supported as the HTML standard has no provision for it.
- SVG and PNG export do not have any metadata support at all.

/2 Documents in bundle export

In bundle export[3.18], a document element represents a single file in the bundle output, in one of Typst's other export formats. When creating a document, you must provide an output path and some content. Typst will compile and export the provided content with the appropriate format. By default, the format is inferred from the file extension of the path you specified, but you can also configure the `format`[3.6.6/3.2] explicitly.

```
#document("index.html", title: [Home])[
  #title()
```

```
  View #link(<list>)[my famous list].
]

#document("list.html", title: [My Famous List])[
  #title()
  - My
  - Famous
  - List
] <list>
```

/2.1 Metadata

Document elements pick up metadata from top-level `set document` rules within them. This means that documents written for single-document export can be used with explicit `document` elements while properly retaining metadata.

```
// Will pick up the title defined in `paper.typ`.
#document("paper.pdf", include "paper.typ")
```

```
// paper.typ
#set document(title: [My Paper])
...
```

Note that document set rules within a `document` override explicit arguments passed to the `document` element.

Moreover, properties configured as explicit arguments to `document` are made contextually available:

```
#document("index.html", title: [My title])[
  // Both of these will pick up `[My title]`
  #title()
  #context document.title
]
```

/3 Parameters

```
document(
  str,
  format: auto str,
  title: none content,
  author: str array,
  description: none content,
  keywords: str array,
  date: none auto datetime,
  content,
) → content
```

/3.1 path `str` *Required* *Positional*

The path in the bundle at which the exported document will be placed.

May contain interior slashes, in which case intermediate directories will be automatically created.

This property is only supported in the bundle[3.18] target.

/3.2 format `auto` or `str` *Settable* Default: `auto`

Which format to export in.

If `auto`, Typst attempts to infer the export format from the path's[3.6.6/3.1] file extension.

This property is only supported in the bundle[3.18] target.

Variant	Details
`"pdf"`	High-fidelity document and graphics format, with focus on exact reproduction in print.
`"png"`	Raster format for illustrations and transparent graphics.
`"svg"`	The vector graphics format of the web.
`"html"`	The document format of the web.

/3.3 **`title`** `none` or `content` *Settable* Default: `none`

The document's title. This is rendered as the title of the PDF viewer window or the browser tab of the page.

By default, the configured title is not visibly rendered in the document. You can add the title to the document's contents by using the `title`[3.6.22] element. It will automatically pick up the title configured here.

Adding a title is important for accessibility, as it makes it easier to navigate to your document and identify it among other open documents. When exporting to PDF/UA, a title is required.

While this can be arbitrary content, PDF viewers only support plain text titles, so the conversion might be lossy.

/3.4 **`author`** `str` or `array` *Settable* Default: `()`

The document's authors.

/3.5 **`description`** `none` or `content` *Settable* Default: `none`

The document's description.

/3.6 **`keywords`** `str` or `array` *Settable* Default: `()`

The document's keywords.

/3.7 **`date`** `none` or `auto` or `datetime` *Settable* Default: `auto`

The document's creation date.

If this is `auto` (default), Typst uses the current date and time. Setting it to `none` prevents Typst from embedding any creation date into the PDF metadata.

The year component must be at least zero in order to be embedded into a PDF.

If you want to create byte-by-byte reproducible PDFs, set this to something other than `auto`.

/3.8 **`body`** `content` *Required* *Positional*

The content that makes up the document.

This property is only supported in the bundle[3.18] target.

3.6.7 `emph` Element

Emphasizes content by toggling italics.

- If the current text style[3.7.12/2.3] is `"normal"`, this turns it into `"italic"`.
- If it is already `"italic"` or `"oblique"`, it turns it back to `"normal"`.

/1 Example

```
This is _emphasized._ \
This is #emph[too.]

#show emph: it => {
  text(blue, it.body)
}

This is _emphasized_ differently.
```

This is *emphasized.*
This is *too.*

This is emphasized differently.

/2 Syntax

This function also has dedicated syntax: To emphasize content, simply enclose it in underscores (_). Note that this only works at word boundaries. To emphasize part of a word, you have to use the function.

/3 Parameters

```
emph(content) → content
```

/3.1 **body** `content` *Required* *Positional*

The content to emphasize.

3.6.8 `figure` Element

A figure with an optional caption.

Automatically detects its kind to select the correct counting track. For example, figures containing images will be numbered separately from figures containing tables.

/1 Examples

The example below shows a basic figure with an image:

```
@glacier shows a glacier. Glaciers
are complex systems.

#figure(
  image("glacier.jpg", width: 80%),
  caption: [A curious figure.],
) <glacier>
```

Figure 1 shows a glacier. Glaciers are complex systems.



Figure 1: A curious figure.

You can also insert tables[3.6.20] into figures to give them a caption. The figure will detect this and automatically use a separate counter.

```
#figure(
  table(
    columns: 4,
    [t], [1], [2], [3],
    [y], [0.3s], [0.4s], [0.8s],
  ),
  caption: [Timing results],
)
```

t	1	2	3
y	0.3s	0.4s	0.8s

Table 1: Timing results

This behaviour can be overridden by explicitly specifying the figure's `kind`. All figures of the same kind share a common counter.

/2 Figure behaviour

By default, figures are placed within the flow of content. To make them float to the top or bottom of the page, you can use the `placement`[3.6.8/5.3] argument.

If your figure is too large and its contents are breakable across pages (e.g. if it contains a large table), then you can make the figure itself breakable across pages as well with this show rule:

```
#show figure: set block(breakable: true)
```

See the block[3.10.4/2.3] documentation for more information about breakable and non-breakable blocks.

/3 Caption customization

You can modify the appearance of the figure's caption with its associated `caption`[3.6.8/6.1] function. In the example below, we emphasize all captions:

```
#show figure.caption: emph

#figure(
  rect[Hello],
  caption: [I am emphasized!],
)
```

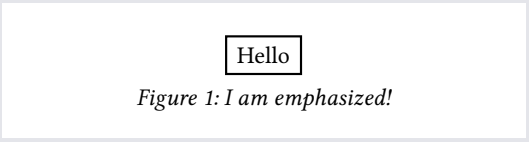


Figure 1: I am emphasized!

By using a `where`[3.5.15/8.2] selector, we can scope such rules to specific kinds of figures. For example, to position the caption above tables, but keep it below for all other kinds of figures, we could write the following show-set rule:

```
#show figure.where(
  kind: table
): set figure.caption(position: top)

#figure(
  table(columns: 2)[A][B][C][D],
  caption: [I'm up here],
)
```

Table 1: I'm up here

A	B
C	D

/4 Accessibility

You can use the `alt`[3.6.8/5.2] parameter to provide an alternative description[4.4.7] of the figure for screen readers and other Assistive Technology (AT). Refer to its documentation[3.6.8/5.2] to learn more.

You can use figures to add alternative descriptions to paths, shapes, or visualizations that do not have their own `alt` parameter. If your graphic is purely decorative and does not have a semantic meaning, consider wrapping it in `pdf.artifact`[3.14.4] instead, which will hide it from AT when exporting to PDF.

AT will always read the figure at the point where it appears in the document, regardless of its `placement`[3.6.8/5.3]. Put its markup where it would make the most sense in the reading order.

/5 Parameters

```
figure(
  content,
  alt: none str,
  placement: none auto alignment,
  scope: str,
  caption: none content,
  kind: auto str function,
  supplement: none auto content function,
  numbering: none str function,
  gap: length,
  outlined: bool,
) → content
```

/5.1 body `content` *Required* *Positional*

The content of the figure. Often, an image[3.11.7].

/5.2 alt `none` or `str` *Settable* Default: none

An alternative description of the figure.

When you add an alternative description, AT will read both it and the caption (if any). However, the content of the figure itself will be skipped.

When the body of your figure is an image[3.11.7] with its own `alt` text set, this parameter should not be used on the figure element. Likewise, do not use this parameter when the figure contains a table, code, or other

content that is already accessible. In such cases, the content of the figure will be read by AT, and adding an alternative description would lead to a loss of information.

You can learn how to write good alternative descriptions in the Accessibility Guide[4.4.7].

/5.3 **placement** `none` or `auto` or `alignment` *Settable* Default: `none`

The figure's placement on the page.

- `none`: The figure stays in-flow exactly where it was specified like other content.
- `auto`: The figure picks `top` or `bottom` depending on which is closer.
- `top`: The figure floats to the top of the page.
- `bottom`: The figure floats to the bottom of the page.

The gap between the main flow content and the floating figure is controlled by the `clearance`[3.10.19/4.4] argument on the `place` function.

```
#set page(height: 200pt)
#show figure: set place(
  clearance: 1em,
)

= Introduction
#figure(
  placement: bottom,
  caption: [A glacier],
  image("glacier.jpg", width: 60%),
)
#lorem(60)
```



/5.4 **scope** `str` *Settable* Default: `"column"`

Relative to which containing scope the figure is placed.

Set this to `"parent"` to create a full-width figure in a two-column document.

Has no effect if `placement` is `none`.

```
#set page(height: 250pt, columns: 2)

= Introduction
#figure(
  placement: bottom,
  scope: "parent",
  caption: [A glacier],
  image("glacier.jpg", width: 60%),
)
#lorem(60)
```



Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod

Figure 1: A glacier

aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At.

Variant	Details
`"column"`	Place into the current column.
`"parent"`	Place relative to the parent, letting the content span over all columns.

/5.5 **caption** `none` or `content` *Settable* Default: `none`

The figure's caption.

/5.6 **kind** `auto` or `str` or `function` *Settable* Default: `auto`

The kind of figure this is.

All figures of the same kind share a common counter.

If set to `auto`, the figure will try to automatically determine its kind based on the type of its body. Automatically detected kinds are tables[3.6.20] and code[3.7.6]. In other cases, the inferred kind is that of an image[3.11.7].

Setting this to something other than `auto` will override the automatic detection. This can be useful if
– you wish to create a custom figure type that is not an image[3.11.7], a table[3.6.20] or code[3.7.6],
– you want to force the figure to use a specific counter regardless of its content.

You can set the kind to be an element function or a string. If you set it to an element function other than `table`[3.6.20], `raw`[3.7.6], or `image`[3.11.7], you will need to manually specify the figure's supplement.

Example · Customizing the figure kind

```
#figure(
  circle(radius: 10pt),
  caption: [A curious atom.],
  kind: "atom",
  supplement: [Atom],
)
```



Atom 1: A curious atom.

If you want to modify a counter to skip a number or reset the counter, you can access the counter[3.12.1] of each kind of figure with a `where`[3.5.15/8.2] selector:

– For tables[3.6.20]: `counter(figure.where(kind: table))`
– For images[3.11.7]: `counter(figure.where(kind: image))`
– For a custom kind: `counter(figure.where(kind: kind))`

Example · Modifying the figure counter for specific kinds

```
#figure(
  table(columns: 2, $n$, $1$),
  caption: [The first table.],
)

#counter(
  figure.where(kind: table)
).update(41)

#figure(
  table(columns: 2, $n$, $42$),
  caption: [The 42nd table],
)

#figure(
  rect[Image],
  caption: [Does not affect images],
)
```

n	1

Table 1: The first table.

n	42

Table 42: The 42nd table



Figure 1: Does not affect images

To conveniently use the correct counter in a show rule, you can access the `counter` field. There is an example of this in the documentation of the `figure.caption` element's `body` field[3.6.8/6.1.3].

/5.7 **supplement** `none` or `auto` or `content` or `function` *Settable* Default: `auto`

The figure's supplement.

If set to `auto`, the figure will try to automatically determine the correct supplement based on the `kind` and the active text language[3.7.12/2.16]. If you are using a custom figure type, you will need to manually specify the supplement.

If a function is specified, it is passed the first descendant of the specified `kind` (typically, the figure's body) and should return content.

```
#figure(
  [The contents of my figure!],
  caption: [My custom figure],
  supplement: [Bar],
  kind: "foo",
)
```

The contents of my figure!
Bar 1: My custom figure

/5.8 **numbering** `none` or `str` or `function` *Settable* Default: `"1"`

How to number the figure. Accepts a numbering pattern or function[3.6.13] taking a single number.

/5.9 **gap** `length` *Settable* Default: `0.65em`

The vertical gap between the body and caption.

/5.10 **outlined** `bool` *Settable* Default: `true`

Whether the figure should appear in an `outline`[3.6.14] of figures.

/6 Definitions

/6.1 caption Element

The caption of a figure. This element can be used in set and show rules to customize the appearance of captions for all figures or figures of a specific kind.

In addition to its `position` and `body`, the `caption` also provides the figure's `kind`, `supplement`, `counter`, and `numbering` as fields. These parts can be used in `where`[3.5.15/8.2] selectors and show rules to build a completely custom caption.

```
#show figure.caption: emph

#figure(
  rect[Hello],
  caption: [A rectangle],
)
```

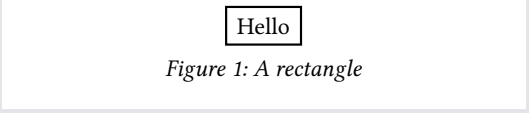


Figure 1: A rectangle

```
figure.caption(
  position: alignment,
  separator: auto content,
  content,
) → content
```

/6.1.1 position `alignment` *Settable* Default: `bottom`

The caption's position in the figure. Either `top` or `bottom`.

```
#show figure.where(
  kind: table
): set figure.caption(position: top)

#figure(
  table(columns: 2)[A][B],
  caption: [I'm up here],
)

#figure(
  rect[Hi],
  caption: [I'm down here],
)

#figure(
  table(columns: 2)[A][B],
  caption: figure.caption(
    position: bottom,
    [I'm down here too!]
  )
)
```

Table 1: I'm up here

A	B



Figure 1: I'm down here

A	B

Table 2: I'm down here too!

/6.1.2 separator `auto` or `content` *Settable* Default: `auto`

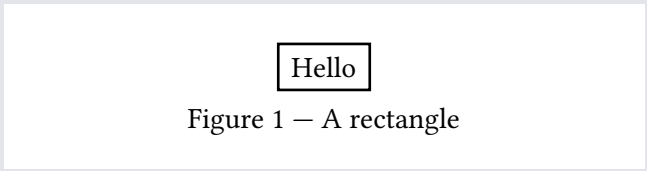
The separator which will appear between the number and body.

If set to `auto`, the separator will be adapted to the current language[3.7.12/2.16] and region[3.7.12/2.17].

```
#set figure.caption(separator: [ --- ])

#figure(
```

```
  rect[Hello],
  caption: [A rectangle],
)
```



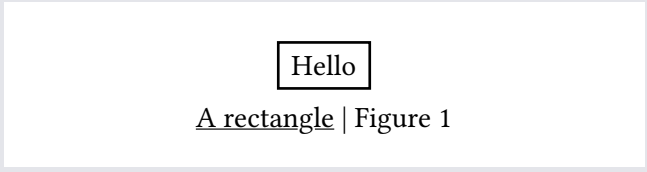
/6.1.3 **body** `content` *Required* *Positional*

The caption's body.

Can be used alongside `kind`, `supplement`, `counter`, `numbering`, and `location` to completely customize the caption.

```
#show figure.caption: it => [
  #underline(it.body) |
  #it.supplement
  #context it.counter.display(it.numbering)
]

#figure(
  rect[Hello],
  caption: [A rectangle],
)
```



3.6.9 `footnote` Element

A footnote.

Includes additional remarks and references on the same page with footnotes. A footnote will insert a superscript number that links to the note at the bottom of the page. Notes are numbered sequentially throughout your document and can break across multiple pages.

To customize the appearance of the entry in the footnote listing, see `footnote.entry`[3.6.9/4.1]. The footnote itself is realized as a normal superscript, so you can use a set rule on the `super`[3.7.11] function to customize it. You can also apply a show rule to customize only the footnote marker (superscript number) in the running text.

/1 Example

```
Check the docs for more details.
#footnote[https://typst.app/docs]
```

Check the docs for more details.[1]

[1] https://typst.app/docs

The footnote automatically attaches itself to the preceding word, even if there is a space before it in the markup. To force space, you can use the string `#" "` or explicit horizontal spacing[3.10.26].

By giving a label to a footnote, you can have multiple references to it.

```
You can edit Typst documents online.
#footnote[https://typst.app/app] <fn>
Checkout Typst's website. @fn
And the online app. #footnote(<fn>)
```

You can edit Typst documents online.[1] Checkout Typst's website.[1] And the online app.[1]

[1] https://typst.app/app

Note: Set and show rules in the scope where `footnote` is called may not apply to the footnote's content. See here[A] for more information.

/2 Accessibility

Footnotes will be read by Assistive Technology (AT) immediately after the spot in the text where they are referenced, just like how they appear in markup.

/3 Parameters

```
footnote(
  numbering: str function,
  label content,
) → content
```

/3.1 `numbering` `str` or `function` *Settable* Default: `"1"`

How to number footnotes. Accepts a numbering pattern or function[3.6.13] taking a single number.

By default, the footnote numbering continues throughout your document. If you prefer per-page footnote numbering, you can reset the footnote counter[3.12.1] in the page header[3.10.17/4.13]. In the future, there might be a simpler way to achieve this.

[A] https://github.com/typst/typst/issues/1467#issuecomment-1588799440

```
#set footnote(numbering: "*")

Footnotes:
#footnote[Star],
#footnote[Dagger]
```

Footnotes:*,†

*Star
†Dagger

/3.2 **body** `label` or `content` *Required* *Positional*

The content to put into the footnote. Can also be the label of another footnote this one should point to.

/4 Definitions

/4.1 **entry** Element

An entry in a footnote list.

This function is not intended to be called directly. Instead, it is used in set and show rules to customize footnote listings.

```
#show footnote.entry: set text(red)

My footnote listing
#footnote[It's down here]
has red text!
```

My footnote listing[1] has red text!

[1] It's down here

Note: Footnote entry properties must be uniform across each page run (a page run is a sequence of pages without an explicit pagebreak in between). For this reason, set and show rules for footnote entries should be defined before any page content, typically at the very start of the document.

```
footnote.entry(
  content,
  separator: content,
  clearance: length,
  gap: length,
  indent: length,
) → content
```

/4.1.1 **note** `content` *Required* *Positional*

The footnote for this entry. Its location can be used to determine the footnote counter state.

```
#show footnote.entry: it => {
  let loc = it.note.location()
  counter(footnote).display(at: loc, "1: ")
  it.note.body
}

Customized #footnote[Hello]
listing #footnote[World! 🌍]
```

Customized[1] listing[2]

1: Hello
2: World! 🌍

/4.1.2 **separator** `content` *Settable* Default: `line(length: 30% + 0pt, stroke: 0.05em)`

The separator between the document body and the footnote listing.

```
#set footnote.entry(
  separator: repeat[.]
)

Testing a different separator.
#footnote[
  Unconventional, but maybe
  not that bad?
]
```

Testing a different separator.[1]

[1] Unconventional, but maybe not that bad?

/4.1.3 clearance length *Settable*

Default: 1em

The amount of clearance between the document body and the separator.

```
#set footnote.entry(clearance: 3em)

Footnotes also need ...
#footnote[
  ... some space to breathe.
]
```

Footnotes also need ...[1]

[1] ... some space to breathe.

/4.1.4 gap length *Settable*

Default: 0.5em

The gap between footnote entries.

```
#set footnote.entry(gap: 0.8em)

Footnotes:
#footnote[Spaced],
#footnote[Apart]
```

Footnotes:[1],[2]

[1] Spaced
[2] Apart

/4.1.5 indent length *Settable*

Default: 1em

The indent of each footnote entry.

```
#set footnote.entry(indent: 0em)

Footnotes:
#footnote[No],
#footnote[Indent]
```

Footnotes:[1],[2]

[1] No
[2] Indent

3.6.10 `heading` Element

A section heading.

With headings, you can structure your document into sections. Each heading has a *level,* which starts at one and is unbounded upwards. This level indicates the logical role of the following content (section, subsection, etc.) A top-level heading indicates a top-level section of the document (not the document's title). To insert a title, use the `title`[3.6.22] element instead.

Typst can automatically number your headings for you. To enable numbering, specify how you want your headings to be numbered with a numbering pattern or function[3.6.13].

Independently of the numbering, Typst can also automatically generate an outline[3.6.14] of all headings for you. To exclude one or more headings from this outline, you can set the `outlined` parameter to `false`.

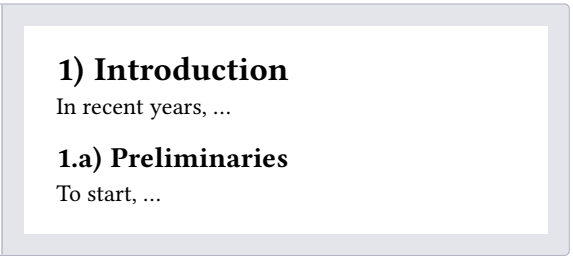
When writing a show rule[3.2.2] that accesses the `body` field[3.6.10/5.9] to create a completely custom look for headings, make sure to wrap the content in a `block`[3.10.4] (which is implicitly sticky[3.10.4/2.13] for headings through a built-in show-set rule). This prevents headings from becoming "orphans", i.e. remaining at the end of the page with the following content being on the next page.

/1 Example

```
#set heading(numbering: "1.a)")

= Introduction
In recent years, ...

== Preliminaries
To start, ...
```



/2 Syntax

Headings have dedicated syntax: They can be created by starting a line with one or multiple equals signs, followed by a space. The number of equals signs determines the heading's logical nesting depth. The `offset` field can be set to configure the starting depth.

/3 Accessibility

Headings are important for accessibility, as they help users of Assistive Technologies (AT) like screen readers to navigate within your document. Screen reader users will be able to skip from heading to heading, or get an overview of all headings in the document.

To make your headings accessible, you should not skip heading levels. This means that you should start with a first-level heading. Also, when the previous heading was of level 3, the next heading should be of level 3 (staying at the same depth), level 4 (going exactly one level deeper), or level 1 or 2 (new hierarchically higher headings).

/4 HTML export

As mentioned above, a top-level heading indicates a top-level section of the document rather than its title. This is in contrast to the HTML `<h1>` element of which there should be only one per document.

For this reason, in HTML export, a `title`[3.6.22] element will turn into an `<h1>` and headings turn into `<h2>` and lower (a level 1 heading thus turns into `<h2>`, a level 2 heading into `<h3>`, etc).

/5 Parameters

```
heading(
  level: auto int,
  depth: int,
  offset: int,
  numbering: none str function,
```

```
  supplement: none auto content function,
  outlined: bool,
  bookmarked: auto bool,
  hanging-indent: auto length,
  content,
) → content
```

/5.1 **level** `auto` or `int` *Settable* Default: `auto`

The absolute nesting depth of the heading, starting from one. If set to `auto`, it is computed from `offset + depth`.

This is primarily useful for usage in show rules[3.2.2] (either with `where`[3.5.15/8.2] selectors or by accessing the level directly on a shown heading).

```
#show heading.where(level: 2): set text(red)

= Level 1
== Level 2

#set heading(offset: 1)
= Also level 2
== Level 3
```



/5.2 **depth** `int` *Settable* Default: `1`

The relative nesting depth of the heading, starting from one. This is combined with `offset` to compute the actual `level`.

This is set by the heading syntax, such that `== Heading` creates a heading with logical depth of 2, but actual level `offset + 2`. If you construct a heading manually, you should typically prefer this over setting the absolute level.

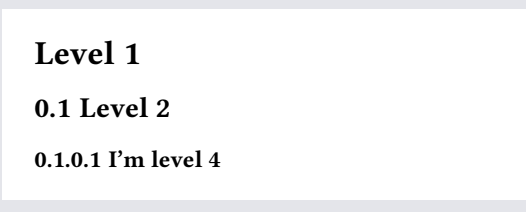
/5.3 **offset** `int` *Settable* Default: `0`

The starting offset of each heading's `level`, used to turn its relative `depth` into its absolute `level`.

```
= Level 1

#set heading(offset: 1, numbering: "1.1")
= Level 2

#heading(offset: 2, depth: 2)[
  I'm level 4
]
```

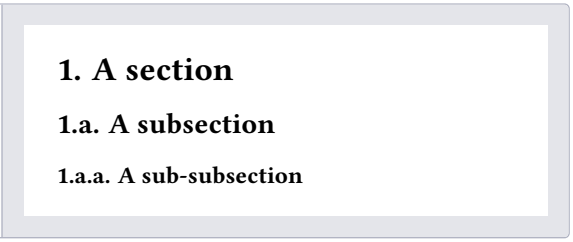


/5.4 **numbering** `none` or `str` or `function` *Settable* Default: `none`

How to number the heading. Accepts a numbering pattern or function[3.6.13] taking multiple numbers.

```
#set heading(numbering: "1.a.")

= A section
== A subsection
=== A sub-subsection
```



/5.5 **supplement** `none` or `auto` or `content` or `function` *Settable* Default: `auto`

A supplement for the heading.

For references to headings, this is added before the referenced number.

If a function is specified, it is passed the referenced heading and should return content.

```
#set heading(numbering: "1.", supplement: [Chapter])

= Introduction <intro>
In @intro, we see how to turn
Sections into Chapters. And
in @intro[Part], it is done
manually.
```



/5.6 **outlined** `bool` *Settable* Default: `true`

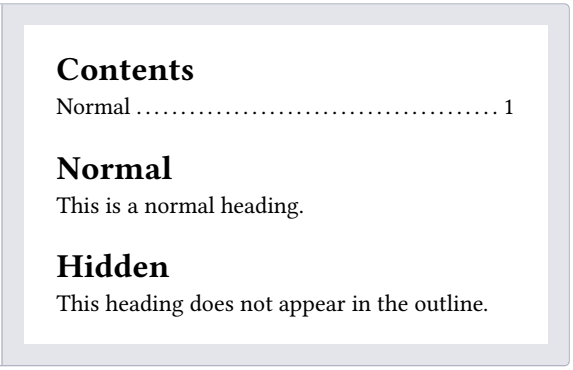
Whether the heading should appear in the outline[3.6.14].

Note that this property, if set to `true`, ensures the heading is also shown as a bookmark in the exported PDF's outline (when exporting to PDF). To change that behavior, use the `bookmarked` property.

```
#outline()

#heading[Normal]
This is a normal heading.

#heading(outlined: false)[Hidden]
This heading does not appear
in the outline.
```



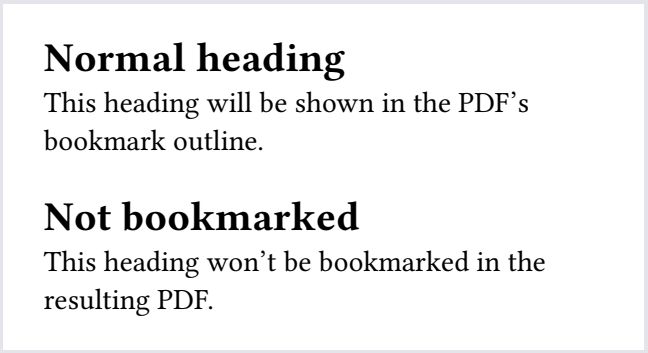
/5.7 **bookmarked** `auto` or `bool` *Settable* Default: `auto`

Whether the heading should appear as a bookmark in the exported PDF's outline. Doesn't affect other export formats, such as PNG.

The default value of `auto` indicates that the heading will only appear in the exported PDF's outline if its `outlined` property is set to `true`, that is, if it would also be listed in Typst's outline[3.6.14]. Setting this property to either `true` (bookmark) or `false` (don't bookmark) bypasses that behavior.

```
#heading[Normal heading]
This heading will be shown in
the PDF's bookmark outline.
```

```
#heading(bookmarked: false)[Not bookmarked]
This heading won't be
bookmarked in the resulting
PDF.
```



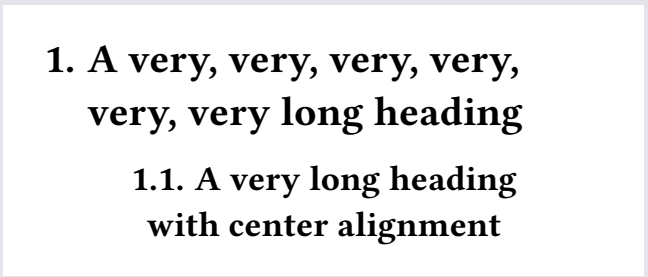
/5.8 **hanging-indent** `auto` or `length` *Settable* Default: `auto`

The indent all but the first line of a heading should have.

The default value of `auto` uses the width of the numbering as indent if the heading is aligned at the start[3.10.8/1.5] of the text direction[3.7.12/2.19], and no indent for center and other alignments.

```
#set heading(numbering: "1.")
= A very, very, very, very, very, very long heading

#show heading: set align(center)
== A very long heading\ with center alignment
```



/5.9 **body** `content` *Required* *Positional*

The heading's title.

3.6.11 `link` Element

Links to a URL or a location in the document.

By default, links do not look any different from normal text. However, you can easily apply a style of your choice with a show rule.

/1 Example

```
#show link: underline

https://example.com \

#link("https://example.com") \
#link("https://example.com")[
  See example.com
]
```

https://example.com

https://example.com
See example.com

/2 Syntax

This function also has dedicated syntax: Text that starts with `http://` or `https://` is automatically turned into a link.

/3 Hyphenation

If you enable hyphenation or justification, by default, it will not apply to links to prevent unwanted hyphenation in URLs. You can opt out of this default via `show link: set text(hyphenate: true)`.

/4 Accessibility

The destination of a link should be clear from the link text itself, or at least from the text immediately surrounding it. In PDF export, Typst will automatically generate a tooltip description for links based on their destination. For links to URLs, the URL itself will be used as the tooltip.

/5 Links in HTML export

In HTML export[3.15], a link to a label[3.5.17] or location[3.12.4] will be turned into a fragment link to a named anchor point. To support this, targets without an existing ID will automatically receive an ID in the DOM. How this works varies by which kind of HTML node(s) the link target turned into:

- If the link target turned into a single HTML element, that element will receive the ID. This is, for instance, typically the case when linking to a top-level heading (which turns into a single `<h2>` element).
- If the link target turned into a single text node, the node will be wrapped in a `<span>`, which will then receive the ID.
- If the link target turned into multiple nodes, the first node will receive the ID.
- If the link target turned into no nodes at all, an empty span will be generated to serve as a link target.

If you rely on a specific DOM structure, you should ensure that the link target turns into one or multiple elements, as the compiler makes no guarantees on the precise segmentation of text into text nodes.

If present, the automatic ID generation tries to reuse the link target's label to create a human-readable ID. A label can be reused if:

- All characters are alphabetic or numeric according to Unicode, or a hyphen, or an underscore.
- The label does not start with a digit or hyphen.

These rules ensure that the label is both a valid CSS identifier and a valid URL fragment for linking.

As IDs must be unique in the DOM, duplicate labels might need disambiguation when reusing them as IDs. The precise rules for this are as follows:

- If a label can be reused and is unique in the document, it will directly be used as the ID.

- If it's reusable, but not unique, a suffix consisting of a hyphen and an integer will be added. For instance, if the label `<mylabel>` exists twice, it would turn into `mylabel-1` and `mylabel-2`.
- Otherwise, a unique ID of the form `loc-` followed by an integer will be generated.

/6 Links in bundle export

In bundle export[3.18], linking still works as usual. For instance, if you attach a label to an element in one document, links in other documents can reference that label. In addition, documents and assets are also directly linkable. To link to a full document or asset, you can attach a label to it or query[3.12.6] for it and extract its location[3.12.4].

```
#document("index.html")[
  // Link to document.
  #link(<appendix>)[To appendix]

  // Link into document.
  See the #link(<glossary>)[Glossary]
  for more information.
]

#document("appendix.html")[
  = Definitions
  ...

  = Glossary <glossary>
  ...
] <appendix>
```

Cross-document links are emitted as relative paths (potentially with fragments). Typst automatically assigns anchor names per document based on the same rules as in HTML export. In HTML and SVG documents, these are emitted as `id` attributes on elements. In PDF documents, they are emitted as *named destinations*. PNG documents do not support linking.

Note that links always use full relative paths. In some scenarios (primarily for multi-page websites), this may not be desirable. For instance, you may want to generate a `/blog/index.html` document while wanting to link to it as just `/blog`. Furthermore, your web server might treat `/blog` and `/blog/` as interchangeable and serve `/blog/index.html` for both. If a user then navigates to `/blog`, relative links to other pages generated by Typst will no longer work. Currently, Typst does not have a way to directly hook into the built-in link handling. That said, in HTML export, depending on your use case, it may be possible to adjust the built-in link handling with a show rule on `html.elem.where(tag: "a")`.

/7 Parameters

```
link(
  str label location dictionary,
  content,
) → content
```

/7.1 dest `str` or `label` or `location` or `dictionary` *Required* *Positional*

The destination the link points to.

- To link to web pages, `dest` should be a valid URL string. If the URL is in the `mailto:` or `tel:` scheme and the `body` parameter is omitted, the email address or phone number will be the link's body, without the scheme.
- To link to another part of the document, `dest` can take one of three forms:
  - A label[3.5.17] attached to an element. If you also want automatic text for the link based on the element, consider using a reference[3.6.18] instead.
  - A `location`[3.12.4] (typically retrieved from `here`[3.12.2], `locate`[3.12.3] or `query`[3.12.6]).

– A dictionary with a page key of type integer[3.5.16] and x and y coordinates of type length[3.10.13]. Pages are counted from one, and the coordinates are relative to the page's top left corner.

```
= Introduction <intro>
#link("mailto:hello@typst.app") \
#link(<intro>)[Go to intro] \
#link((page: 1, x: 0pt, y: 0pt))[
  Go to top
]
```

Introduction
hello@typst.app
Go to intro
Go to top

/7.2 **body** `content` *Required* *Positional*

The content that should become a link.

If `dest` is an URL string, the parameter can be omitted. In this case, the URL will be shown as the link.

3.6.12 `enum` Element

A numbered list.

Displays a sequence of items vertically and numbers them consecutively.

/1 Example

```
Automatically numbered:
+ Preparations
+ Analysis
+ Conclusions

Manually numbered:
2. What is the first step?
5. I am confused.
+  Moving on ...

Multiple lines:
+ This enum item has multiple
  lines because the next line
  is indented.

Function call.
#enum[First][Second]
```

Automatically numbered:
1. Preparations
2. Analysis
3. Conclusions

Manually numbered:
2. What is the first step?
5. I am confused.
6. Moving on ...

Multiple lines:
1. This enum item has multiple lines because the next line is indented.

Function call.
1. First
2. Second

You can easily switch all your enumerations to a different numbering style with a set rule.

```
#set enum(numbering: "a)")

+ Starting off ...
+ Don't forget step two
```

a) Starting off ...
b) Don't forget step two

You can also use `enum.item`[3.6.12/4.1] to programmatically customize the number of each item in the enumeration:

```
#enum(
  enum.item(1)[First step],
  enum.item(5)[Fifth step],
  enum.item(10)[Tenth step]
)
```

1. First step
5. Fifth step
10. Tenth step

/2 Syntax

This functions also has dedicated syntax:

- Starting a line with a plus sign creates an automatically numbered enumeration item.
- Starting a line with a number followed by a dot creates an explicitly numbered enumeration item.

Enumeration items can contain multiple paragraphs and other block-level content. All content that is indented more than an item's marker becomes part of that item.

/3 Parameters

```
enum(
  tight: bool,
  numbering: str function,
  start: auto int,
```

```
  full: bool,
  reversed: bool,
  indent: length,
  body-indent: length,
  spacing: auto length,
  number-align: alignment,
  ..content array,
) → content
```

/3.1 tight bool *Settable* Default: true

Defines the default spacing[3.6.12/3.8] of the enumeration. If it is false, the items are spaced apart with paragraph spacing[3.6.15/4.2]. If it is true, they use paragraph leading[3.6.15/4.1] instead. This makes the list more compact, which can look better if the items are short.

In markup mode, the value of this parameter is determined based on whether items are separated with a blank line. If items directly follow each other, this is set to true; if items are separated by a blank line, this is set to false. The markup-defined tightness cannot be overridden with set rules.

```
+ If an enum has a lot of text, and
  maybe other inline content, it
  should not be tight anymore.

+ To make an enum wide, simply
  insert a blank line between the
  items.
```

1. If an enum has a lot of text, and maybe other inline content, it should not be tight anymore.
2. To make an enum wide, simply insert a blank line between the items.

/3.2 numbering str or function *Settable* Default: "1."

How to number the enumeration. Accepts a numbering pattern or function[3.6.13].

If the numbering pattern contains multiple counting symbols, they apply to nested enums. If given a function, the function receives one argument if `full` is false and multiple arguments if `full` is true.

```
#set enum(numbering: "1.a)")
+ Different
+ Numbering
  + Nested
  + Items
+ Style

#set enum(numbering: n => super[#n])
+ Superscript
+ Numbering!
```

1) Different
2) Numbering
   a) Nested
   b) Items
3) Style

[1] Superscript
[2] Numbering!

/3.3 start auto or int *Settable* Default: auto

Which number to start the enumeration with.

```
#enum(
  start: 3,
  [Skipping],
  [Ahead],
)
```

3. Skipping
4. Ahead

/3.4 full bool *Settable* Default: false

Whether to display the full numbering, including the numbers of all parent enumerations.

```
#set enum(numbering: "1.a)", full: true)
+ Cook
  + Heat water
  + Add ingredients
+ Eat
```

1) Cook
   1.a) Heat water
   1.b) Add ingredients
2) Eat

/3.5 **reversed** `bool` *Settable* Default: `false`

Whether to reverse the numbering for this enumeration.

```
#set enum(reversed: true)
+ Coffee
+ Tea
+ Milk
```

3. Coffee
2. Tea
1. Milk

/3.6 **indent** `length` *Settable* Default: `0pt`

The indentation of each item.

/3.7 **body-indent** `length` *Settable* Default: `0.5em`

The space between the numbering and the body of each item.

/3.8 **spacing** `auto` or `length` *Settable* Default: `auto`

The spacing between the items of the enumeration.

If set to `auto`, uses paragraph `leading`[3.6.15/4.1] for tight enumerations and paragraph `spacing`[3.6.15/4.2] for wide (non-tight) enumerations.

/3.9 **number-align** `alignment` *Settable* Default: `end`

The alignment that enum numbers should have.

By default, this is set to `end`, which aligns enum numbers horizontally towards the end of the current text direction (in a left-to-right script, for example, this is the same as `right`). In addition, the lack of a vertical alignment places each number vertically just above the baseline of the item, as if it were part of its first line of text.

The choice of `end` for horizontal alignment of enum numbers is usually preferred over `start`, as numbers then grow away from the text instead of towards it. This option lets you override this behaviour, however.

As for vertical alignment, it can be overridden if baseline alignment is not desired. For example, an alignment of `end + top` would always place the marker vertically near the top of the item, whereas `end + bottom` would move it near the bottom.

Also to note is that the unordered list[3.6.3] possesses a similar option named `marker-align` instead, which also controls both axes of marker alignment in the exact same way as `enum` numbers.

```
#set enum(number-align: start + bottom)

Here are some powers of two:
1. One
2. Two
4. Four
8. Eight
16. Sixteen
32. Thirty two
```

Here are some powers of two:
1. One
2. Two
4. Four
8. Eight
16. Sixteen
32. Thirty two

/3.10 **children** `content` or `array` *Required* *Positional* *Variadic*

The numbered list's items.

When using the enum syntax, adjacent items are automatically collected into enumerations, even through constructs like for loops.

```
#for phase in (
  "Launch",
  "Orbit",
  "Descent",
) [+ #phase]
```

1. Launch
2. Orbit
3. Descent

/4 Definitions

/4.1 item Element

An enumeration item.

```
enum.item(
  auto int,
  content,
) → content
```

/4.1.1 number auto or int *Positional* *Settable* Default: auto

The item's number.

/4.1.2 body content *Required* *Positional*

The item's body.

3.6.13 numbering

Applies a numbering to a sequence of numbers.

A numbering defines how a sequence of numbers should be displayed as content. It is defined either through a pattern string or an arbitrary function.

A numbering pattern consists of counting symbols, for which the actual number is substituted, their prefixes, and one suffix. The prefixes and the suffix are displayed as-is.

/1 Example

```
#numbering("1.1)", 1, 2, 3) \
#numbering("1.a.i", 1, 2) \
#numbering("I – 1", 12, 2) \
#numbering(
  (..nums) => nums
    .pos()
    .map(str)
    .join(".") + ")",
  1, 2, 3,
)
```

1.2.3)
1.b
XII – 2
1.2.3)

/2 Numbering patterns and numbering functions

There are multiple instances where you can provide a numbering pattern or function in Typst. For example, when defining how to number headings[3.6.10] or figures[3.6.8]. Every time, the expected format is the same as the one described below for the `numbering`[3.6.13/3.1] parameter.

The following example illustrates that a numbering function is just a regular function[3.5.15] that accepts numbers and returns `content`[3.5.8].

```
#let unary(.., last) = "|" * last
#set heading(numbering: unary)
= First heading
= Second heading
= Third heading
```

| First heading

|| Second heading

||| Third heading

/3 Parameters

```
numbering(
  str function,
  ..int,
) → any
```

/3.1 numbering `str` or `function` *Required* *Positional*

Defines how the numbering works.

Counting symbols are `1`, `a`, `A`, `i`, `I`, `α`, `Α`, 一, 壹, あ, い, ア, イ, ㄱ, 가, ㄱ, `*`, ١, ۱, १, ๑, ๑, ①, and ⓪. They are replaced by the number in the sequence, preserving the original case.

The `*` character means that symbols should be used to count, in the order of `*`, †, ‡, §, ¶, ‖. If there are more than six items, the number is represented using repeated symbols.

Suffixes are all characters after the last counting symbol. They are displayed as-is at the end of any rendered number.

Prefixes are all characters that are neither counting symbols nor suffixes. They are displayed as-is at in front of their rendered equivalent of their counting symbol.

This parameter can also be an arbitrary function that gets each number as an individual argument. When given a function, the **numbering** function just forwards the arguments to that function. While this is not particularly useful in itself, it means that you can just give arbitrary numberings to the **numbering** function without caring whether they are defined as a pattern or function.

/3.2 **numbers** `int` *Required* *Positional* *Variadic*

The numbers to apply the numbering to. Must be non-negative.

In general, numbers are counted from one. A number of zero indicates that the first element has not yet appeared.

If **numbering** is a pattern and more numbers than counting symbols are given, the last counting symbol with its prefix is repeated.

3.6.14 `outline` Element

A table of contents, figures, or other elements.

This function generates a list of all occurrences of an element in the document, up to a given depth[3.6.14/4.3]. The element's numbering and page number will be displayed in the outline alongside its title or caption.

/1 Example

```
#set heading(numbering: "1.")
#outline()

= Introduction
#lorem(5)

= Methods
== Setup
#lorem(10)
```

Contents

1. Introduction 1
2. Methods 1
   2.1. Setup 1

1. Introduction

Lorem ipsum dolor sit amet.

2. Methods

2.1. Setup

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

/2 Alternative outlines

In its default configuration, this function generates a table of contents. By setting the `target` parameter, the outline can be used to generate a list of other kinds of elements than headings.

In the example below, we list all figures containing images by setting `target` to `figure.where(kind: image)`. Just the same, we could have set it to `figure.where(kind: table)` to generate a list of tables.

We could also set it to just `figure`, without using a `where`[3.5.15/8.2] selector, but then the list would contain *all* figures, be it ones containing images, tables, or other material.

```
#outline(
  title: [List of Figures],
  target: figure.where(kind: image),
)

#figure(
  image("tiger.jpg"),
  caption: [A nice figure!],
)
```

List of Figures

Figure 1 A nice figure! 1



Figure 1: A nice figure!

/3 Styling the outline

At the most basic level, you can style the outline by setting properties on it and its entries. This way, you can customize the outline's title[3.6.14/4.1], how outline entries are indented[3.6.14/4.4], and how the space between an entry's text and its page number should be filled[3.6.14/5.1.3].

Richer customization is possible through configuration of the outline's entries[3.6.14/5.1]. The outline generates one entry for each outlined element.

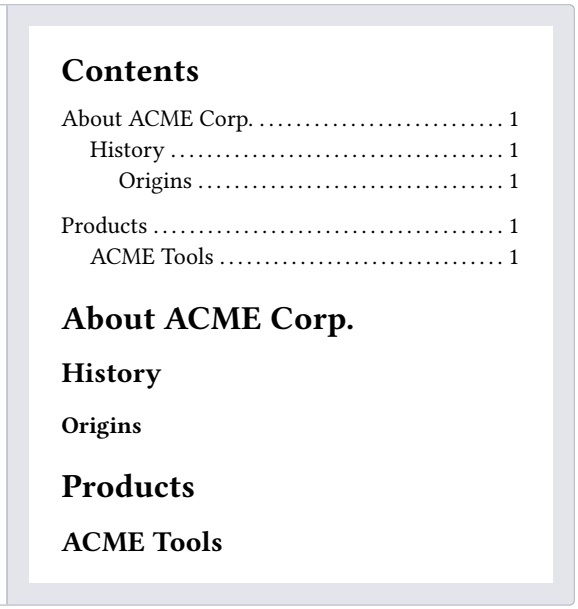
/3.1 Spacing the entries

Outline entries are blocks[3.10.4], so you can adjust the spacing between them with normal block-spacing rules:

```
#show outline.entry.where(
  level: 1
): set block(above: 1.2em)

#outline()

= About ACME Corp.
== History
=== Origins
= Products
== ACME Tools
```



/3.2 Building an outline entry from its parts

For full control, you can also write a transformational show rule on `outline.entry`. However, the logic for properly formatting and indenting outline entries is quite complex and the outline entry itself only contains two fields: The level and the outlined element.

For this reason, various helper functions are provided. You can mix and match these to compose an entry from just the parts you like.

The default show rule for an outline entry looks like this[A]:

```
#show outline.entry: it => link(
  it.element.location(),
  it.indented(it.prefix(), it.inner()),
)
```

- The `indented`[3.6.14/5.1.4.1] function takes an optional prefix and inner content and automatically applies the proper indentation to it, such that different entries align nicely and long headings wrap properly.
- The `prefix`[3.6.14/5.1.4.2] function formats the element's numbering (if any). It also appends a supplement for certain elements.
- The `inner`[3.6.14/5.1.4.3] function combines the element's `body`[3.6.14/5.1.4.4], the filler, and the page number[3.6.14/5.1.4.5].

You can use these individual functions to format the outline entry in different ways. Let's say, you'd like to fully remove the filler and page numbers. To achieve this, you could write a show rule like this:

```
#show outline.entry: it => link(
  it.element.location(),
  // Keep just the body, dropping
  // the fill and the page.
  it.indented(it.prefix(), it.body()),
)

#outline()

= About ACME Corp.
== History
```



[A] The outline of equations is the exception to this rule as it does not have a body and thus does not use indented layout.

/4 Parameters

```
outline(
  title: none auto content,
  target: label selector location function,
  depth: none int,
  indent: auto relative function,
) → content
```

/4.1 title — none or auto or content — *Settable* — Default: auto

The title of the outline.

- When set to auto, an appropriate title for the text language[3.7.12/2.16] will be used.
- When set to none, the outline will not have a title.
- A custom title can be set by passing content.

The outline's heading will not be numbered by default, but you can force it to be with a show-set rule: `show outline: set heading(numbering: "1.")`

/4.2 target — label or selector or location or function — *Settable* — Default: heading

The type of element to include in the outline.

To list figures containing a specific kind of element, like an image or a table, you can specify the desired kind in a where[3.5.15/8.2] selector. See the section on alternative outlines[3.6.14/2] for more details.

```
#outline(
  title: [List of Tables],
  target: figure.where(kind: table),
)

#figure(
  table(
    columns: 4,
    [t], [1], [2], [3],
    [y], [0.3], [0.7], [0.5],
  ),
  caption: [Experiment results],
)
```

List of Tables

Table 1 Experiment results 1

t	1	2	3
y	0.3	0.7	0.5

Table 1: Experiment results

/4.3 depth — none or int — *Settable* — Default: none

The maximum level up to which elements are included in the outline. When this argument is none, all elements are included.

```
#set heading(numbering: "1.")
#outline(depth: 2)

= Yes
Top-level section.

== Still
Subsection.

=== Nope
Not included.
```

Contents

1. Yes 1
   1.1. Still 1

1. Yes

Top-level section.

1.1. Still

Subsection.

1.1.1. Nope

Not included.

/4.4 **indent** `auto` or `relative` or `function` *Settable* Default: `auto`

How to indent the outline's entries.

- `auto`: Indents the numbering/prefix of a nested entry with the title of its parent entry. If the entries are not numbered (e.g., via heading numbering[3.6.10/5.4]), this instead simply inserts a fixed amount of `1.2em` indent per level.
- Relative length[3.10.21]: Indents the entry by the specified length per nesting level. Specifying `2em`, for instance, would indent top-level headings by `0em` (not nested), second level headings by `2em` (nested once), third-level headings by `4em` (nested twice) and so on.
- Function[3.5.15]: You can further customize this setting with a function. That function receives the nesting level as a parameter (starting at 0 for top-level headings/elements) and should return a (relative) length. For example, `n => n * 2em` would be equivalent to just specifying `2em`.

```
#set heading(numbering: "I-I.")
#set outline(title: none)

#outline()
#line(length: 100%)
#outline(indent: 3em)

= Software engineering technologies
== Requirements
== Tools and technologies
=== Code editors
== Analyzing alternatives
= Designing software components
= Testing and integration
```

I. Software engineering technologies 1
I-I. Requirements 1
I-II. Tools and technologies 1
I-II-I. Code editors 1
I-III. Analyzing alternatives 1
II. Designing software components 1
III. Testing and integration 1

I. Software engineering technologies 1
I-I. Requirements 1
I-II. Tools and technologies 1
I-II-I. Code editors 1
I-III. Analyzing alternatives 1
II. Designing software components 1
III. Testing and integration 1

/5 Definitions

/5.1 entry Element

Represents an entry line in an outline.

With show-set and show rules on outline entries, you can richly customize the outline's appearance. See the section on styling the outline[3.6.14/3] for details.

```
outline.entry(
  int,
  content,
  fill: none content,
) → content
```

/5.1.1 level `int` *Required* *Positional*

The nesting level of this outline entry. Starts at `1` for top-level entries.

/5.1.2 element `content` *Required* *Positional*

The element this entry refers to. Its location will be available through the `location`[3.5.8/3.5] method on the content and can be linked[3.6.11] to.

/5.1.3 fill `none` or `content` *Settable* Default: `repeat(body: [.], gap: 0.15em)`

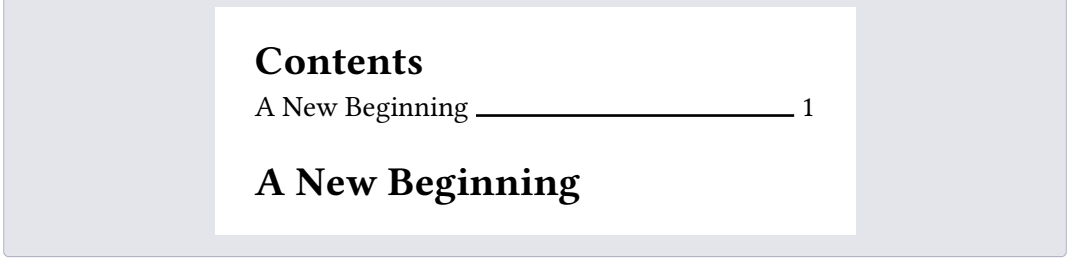
Content to fill the space between the title and the page number. Can be set to `none` to disable filling.

The `fill` will be placed into a fractionally sized box that spans the space between the entry's body and the page number. When using show rules to override outline entries, it is thus recommended to wrap the fill in a `box`[3.10.5] with fractional width, i.e. `box(width: 1fr, it.fill)`.

When using repeat[3.10.22], the gap[3.10.22/3.2] property can be useful to tweak the visual weight of the fill.

```
#set outline.entry(fill: line(length: 100%))
#outline()

= A New Beginning
```



/5.1.4 Definitions on entry

/5.1.4.1 indented Contextual

A helper function for producing an indented entry layout: Lays out a prefix and the rest of the entry in an indent-aware way.

If the parent outline's indent[3.6.14/4.4] is auto, the inner content of all entries at level N is aligned with the prefix of all entries at level N + 1, leaving at least gap space between the prefix and inner parts. Furthermore, the inner contents of all entries at the same level are aligned.

If the outline's indent is a fixed value or a function, the prefixes are indented, but the inner contents are simply offset from the prefix by the specified gap, rather than aligning outline-wide. For a visual explanation, see outline.indent[3.6.14/4.4].

```
self.indented(
  none content,
  content,
  gap: length,
) → content
```

/5.1.4.1.1 prefix none or content *Required* *Positional*

The prefix is aligned with the inner content of entries that have level one less.

In the default show rule, this is just it.prefix(), but it can be freely customized.

/5.1.4.1.2 inner content *Required* *Positional*

The formatted inner content of the entry.

In the default show rule, this is just it.inner(), but it can be freely customized.

/5.1.4.1.3 gap length Default: 0.5em

The gap between the prefix and the inner content.

/5.1.4.2 prefix Contextual

Formats the element's numbering (if any).

This also appends the element's supplement in case of figures or equations. For instance, it would output 1.1 for a heading, but Figure 1 for a figure, as is usual for outlines.

```
self.prefix() → none content
```

/5.1.4.3 inner Contextual

Creates the default inner content of the entry.

This includes the body, the fill, and page number.

```
self.inner() → content
```

/5.1.4.4 **body**

The content which is displayed in place of the referred element at its entry in the outline. For a heading, this is its body[3.6.10/5.9]; for a figure a caption and for equations, it is empty.

```
self.body() → content
```

/5.1.4.5 **page** Contextual

The page number of this entry's element, formatted with the numbering set for the referenced page.

```
self.page() → content
```

3.6.15 `par` Element

A logical subdivison of textual content.

Typst automatically collects *inline-level* elements into paragraphs. Inline-level elements include text[3.7.12], horizontal spacing[3.10.26], boxes[3.10.5], and inline equations[3.8.15].

To separate paragraphs, use a blank line (or an explicit `parbreak`[3.6.16]). Paragraphs are also automatically interrupted by any block-level element (like `block`[3.10.4], `place`[3.10.19], or anything that shows itself as one of these).

The `par` element is primarily used in set rules to affect paragraph properties, but it can also be used to explicitly display its argument as a paragraph of its own. Then, the paragraph's body may not contain any block-level content.

/1 Boxes and blocks

As explained above, usually paragraphs only contain inline-level content. However, you can integrate any kind of block-level content into a paragraph by wrapping it in a `box`[3.10.5].

Conversely, you can separate inline-level content from a paragraph by wrapping it in a `block`[3.10.4]. In this case, it will not become part of any paragraph at all. Read the following section for an explanation of why that matters and how it differs from just adding paragraph breaks around the content.

/2 What becomes a paragraph?

When you add inline-level content to your document, Typst will automatically wrap it in paragraphs. However, a typical document also contains some text that is not semantically part of a paragraph, for example in a heading or caption.

The rules for when Typst wraps inline-level content in a paragraph are as follows:

- All text at the root of a document is wrapped in paragraphs.
- Text in a container (like a `block`) is only wrapped in a paragraph if the container holds any block-level content. If all of the contents are inline-level, no paragraph is created.

In the laid-out document, it's not immediately visible whether text became part of a paragraph. However, it is still important for various reasons:

- Certain paragraph styling like `first-line-indent` will only apply to proper paragraphs, not any text. Similarly, `par` show rules of course only trigger on paragraphs.
- A proper distinction between paragraphs and other text helps people who rely on Assistive Technology (AT) (such as screen readers) navigate and understand the document properly.
- PDF export will generate a `P` tag only for paragraphs.
- HTML export will generate a `<p>` tag only for paragraphs.

When creating custom reusable components, you can and should take charge over whether Typst creates paragraphs. By wrapping text in a `block`[3.10.4] instead of just adding paragraph breaks around it, you can force the absence of a paragraph. Conversely, by adding a `parbreak`[3.6.16] after some content in a container, you can force it to become a paragraph even if it's just one word. This is, for example, what non-`tight`[3.6.3/3.1] lists do to force their items to become paragraphs.

/3 Example

```
#set par(
  first-line-indent: 1em,
  spacing: 0.65em,
  justify: true,
)

We proceed by contradiction.
Suppose that there exists a set
of positive integers $a$, $b$, and
```

We proceed by contradiction. Suppose that there exists a set of positive integers a, b, and c that satisfies the equation $a^n + b^n = c^n$ for some integer value of $n > 2$.

Without loss of generality, let a be the smallest of the three integers. Then, we ...

```
$c$ that satisfies the equation
$a^n + b^n = c^n$ for some
integer value of $n > 2$.

Without loss of generality,
let $a$ be the smallest of the
three integers. Then, we ...
```

/4 Parameters

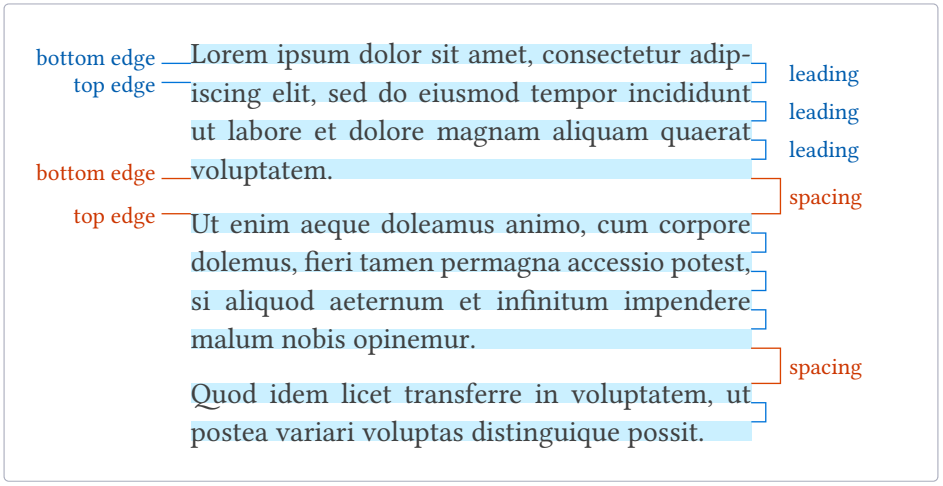
```
par(
  leading: length,
  spacing: length,
  justify: bool,
  justification-limits: dictionary,
  linebreaks: auto str,
  first-line-indent: length dictionary,
  hanging-indent: length,
  content,
) → content
```

/4.1 leading `length` *Settable* Default: `0.65em`

The spacing between lines.

Leading defines the spacing between the bottom edge[3.7.12/2.15] of one line and the top edge[3.7.12/2.14] of the following line. By default, these two properties are up to the font, but they can also be configured manually with a text set rule.

By setting top edge, bottom edge, and leading, you can also configure a consistent baseline-to-baseline distance. You could, for instance, set the leading to `1em`, the top-edge to `0.8em`, and the bottom-edge to `-0.2em` to get a baseline gap of exactly `2em`. The exact distribution of the top- and bottom-edge values affects the bounds of the first and last line.



/4.2 spacing `length` *Settable* Default: `1.2em`

The spacing between paragraphs.

Just like leading, this defines the spacing between the bottom edge of a paragraph's last line and the top edge of the next paragraph's first line. Spacing acts both above and below, collapsing to the greater of the amounts defined by adjacent paragraphs.

When a paragraph is adjacent to a `block`[3.10.4], that block's `above`[3.10.4/2.10] or `below`[3.10.4/2.11] property takes precedence over the paragraph spacing. Headings, for instance, reduce the spacing below them by default for a better look.

/4.3 **justify** `bool` *Settable* Default: `false`

Whether to justify text in its line.

Hyphenation will be enabled for justified paragraphs if the text function's `hyphenate` property[3.7.12/2.20] is set to `auto` and the current language is known.

Note that the current alignment[3.10.1/5.1] still has an effect on the placement of the last line except if it ends with a justified line break[3.7.2/3.1].

By default, Typst only changes the spacing between words to achieve justification. However, you can also allow it to adjust the spacing between individual characters using the `justification-limits` property[3.6.15/4.4].

/4.4 **justification-limits** `dictionary` *Settable*

Default:
```
(
  spacing: (min: 66.67% + 0pt, max: 150% + 0pt),
  tracking: (min: 0pt, max: 0pt),
)
```

How much the spacing between words and characters may be adjusted during justification.

When justifying text, Typst needs to stretch or shrink a line to the full width of the measure. To achieve this, by default, it adjusts the spacing between words. Additionally, it can also adjust the spacing between individual characters. This property allows you to configure lower and upper bounds for these adjustments.

The property accepts a dictionary with two entries, `spacing` and `tracking`, each containing a dictionary with the keys `min` and `max`. The `min` keys define down to which lower bound gaps may be shrunk while the `max` keys define up to which upper bound they may be stretched.

- The `spacing` entry defines how much the width of spaces between words may be adjusted. It is closely related to `text.spacing`[3.7.12/2.10] and its `min` and `max` keys accept relative lengths[3.10.21], just like the `spacing` property.

  A `min` value of `100%` means that spaces should retain their normal size (i.e. not be shrunk), while a value of `90% - 0.01em` would indicate that a space can be shrunk to a width of 90% of its normal width minus 0.01× the current font size. Similarly, a `max` value of `100% + 0.02em` means that a space's width can be increased by 0.02× the current font size. The ratio part must always be positive. The length part, meanwhile, must not be positive for `min` and not be negative for `max`.

  Note that spaces may still be expanded beyond the `max` value if there is no way to justify the line otherwise. However, other means of justification (e.g. spacing apart characters if the `tracking` entry is configured accordingly) are first used to their maximum.

- The `tracking` entry defines how much the spacing between letters may be adjusted. It is closely related to `text.tracking`[3.7.12/2.9] and its `min` and `max` keys accept lengths[3.10.13], just like the `tracking` property. Unlike `spacing`, it does not accept relative lengths because the base of the relative length would vary for each character, leading to an uneven visual appearance. The behavior compared to `spacing` is as if the base was `100%`.

  Otherwise, the `min` and `max` values work just like for `spacing`. A `max` value of `0.01em` means that additional spacing amounting to 0.01× of the current font size may be inserted between every pair of characters. Note that this also includes the gaps between spaces and characters, so for spaces the values of `tracking` act in addition to the values for `spacing`.

If you only specify one of `spacing` or `tracking`, the other retains its previously set value (or the default if it was not previously set).

If you want to enable character-level justification, a good value for the `min` and `max` keys is around `0.01em` to `0.02em` (negated for `min`). Using the same value for both gives a good baseline, but tweaking the two values individually may produce more balanced results, as demonstrated in the example below. Be careful not to set the bounds too wide, as it quickly looks unnatural.

Using character-level justification is an impactful microtypographical technique that can improve the appearance of justified text, especially in narrow columns. Note though that character-level justification does not work with every font or language. For example, cursive fonts connect letters. Using character-level justification would lead to jagged connections.

Example · Character-level justification

```
#let example(name) = columns(2, gutter: 10pt)[
  #place(top, float: true, scope: "parent", strong(name))
  /* Text from https://en.wikipedia.org/wiki/Anne_Bayley */
]

#set page(width: 440pt, height: 21em, margin: 15pt)
#set par(justify: true)
#set text(size: 0.8em)

#grid(
  columns: (1fr, 1fr),
  gutter: 20pt,
  {
    // These are Typst's default limits.
    set par(justification-limits: (
      spacing: (min: 100% * 2 / 3, max: 150%),
      tracking: (min: 0em, max: 0em),
    ))
    example[Word-level justification]
  },
  {
    // These are our custom character-level limits.
    set par(justification-limits: (
      tracking: (min: -0.01em, max: 0.02em),
    ))
    example[Character-level justification]
  },
)
```

Word-level justification

Anne Christine Bayley (1 June 1934 – 31 December 2024) was an English surgeon. She was awarded the Order of the British Empire for her research into HIV/AIDS patients in Zambia and for documenting the spread of the disease among heterosexual patients in Africa. In addition to her clinical work, she was a lecturer and head of the surgery department at the University of Zambia School of Medicine. In the 1990s, she returned to England, where she was ordained as an Anglican priest. She continued to be active in Africa throughout her retirement years.

Character-level justification

Anne Christine Bayley (1 June 1934 – 31 December 2024) was an English surgeon. She was awarded the Order of the British Empire for her research into HIV/AIDS patients in Zambia and for documenting the spread of the disease among heterosexual patients in Africa. In addition to her clinical work, she was a lecturer and head of the surgery department at the University of Zambia School of Medicine. In the 1990s, she returned to England, where she was ordained as an Anglican priest. She continued to be active in Africa throughout her retirement years.

/4.5 **linebreaks** `auto` or `str` *Settable* Default: `auto`

How to determine line breaks.

When this property is set to `auto`, its default value, optimized line breaks will be used for justified paragraphs. Enabling optimized line breaks for ragged paragraphs may also be worthwhile to improve the appearance of the text.

```
#set page(width: 207pt)
#set par(linebreaks: "simple")
Some texts feature many longer
words. Those are often exceedingly
challenging to break in a visually
pleasing way.

#set par(linebreaks: "optimized")
Some texts feature many longer
words. Those are often exceedingly
challenging to break in a visually
pleasing way.
```

Some texts feature many longer words. Those are often exceedingly challenging to break in a visually pleasing way.

Some texts feature many longer words. Those are often exceedingly challenging to break in a visually pleasing way.

Variant	Details
`"simple"`	Determine the line breaks in a simple first-fit style.
`"optimized"`	Optimize the line breaks for the whole paragraph. Typst will try to produce more evenly filled lines of text by considering the whole paragraph when calculating line breaks.

/4.6 **first-line-indent** `length` or `dictionary` *Settable* Default: `(amount: 0pt, all: false)`

The indent the first line of a paragraph should have.

By default, only the first line of a consecutive paragraph will be indented (not the first one in the document or container, and not paragraphs immediately following other block-level elements).

If you want to indent all paragraphs instead, you can pass a dictionary containing the `amount` of indent as a length and the pair `all: true`. When `all` is omitted from the dictionary, it defaults to `false`.

By typographic convention, paragraph breaks are indicated either by some space between paragraphs or by indented first lines. Consider
– reducing the paragraph spacing[3.6.15/4.2] to the leading[3.6.15/4.1] using `set par(spacing: 0.65em)`
– increasing the block spacing[3.10.4/2.9] (which inherits the paragraph spacing by default) to the original paragraph spacing using `set block(spacing: 1.2em)`

```
#set block(spacing: 1.2em)
#set par(
  first-line-indent: 1.5em,
  spacing: 0.65em,
)

The first paragraph is not affected
by the indent.

But the second paragraph is.

#line(length: 100%)

#set par(first-line-indent: (
  amount: 1.5em,
  all: true,
))

Now all paragraphs are affected
by the first line indent.

Even the first one.
```

The first paragraph is not affected by the indent.

But the second paragraph is.

Now all paragraphs are affected by the first line indent.

Even the first one.

/4.7 **hanging-indent** `length` *Settable* Default: `0pt`

The indent that all but the first line of a paragraph should have.

```
#set par(hanging-indent: 1em)

#lorem(15)
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.

/4.8 **body** `content` *Required* *Positional*

The contents of the paragraph.

/5 Definitions

/5.1 `line` Element

A paragraph line.

This element is exclusively used for line number configuration through set rules and cannot be placed.

The `numbering`[3.6.15/5.1.1] option is used to enable line numbers by specifying a numbering format.

```
#set par.line(numbering: "1")

Roses are red. \
Violets are blue. \
Typst is there for you.
```

1 Roses are red.
2 Violets are blue.
3 Typst is there for you.

The `numbering` option takes either a predefined numbering pattern[3.6.13] or a function returning styled content. You can disable line numbers for text inside certain elements by setting the numbering to `none` using show-set rules.

```
// Styled red line numbers.
#set par.line(
  numbering: n => text(red)[#n]
)

// Disable numbers inside figures.
#show figure: set par.line(
  numbering: none
)

Roses are red. \
Violets are blue.

#figure(
  caption: [Without line numbers.]
)[
  Lorem ipsum \
  dolor sit amet
]

The text above is a sample \
originating from distant times.
```

1 Roses are red.
2 Violets are blue.

Lorem ipsum
dolor sit amet
Figure 1: Without line numbers.

3 The text above is a sample
4 originating from distant times.

This element exposes further options which may be used to control other aspects of line numbering, such as its alignment[3.6.15/5.1.2] or margin[3.6.15/5.1.3]. In addition, you can control whether the numbering is reset on each page through the `numbering-scope`[3.6.15/5.1.5] option.

```
par.line(
  numbering: none str function,
  number-align: auto alignment,
  number-margin: alignment,
```

```
  number-clearance: auto length,
  numbering-scope: str,
) → content
```

/5.1.1 **numbering** none or str or function *Settable* Default: none

How to number each line. Accepts a numbering pattern or function[3.6.13] taking a single number.

```
#set par.line(numbering: "I")

Roses are red. \
Violets are blue. \
Typst is there for you.
```

I Roses are red.
II Violets are blue.
III Typst is there for you.

```
#set par.line(
  numbering: i => if calc.rem(i, 5) == 0 or i == 1 { i },
)

#lorem(60)
```

1 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam
5 quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum
10 impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri
15 amplificarique non possit. At.

/5.1.2 **number-align** auto or alignment *Settable* Default: auto

The alignment of line numbers associated with each line.

The default of auto indicates a smart default where numbers grow horizontally away from the text, considering the margin they're in and the current text direction.

```
#set par.line(
  numbering: "I",
  number-align: left,
)

Hello world! \
Today is a beautiful day \
For exploring the world.
```

I Hello world!
II Today is a beautiful day
III For exploring the world.

/5.1.3 **number-margin** alignment *Settable* Default: start

The margin at which line numbers appear.

Note: In a multi-column document, the line numbers for paragraphs inside the last column will always appear on the end margin (right margin for left-to-right text and left margin for right-to-left), regardless of this configuration. That behavior cannot be changed at this moment.

```
#set par.line(
  numbering: "1",
  number-margin: right,
)

= Report
- Brightness: Dark, yet darker
- Readings: Negative
```

Report 1
- Brightness: Dark, yet darker 2
- Readings: Negative 3

/5.1.4 number-clearance `auto` or `length` *Settable* Default: `auto`

The distance between line numbers and text.

The default value of `auto` results in a clearance that is adaptive to the page width and yields reasonable results in most cases.

```
#set par.line(
  numbering: "1",
  number-clearance: 4pt,
)

Typesetting \
Styling \
Layout
```

1 Typesetting
2 Styling
3 Layout

/5.1.5 numbering-scope `str` *Settable* Default: `"document"`

Controls when to reset line numbering.

Note: The line numbering scope must be uniform across each page run (a page run is a sequence of pages without an explicit pagebreak in between). For this reason, set rules for it should be defined before any page content, typically at the very start of the document.

```
#set par.line(
  numbering: "1",
  numbering-scope: "page",
)

First line \
Second line
#pagebreak()
First line again \
Second line again
```

1 First line
2 Second line

1 First line again
2 Second line again

Variant	Details
`"document"`	Indicates that the line number counter spans the whole document, i.e., it's never automatically reset.
`"page"`	Indicates that the line number counter should be reset at the start of every new page.

3.6.16 `parbreak` Element

A paragraph break.

This starts a new paragraph. Especially useful when used within code like for loops[3.3.5]. Multiple consecutive paragraph breaks collapse into a single one.

/1 Example

```
#for i in range(3) {
  [Blind text #i: ]
  lorem(5)
  parbreak()
}
```

Blind text 0: Lorem ipsum dolor sit amet.

Blind text 1: Lorem ipsum dolor sit amet.

Blind text 2: Lorem ipsum dolor sit amet.

/2 Syntax

Instead of calling this function, you can insert a blank line into your markup to create a paragraph break.

/3 Parameters

```
parbreak() → content
```

3.6.17 `quote` Element

Displays a quote alongside an optional attribution.

/1 Example

```
Plato is often misquoted as the author of #quote[I know that I know
nothing], however, this is a derivation form his original quote:

#set quote(block: true)

#quote(attribution: [Plato])[
  ... ἔοικα γουῦ τούτου γε σμικρῷ τινι αὐτῷ τούτῳ σοφώτερος εἶναι, ὅτι
  ἃ μὴ οἶδα οὐδὲ οἴομαι εἰδέναι.
]
#quote(attribution: [from the Henry Cary literal translation of 1897])[
  ... I seem, then, in just this little thing to be wiser than this man at
  any rate, that what I do not know I do not think I know either.
]
```

Plato is often misquoted as the author of "I know that I know nothing", however, this is a derivation form his original quote:

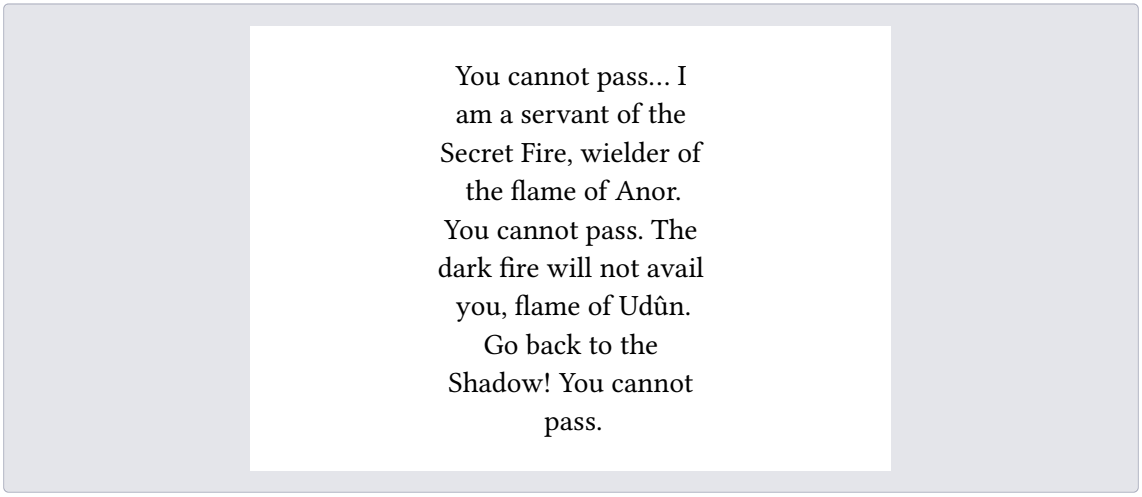
> ... ἔοικα γοῦν τούτου γε σμικρῷ τινι αὐτῷ τούτῳ σοφώτερος εἶναι, ὅτι ἃ μὴ οἶδα οὐδὲ οἴομαι εἰδέναι.
>
> — Plato

> ... I seem, then, in just this little thing to be wiser than this man at any rate, that what I do not know I do not think I know either.
>
> — from the Henry Cary literal translation of 1897

By default block quotes are padded left and right by `1em`, alignment and padding can be controlled with show rules:

```
#set quote(block: true)
#show quote: set align(center)
#show quote: set pad(x: 5em)

#quote[
  You cannot pass... I am a servant of the Secret Fire, wielder of the
  flame of Anor. You cannot pass. The dark fire will not avail you,
  flame of Udûn. Go back to the Shadow! You cannot pass.
]
```



/2 Parameters

```
quote(
  block: bool,
  quotes: auto bool,
  attribution: none label content,
  content,
) → content
```

/2.1 block `bool` *Settable* Default: `false`

Whether this is a block quote.

```
An inline citation would look like
this: #quote(
  attribution: [René Descartes]
)[
  cogito, ergo sum
], and a block equation like this:
#quote(
  block: true,
  attribution: [JFK]
)[
  Ich bin ein Berliner.
]
```

An inline citation would look like this: “cogito, ergo sum”, and a block equation like this:

> Ich bin ein Berliner.
>
> — JFK

/2.2 quotes `auto` or `bool` *Settable* Default: `auto`

Whether double quotes should be added around this quote.

The double quotes used are inferred from the `quotes` property on smartquote[3.7.8], which is affected by the `lang` property on text[3.7.12].

- `true`: Wrap this quote in double quotes.
- `false`: Do not wrap this quote in double quotes.
- `auto`: Infer whether to wrap this quote in double quotes based on the `block` property. If `block` is `false`, double quotes are automatically added.

```
#set text(lang: "de")

Ein deutsch-sprechender Author
zitiert unter umständen JFK:
#quote[Ich bin ein Berliner.]

#set text(lang: "en")

And an english speaking one may
translate the quote:
#quote[I am a Berliner.]
```

Ein deutsch-sprechender Author zitiert unter umständen JFK: „Ich bin ein Berliner.“

And an english speaking one may translate the quote: “I am a Berliner.”

/2.3 **attribution** `none` or `label` or `content` *Settable* Default: `none`

The attribution of this quote, usually the author or source. Can be a label pointing to a bibliography entry or any content. By default only displayed for block quotes, but can be changed using a `show` rule.

```
#quote(attribution: [René Descartes])[
  cogito, ergo sum
]

#show quote.where(block: false): it => {
  ["] + h(0pt, weak: true) + it.body + h(0pt, weak: true) + ["]
  if it.attribution != none [ (#it.attribution)]
}

#quote(
  attribution: link("https://typst.app/home")[typst.app]
)[
  Compose papers faster
]

#set quote(block: true)

#quote(attribution: <tolkien54>)[
  You cannot pass... I am a servant
  of the Secret Fire, wielder of the
  flame of Anor. You cannot pass. The
  dark fire will not avail you, flame
  of Udûn. Go back to the Shadow! You
  cannot pass.
]

#bibliography("works.bib", style: "apa")
```

“cogito, ergo sum”

“Compose papers faster” (typst.app)

> You cannot pass… I am a servant of the Secret Fire, wielder of the flame of Anor. You cannot pass. The dark fire will not avail you, flame of Udûn. Go back to the Shadow! You cannot pass.
>
> — Tolkien (1954)

Bibliography

Tolkien, J. R. R. (1954). *The Fellowship of the Ring* (Vol. 1, Pt. 1, Chapter 1). Allen & Unwin.

/2.4 **body** `content` *Required* *Positional*

The quote.

3.6.18 `ref` Element

A reference to a label or bibliography.

Takes a label and cross-references it. There are two kind of references, determined by its `form`[3.6.18/4.3]: `"normal"` and `"page"`.

The default, a `"normal"` reference, produces a textual reference to a label. For example, a reference to a heading will yield an appropriate string such as "Section 1" for a reference to the first heading. The word "Section" depends on the `lang`[3.7.12/2.16] setting and is localized accordingly. The references are also links to the respective element. Reference syntax can also be used to cite[3.6.4] from a bibliography.

As the default form requires a supplement and numbering, the label must be attached to a *referenceable element*. Referenceable elements include headings[3.6.10], figures[3.6.8], equations[3.8.15], and footnotes[3.6.9]. To create a custom referenceable element like a theorem, you can create a figure of a custom `kind`[3.6.8/5.6] and write a show rule for it. In the future, there might be a more direct way to define a custom referenceable element.

If you just want to link to a labelled element and not get an automatic textual reference, consider using the `link`[3.6.11] function instead.

A `"page"` reference produces a page reference to a label, displaying the page number at its location. You can use the page's supplement[3.10.17/4.11] to modify the text before the page number. Unlike a `"normal"` reference, the label can be attached to any element as long as the corresponding page's numbering[3.10.17/4.10] is set.

/1 Example

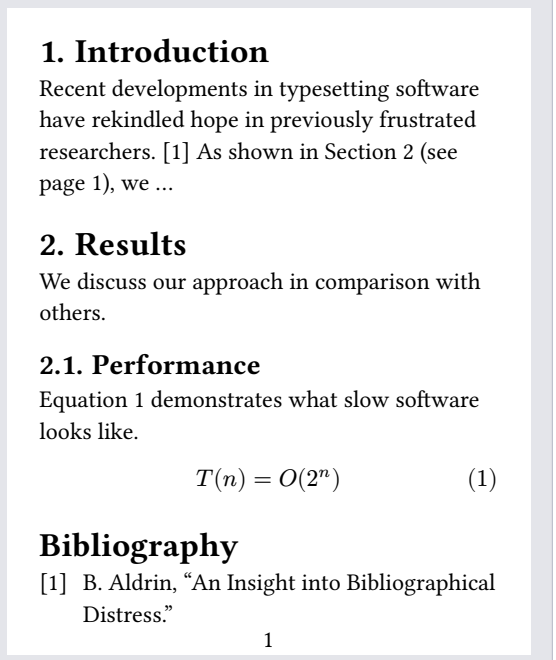
```
#set page(numbering: "1")
#set heading(numbering: "1.")
#set math.equation(numbering: "(1)")

= Introduction <intro>
Recent developments in
typesetting software have
rekindled hope in previously
frustrated researchers. @distress
As shown in @results (see
#ref(<results>, form: "page")),
we ...

= Results <results>
We discuss our approach in
comparison with others.

== Performance <perf>
@slow demonstrates what slow
software looks like.
$ T(n) = O(2^n) $ <slow>

#bibliography("works.bib")
```



/2 Syntax

This function also has dedicated syntax: A `"normal"` reference to a label can be created by typing an `@` followed by the name of the label (e.g. `= Introduction <intro>` can be referenced by typing `@intro`).

To customize the supplement, add content in square brackets after the reference: `@intro[Chapter]`.

/3 Customization

When you only ever need to reference pages of a figure/table/heading/etc. in a document, the default `form` field value can be changed to `"page"` with a set rule. If you prefer a short "p." supplement over "page", the `page.supplement`[3.10.17/4.11] field can be used for changing this:

```
#set page(
  numbering: "1",
  supplement: "p.",
)
#set ref(form: "page")

#figure(
  stack(
    dir: ltr,
    spacing: 1em,
    circle(),
    square(),
  ),
  caption: [Shapes],
) <shapes>

#pagebreak()

See @shapes for examples
of different shapes.
```



Figure 1: Shapes

1

See p. 1 for examples of different shapes.

2

If you write a show rule for references, you can access the referenced element through the `element` field of the reference. The `element` may be `none` even if it exists if Typst hasn't discovered it yet, so you always need to handle that case in your code.

```
#set heading(numbering: "1.")
#set math.equation(numbering: "(1)")

#show ref: it => {
  let eq = math.equation
  let el = it.element
  // Skip all other references.
  if el == none or el.func() != eq { return it }
  // Override equation references.
  link(el.location(), counter(eq).display(at: el.location()))
}

= Beginnings <beginning>
In @beginning we prove @pythagoras.
$ a^2 + b^2 = c^2 $ <pythagoras>
```

1. Beginnings

In Section 1 we prove (1).

$$a^2 + b^2 = c^2 \qquad (1)$$

/4 Parameters

```
ref(
  label,
  supplement: none auto content function,
  form: str,
) → content
```

/4.1 target `label` *Required* *Positional*

The target label that should be referenced.

Can be a label that is defined in the document or, if the `form`[3.6.18/4.3] is set to `"normal"`, an entry from the `bibliography`[3.6.2].

/4.2 **supplement** `none` or `auto` or `content` or `function` *Settable* Default: `auto`

A supplement for the reference.

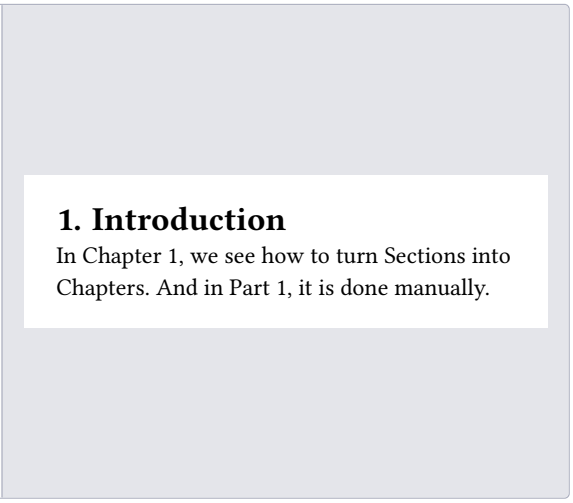
If the `form`[3.6.18/4.3] is set to `"normal"`:
– For references to headings or figures, this is added before the referenced number.
– For citations, this can be used to add a page number.

If the `form`[3.6.18/4.3] is set to `"page"`, then this is added before the page number of the label referenced.

If a function is specified, it is passed the referenced element and should return content.

```
#set heading(numbering: "1.")
#show ref.where(
  form: "normal"
): set ref(supplement: it => {
  if it.func() == heading {
    "Chapter"
  } else {
    "Thing"
  }
})

= Introduction <intro>
In @intro, we see how to turn
Sections into Chapters. And
in @intro[Part], it is done
manually.
```



/4.3 **form** `str` *Settable* Default: `"normal"`

The kind of reference to produce.

```
#set page(numbering: "1")

Here <here> we are on
#ref(<here>, form: "page").
```



Variant	Details
`"normal"`	Produces a textual reference to a label.
`"page"`	Produces a page reference to a label.

3.6.19 `strong` Element

Strongly emphasizes content by increasing the font weight.

Increases the current font weight by a given `delta`.

/1 Example

```
This is *strong.* \
This is #strong[too.] \

#show strong: set text(red)
And this is *evermore.*
```

This is **strong.**
This is **too.**

And this is **evermore.**

/2 Syntax

This function also has dedicated syntax: To strongly emphasize content, simply enclose it in stars/asterisks (`*`). Note that this only works at word boundaries. To strongly emphasize part of a word, you have to use the function.

/3 Parameters

```
strong(
  delta: int,
  content,
) → content
```

/3.1 `delta` `int` *Settable* Default: `300`

The delta to apply on the font weight.

```
#set strong(delta: 0)
No *effect!*
```

No effect!

/3.2 `body` `content` *Required* *Positional*

The content to strongly emphasize.

3.6.20 `table` Element

A table of items.

Tables are used to arrange content in cells. Cells can contain arbitrary content, including multiple paragraphs and are specified in row-major order. For a hands-on explanation of all the ways you can use and customize tables in Typst, check out the Table Guide[4.3].

Because tables are just grids with different defaults for some cell properties (notably `stroke` and `inset`), refer to the grid documentation[3.10.10/1] for more information on how to size the table tracks and specify the cell appearance properties.

If you are unsure whether you should be using a table or a grid, consider whether the content you are arranging semantically belongs together as a set of related data points or similar or whether you are just want to enhance your presentation by arranging unrelated content in a grid. In the former case, a table is the right choice, while in the latter case, a grid is more appropriate. Furthermore, Assistive Technology (AT) like screen readers will announce content in a `table` as tabular while a grid's content will be announced no different than multiple content blocks in the document flow. AT users will be able to navigate tables two-dimensionally by cell.

Note that, to override a particular cell's properties or apply show rules on table cells, you can use the `table.cell`[3.6.20/4.1] element. See its documentation for more information.

Although the `table` and the `grid` share most properties, set and show rules on one of them do not affect the other. Locating most of your styling in set and show rules is recommended, as it keeps the table's actual usages clean and easy to read. It also allows you to easily change the appearance of all tables in one place.

To give a table a caption and make it referenceable[3.6.18], put it into a figure[3.6.8].

/1 Example

The example below demonstrates some of the most common table options.

```
#table(
  columns: (1fr, auto, auto),
  inset: 10pt,
  align: horizon,
  table.header(
    [], [*Volume*], [*Parameters*],
  ),
  image("cylinder.svg"),
  $ pi h (D^2 - d^2) / 4 $,
  [
    $h$: height \
    $D$: outer radius \
    $d$: inner radius
  ],
  image("tetrahedron.svg"),
  $ sqrt(2) / 12 a^3 $,
  [$a$: edge length]
)
```

	Volume	Parameters
	$\pi h \frac{D^2 - d^2}{4}$	h: height D: outer radius d: inner radius
	$\frac{\sqrt{2}}{12} a^3$	a: edge length

Much like with grids, you can use `table.cell`[3.6.20/4.1] to customize the appearance and the position of each cell.

```
#set table(
  stroke: none,
  gutter: 0.2em,
  fill: (x, y) =>
    if x == 0 or y == 0 { gray },
  inset: (right: 1.5em),
)

#show table.cell: it => {
```

	Exam 1	Exam 2	Exam 3
John	N/A	A	N/A
Mary	N/A	A	A
Robert	B	A	B

```
  if it.x == 0 or it.y == 0 {
    set text(white)
    strong(it)
  } else if it.body == [] {
    // Replace empty cells with 'N/A'
    pad(..it.inset)[_N/A_]
  } else {
    it
  }
}

#let a = table.cell(
  fill: green.lighten(60%),
)[A]
#let b = table.cell(
  fill: aqua.lighten(60%),
)[B]

#table(
  columns: 4,
  [], [Exam 1], [Exam 2], [Exam 3],

  [John], [], a, [],
  [Mary], [], a, a,
  [Robert], b, a, b,
)
```

/2 Accessibility

Tables are challenging to consume for users of Assistive Technology (AT). To make the life of AT users easier, we strongly recommend that you use `table.header`[3.6.20/4.4] and `table.footer`[3.6.20/4.5] to mark the header and footer sections of your table. This will allow AT to announce the column labels for each cell.

Because navigating a table by cell is more cumbersome than reading it visually, you should consider making the core information in your table available as text as well. You can do this by wrapping your table in a figure[3.6.8] and using its caption to summarize the table's content.

/3 Parameters

```
table(
  columns: auto int relative fraction array,
  rows: auto int relative fraction array,
  gutter: auto int relative fraction array,
  column-gutter: auto int relative fraction array,
  row-gutter: auto int relative fraction array,
  inset: relative array dictionary function,
  align: auto array function alignment,
  fill: none color gradient tiling array function,
  stroke: none length color gradient stroke tiling array dictionary function,
  ..content,
) → content
```

/3.1 columns `auto` or `int` or `relative` or `fraction` or `array` *Settable* Default: ()

The column sizes. See the grid documentation[3.10.10/1] for more information on track sizing.

/3.2 rows `auto` or `int` or `relative` or `fraction` or `array` *Settable* Default: ()

The row sizes. See the grid documentation[3.10.10/1] for more information on track sizing.

/3.3 **gutter** auto or int or relative or fraction or array Default: ()

The gaps between rows and columns. This is a shorthand for setting column-gutter and row-gutter to the same value. See the grid documentation[3.10.10/5.3] for more information on gutters.

/3.4 **column-gutter** auto or int or relative or fraction or array *Settable* Default: ()

The gaps between columns. Takes precedence over gutter. See the grid documentation[3.10.10/5.3] for more information on gutters.

/3.5 **row-gutter** auto or int or relative or fraction or array *Settable* Default: ()

The gaps between rows. Takes precedence over gutter. See the grid documentation[3.10.10/5.3] for more information on gutters.

/3.6 **inset** relative or array or dictionary or function *Settable* Default: 0% + 5pt

How much to pad the cells' content.

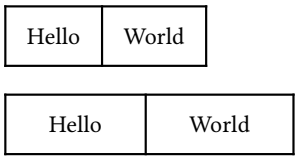
To specify the same inset for all cells, use a single length for all sides, or a dictionary of lengths for individual sides. See the box's documentation[3.10.5/2.7] for more details.

To specify a varying inset for different cells, you can:
– use a single, uniform inset for all cells
– use an array of insets for each column
– use a function that maps a cell's X/Y position (both starting from zero) to its inset

See the grid documentation[3.10.10/3] for more details.

```
#table(
  columns: 2,
  inset: 10pt,
  [Hello],
  [World],
)

#table(
  columns: 2,
  inset: (x: 20pt, y: 10pt),
  [Hello],
  [World],
)
```



/3.7 **align** auto or array or function or alignment *Settable* Default: auto

How to align the cells' content.

If set to auto, the outer alignment is used.

You can specify the alignment in any of the following fashions:
– use a single alignment for all cells
– use an array of alignments corresponding to each column
– use a function that maps a cell's X/Y position (both starting from zero) to its alignment

See the Table Guide[4.3.6] for details.

```
#table(
  columns: 3,
  align: (left, center, right),
  [Hello], [Hello], [Hello],
  [A], [B], [C],
)
```

Hello	Hello	Hello
A	B	C

/3.8 **fill** none or color or gradient or tiling or array or function *Settable* Default: none

How to fill the cells.

This can be:
– a single fill for all cells

– an array of fill corresponding to each column
– a function that maps a cell's position to its fill

Most notably, arrays and functions are useful for creating striped tables. See the Table Guide[4.3.4] for more details.

```
#table(
  fill: (x, _) =>
    if calc.odd(x) { luma(240) }
    else { white },
  align: (x, y) =>
    if y == 0 { center }
    else if x == 0 { left }
    else { right },
  columns: 4,
  [], [*Q1*], [*Q2*], [*Q3*],
  [Revenue:], [1000 €], [2000 €], [3000 €],
  [Expenses:], [500 €], [1000 €], [1500 €],
  [Profit:], [500 €], [1000 €], [1500 €],
)
```

	Q1	Q2	Q3
Revenue:	1000 €	2000 €	3000 €
Expenses:	500 €	1000 €	1500 €
Profit:	500 €	1000 €	1500 €

/3.9 **stroke** none or length or color or gradient or stroke or tiling or array or dictionary or function *Settable* Default: 1pt + black

How to stroke[3.11.12] the cells.

Strokes can be disabled by setting this to none.

If it is necessary to place lines which can cross spacing between cells produced by the gutter[3.6.20/3.3] option, or to override the stroke between multiple specific cells, consider specifying one or more of table.hline[3.6.20/4.2] and table.vline[3.6.20/4.3] alongside your table cells.

To specify the same stroke for all cells, use a single stroke[3.11.12] for all sides, or a dictionary of strokes[3.11.12] for individual sides. See the rectangle's documentation[3.11.10/2.4] for more details.

To specify varying strokes for different cells, you can:
– use a single stroke for all cells
– use an array of strokes corresponding to each column
– use a function that maps a cell's position to its stroke

See the Table Guide[4.3.5] for more details.

/3.10 **children** content *Required* *Positional* *Variadic*

The contents of the table cells, plus any extra table lines specified with the table.hline[3.6.20/4.2] and table.vline[3.6.20/4.3] elements.

/4 Definitions

/4.1 cell Element

A cell in the table. Use this to position a cell manually or to apply styling. To do the latter, you can either use the function to override the properties for a particular cell, or use it in show rules to apply certain styles to multiple cells at once.

Perhaps the most important use case of table.cell is to make a cell span multiple columns and/or rows with the colspan and rowspan fields.

```
#show table.cell.where(y: 0): strong
#set table(
  stroke: (x, y) => if y == 0 {
    (bottom: 0.7pt + black)
  },
  align: (x, y) => (
    if x > 0 { center }
    else { left }
  )
)

#table(
  columns: 3,
  table.header(
    [Substance],
    [Subcritical °C],
    [Supercritical °C],
  ),
  [Hydrochloric Acid],
  [12.0], [92.1],
  [Sodium Myreth Sulfate],
  [16.6], [104],
  [Potassium Hydroxide],
  table.cell(colspan: 2)[24.7],
)
```

Substance	Subcritical °C	Supercritical °C
Hydrochloric Acid	12.0	92.1
Sodium Myreth Sulfate	16.6	104
Potassium Hydroxide	24.7	

For example, you can override the fill, alignment or inset for a single cell:

```
// You can also import those.
#import table: cell, header

#table(
  columns: 2,
  align: center,
  header(
    [*Trip progress*],
    [*Itinerary*],
  ),
  cell(
    align: right,
    fill: fuchsia.lighten(80%),
    [🚗],
  ),
  [Get in, folks!],
  [🚗], [Eat curbside hotdog],
  cell(align: left)[🌴🚗],
  cell(
    inset: 0.06em,
    text(1.62em)[🏝️🌅🌊],
  ),
)
```

Trip progress	Itinerary
🚗	Get in, folks!
🚗	Eat curbside hotdog
🌴🚗	🏝️🌅🌊

You may also apply a show rule on `table.cell` to style all cells at once. Combined with selectors, this allows you to apply styles based on a cell's position:

```
#show table.cell.where(x: 0): strong

#table(
  columns: 3,
  gutter: 3pt,
  [Name], [Age], [Strength],
  [Hannes], [36], [Grace],
  [Irma], [50], [Resourcefulness],
  [Vikram], [49], [Perseverance],
)
```

Name	Age	Strength
Hannes	36	Grace
Irma	50	Resourcefulness
Vikram	49	Perseverance

```
table.cell(
  content,
  x: auto int,
  y: auto int,
  colspan: int,
  rowspan: int,
  inset: auto relative dictionary,
  align: auto alignment,
  fill: none auto color gradient tiling,
  stroke: none length color gradient stroke tiling dictionary,
  breakable: auto bool,
) → content
```

/4.1.1 **body** content *Required* *Positional*

The cell's body.

/4.1.2 **x** auto or int *Settable* Default: auto

The cell's column (zero-indexed). Functions identically to the x field in grid.cell[3.10.10/6.1].

/4.1.3 **y** auto or int *Settable* Default: auto

The cell's row (zero-indexed). Functions identically to the y field in grid.cell[3.10.10/6.1].

/4.1.4 **colspan** int *Settable* Default: 1

The amount of columns spanned by this cell.

/4.1.5 **rowspan** int *Settable* Default: 1

The amount of rows spanned by this cell.

/4.1.6 **inset** auto or relative or dictionary *Settable* Default: auto

The cell's inset[3.6.20/3.6] override.

/4.1.7 **align** auto or alignment *Settable* Default: auto

The cell's alignment[3.6.20/3.7] override.

/4.1.8 **fill** none or auto or color or gradient or tiling *Settable* Default: auto

The cell's fill[3.6.20/3.8] override.

/4.1.9 **stroke** none or length or color or gradient or stroke or tiling or dictionary *Settable* Default: (:)

The cell's stroke[3.6.20/3.9] override.

/4.1.10 **breakable** auto or bool *Settable* Default: auto

Whether rows spanned by this cell can be placed in different pages. When equal to auto, a cell spanning only fixed-size rows is unbreakable, while a cell spanning at least one auto-sized row is breakable.

/4.2 hline Element

A horizontal line in the table.

Overrides any per-cell stroke, including stroke specified through the table's `stroke` field. Can cross spacing between cells created through the table's `column-gutter`[3.6.20/3.4] option.

Use this function instead of the table's `stroke` field if you want to manually place a horizontal line at a specific position in a single table. Consider using table's `stroke`[3.6.20/3.9] field or `table.cell`'s `stroke`[3.6.20/4.1.9] field instead if the line you want to place is part of all your tables' designs.

```
#set table.hline(stroke: .6pt)

#table(
  stroke: none,
  columns: (auto, 1fr),
  [09:00], [Badge pick up],
  [09:45], [Opening Keynote],
  [10:30], [Talk: Typst's Future],
  [11:15], [Session: Good PRs],
  table.hline(start: 1),
  [Noon], [_Lunch break_],
  table.hline(start: 1),
  [14:00], [Talk: Tracked Layout],
  [15:00], [Talk: Automations],
  [16:00], [Workshop: Tables],
  table.hline(),
  [19:00], [Day 1 Attendee Mixer],
)
```

09:00	Badge pick up
09:45	Opening Keynote
10:30	Talk: Typst's Future
11:15	Session: Good PRs
Noon	*Lunch break*
14:00	Talk: Tracked Layout
15:00	Talk: Automations
16:00	Workshop: Tables
19:00	Day 1 Attendee Mixer

```
table.hline(
  y: auto int,
  start: int,
  end: none int,
  stroke: none length color gradient stroke tiling dictionary,
  position: alignment,
) → content
```

/4.2.1 y `auto` or `int` *Settable* Default: `auto`

The row above which the horizontal line is placed (zero-indexed). Functions identically to the y field in `grid.hline`[3.10.10/6.2.1].

/4.2.2 start `int` *Settable* Default: `0`

The column at which the horizontal line starts (zero-indexed, inclusive).

/4.2.3 end `none` or `int` *Settable* Default: `none`

The column before which the horizontal line ends (zero-indexed, exclusive).

/4.2.4 stroke `none` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable*

Default: `1pt + black`

The line's stroke.

Specifying `none` removes any lines previously placed across this line's range, including hlines or per-cell stroke below it.

/4.2.5 position `alignment` *Settable* Default: `top`

The position at which the line is placed, given its row (y) - either `top` to draw above it or `bottom` to draw below it.

This setting is only relevant when row gutter is enabled (and shouldn't be used otherwise - prefer just increasing the y field by one instead), since then the position below a row becomes different from the position above the next row due to the spacing between both.

/4.3 vline Element

A vertical line in the table. See the docs for grid.vline[3.10.10/6.3] for more information regarding how to use this element's fields.

Overrides any per-cell stroke, including stroke specified through the table's stroke field. Can cross spacing between cells created through the table's row-gutter[3.6.20/3.5] option.

Similar to table.hline[3.6.20/4.2], use this function if you want to manually place a vertical line at a specific position in a single table and use the table's stroke[3.6.20/3.9] field or table.cell's stroke[3.6.20/4.1.9] field instead if the line you want to place is part of all your tables' designs.

```
table.vline(
  x: auto int,
  start: int,
  end: none int,
  stroke: none length color gradient stroke tiling dictionary,
  position: alignment,
) → content
```

/4.3.1 x auto or int *Settable* Default: auto

The column before which the vertical line is placed (zero-indexed). Functions identically to the x field in grid.vline[3.10.10/6.3].

/4.3.2 start int *Settable* Default: 0

The row at which the vertical line starts (zero-indexed, inclusive).

/4.3.3 end none or int *Settable* Default: none

The row on top of which the vertical line ends (zero-indexed, exclusive).

/4.3.4 stroke none or length or color or gradient or stroke or tiling or dictionary *Settable*

Default: 1pt + black

The line's stroke.

Specifying none removes any lines previously placed across this line's range, including vlines or per-cell stroke below it.

/4.3.5 position alignment *Settable* Default: start

The position at which the line is placed, given its column (x) - either start to draw before it or end to draw after it.

The values left and right are also accepted, but discouraged as they cause your table to be inconsistent between left-to-right and right-to-left documents.

This setting is only relevant when column gutter is enabled (and shouldn't be used otherwise - prefer just increasing the x field by one instead), since then the position after a column becomes different from the position before the next column due to the spacing between both.

/4.4 header Element

A repeatable table header.

You should wrap your tables' heading rows in this function even if you do not plan to wrap your table across pages because Typst uses this function to attach accessibility metadata to tables and ensure Universal Access[4.4.1] to your document.

You can use the repeat parameter to control whether your table's header will be repeated across pages.

Currently, this function is unsuitable for creating a header column or single header cells. Either use regular cells, or, if you are exporting a PDF, you can also use the pdf.header-cell[3.14.7] function to mark a cell as a header cell. Likewise, you can use pdf.data-cell[3.14.6] to mark cells in this function as data cells. Note that these functions are not final and thus only available when you enable the a11y-extras feature (see the PDF module documentation[3.14] for details).

```
#set page(height: 11.5em)
#set table(
  fill: (x, y) =>
    if x == 0 or y == 0 {
      gray.lighten(40%)
    },
  align: right,
)

#show table.cell.where(x: 0): strong
#show table.cell.where(y: 0): strong

#table(
  columns: 4,
  table.header(
    [], [Blue chip],
    [Fresh IPO], [Penny st'k],
  ),
  table.cell(
    rowspan: 6,
    align: horizon,
    rotate(-90deg, reflow: true)[
      *USD / day*
    ],
  ),
  [0.20], [104], [5],
  [3.17], [108], [4],
  [1.59], [84],  [1],
  [0.26], [98],  [15],
  [0.01], [195], [4],
  [7.34], [57],  [2],
)
```

	Blue chip	Fresh IPO	Penny st'k
USD / day	0.20	104	5
	3.17	108	4
	1.59	84	1
	0.26	98	15
	0.01	195	4
	7.34	57	2

```
table.header(
  repeat: bool,
  level: int,
  ..content,
) → content
```

/4.4.1 **repeat** `bool` *Settable* Default: `true`

Whether this header should be repeated across pages.

/4.4.2 **level** `int` *Settable* Default: `1`

The level of the header. Must not be zero.

This allows repeating multiple headers at once. Headers with different levels can repeat together, as long as they have ascending levels.

Notably, when a header with a lower level starts repeating, all higher or equal level headers stop repeating (they are "replaced" by the new header).

/4.4.3 **children** `content` *Required* *Positional* *Variadic*

The cells and lines within the header.

/4.5 footer Element

A repeatable table footer.

Just like the `table.header`[3.6.20/4.4] element, the footer can repeat itself on every page of the table. This is useful for improving legibility by adding the column labels in both the header and footer of a large table, totals, or other information that should be visible on every page.

No other table cells may be placed after the footer.

```
table.footer(
  repeat: bool,
  ..content,
) → content
```

/4.5.1 **repeat** `bool` *Settable* Default: `true`

Whether this footer should be repeated across pages.

/4.5.2 **children** `content` *Required* *Positional* *Variadic*

The cells and lines within the footer.

3.6.21 `terms` Element

A list of terms and their descriptions.

Displays a sequence of terms and their descriptions vertically. When the descriptions span over multiple lines, they use hanging indent to communicate the visual hierarchy.

/1 Example

```
/ Ligature: A merged glyph.
/ Kerning: A spacing adjustment
  between two adjacent letters.
```

Ligature A merged glyph.
Kerning A spacing adjustment between two adjacent letters.

/2 Syntax

This function also has dedicated syntax: Starting a line with a slash, followed by a term, a colon and a description creates a term list item.

/3 Parameters

```
terms(
  tight: bool,
  separator: content,
  indent: length,
  hanging-indent: length,
  spacing: auto length,
  ..content array,
) → content
```

/3.1 `tight` `bool` *Settable* Default: `true`

Defines the default spacing[3.6.21/3.5] of the term list. If it is `false`, the items are spaced apart with paragraph spacing[3.6.15/4.2]. If it is `true`, they use paragraph leading[3.6.15/4.1] instead. This makes the list more compact, which can look better if the items are short.

In markup mode, the value of this parameter is determined based on whether items are separated with a blank line. If items directly follow each other, this is set to `true`; if items are separated by a blank line, this is set to `false`. The markup-defined tightness cannot be overridden with set rules.

```
/ Fact: If a term list has a lot
  of text, and maybe other inline
  content, it should not be tight
  anymore.

/ Tip: To make it wide, simply
  insert a blank line between the
  items.
```

Fact If a term list has a lot of text, and maybe other inline content, it should not be tight anymore.

Tip To make it wide, simply insert a blank line between the items.

/3.2 `separator` `content` *Settable* Default: `h(amount: 0.6em, weak: true)`

The separator between the item and the description.

If you want to just separate them with a certain amount of space, use `h(2cm, weak: true)` as the separator and replace `2cm` with your desired amount of space.

```
#set terms(separator: [: ])

/ Colon: A nice separator symbol.
```

Colon: A nice separator symbol.

/3.3 **indent** `length` *Settable* Default: `0pt`

The indentation of each item.

/3.4 **hanging-indent** `length` *Settable* Default: `2em`

The hanging indent of the description.

This is in addition to the whole item's `indent`.

```
#set terms(hanging-indent: 0pt)
/ Term: This term list does not
  make use of hanging indents.
```

Term This term list does not make use of hanging indents.

/3.5 **spacing** `auto` or `length` *Settable* Default: `auto`

The spacing between the items of the term list.

If set to `auto`, uses paragraph `leading`[3.6.15/4.1] for tight term lists and paragraph `spacing`[3.6.15/4.2] for wide (non-tight) term lists.

/3.6 **children** `content` or `array` *Required* *Positional* *Variadic*

The term list's children.

When using the term list syntax, adjacent items are automatically collected into term lists, even through constructs like for loops.

```
#for (year, product) in (
  "1978": "TeX",
  "1984": "LaTeX",
  "2019": "Typst",
) [/ #product: Born in #year.]
```

TeX Born in 1978.
LaTeX Born in 1984.
Typst Born in 2019.

/4 Definitions

/4.1 **item** Element

A term list item.

```
terms.item(
  content,
  content,
) → content
```

/4.1.1 **term** `content` *Required* *Positional*

The term described by the list item.

/4.1.2 **description** `content` *Required* *Positional*

The description of the term.

3.6.22 `title` Element

A document title.

This should be used to display the main title of the whole document and should occur only once per document. In contrast, level 1 headings[3.6.10] are intended to be used for the top-level sections of the document.

Note that additional frontmatter (like an author list) that should appear together with the title does not belong in its body.

In HTML export, this shows as a `h1` element while level 1 headings show as `h2` elements.

/1 Example

```
#set document(
  title: [Interstellar Mail Delivery]
)

#title()

= Introduction
In recent years, ...
```



/2 Parameters

```
title(auto content) → content
```

/2.1 body `auto` or `content` *Positional* *Settable* Default: `auto`

The content of the title.

When omitted (or `auto`), this will default to `document.title`[3.6.6/3.3]. In this case, a document title must have been previously set with `set document(title: [..])`.

```
#set document(title: [Course ABC, Homework 1])
#title[Homework 1]

...
```



3.7 Text

Text styling.

The text function[3.7.12] is of particular interest.

Definitions

3.7.1 Highlight Highlights text with a background color. . . . 226
3.7.2 Line Break Inserts a line break. . . . 228
3.7.3 Lorem Creates blind text. . . . 229
3.7.4 Lowercase Converts a string or content to lowercase. . . . 230
3.7.5 Overline Adds a line over text. . . . 231
3.7.6 Raw Text / Code Raw text with optional syntax highlighting. . . . 233
3.7.7 Small Capitals Displays text in small capitals. . . . 240
3.7.8 Smartquote A language-aware quote that reacts to its context. . . . 241
3.7.9 Strikethrough Strikes through text. . . . 243
3.7.10 Subscript Renders text in subscript. . . . 245
3.7.11 Superscript Renders text in superscript. . . . 246
3.7.12 Text Customizes the look and layout of text in a variety of ways. . . . 248
3.7.13 Underline Underlines text. . . . 261
3.7.14 Uppercase Converts a string or content to uppercase. . . . 263

3.7.1 `highlight` Element

Highlights text with a background color.

/1 Example

```
This is #highlight[important].
```

This is important.

/2 Parameters

```
highlight(
  fill: none color gradient tiling,
  stroke: none length color gradient stroke tiling dictionary,
  top-edge: length str,
  bottom-edge: length str,
  extent: length,
  radius: relative dictionary,
  content,
) → content
```

/2.1 `fill` `none` or `color` or `gradient` or `tiling` *Settable*

Default: `rgb("#fffd11a1")`

The color to highlight the text with.

```
This is #highlight(
  fill: blue
)[highlighted with blue].
```

This is highlighted with blue.

/2.2 `stroke` `none` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary`

Settable

Default: `(:)`

The highlight's border color. See the rectangle's documentation[3.11.10/2.4] for more details.

```
This is a #highlight(
  stroke: fuchsia
)[stroked highlighting].
```

This is a stroked highlighting.

/2.3 `top-edge` `length` or `str` *Settable*

Default: `"ascender"`

The top end of the background rectangle.

```
#set highlight(top-edge: "ascender")
#highlight[a] #highlight[aib]

#set highlight(top-edge: "x-height")
#highlight[a] #highlight[aib]
```

a aib

a aib

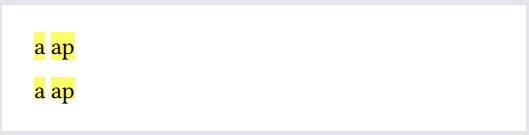
Variant	Details
`"ascender"`	The font's ascender, which typically exceeds the height of all glyphs.
`"cap-height"`	The approximate height of uppercase letters.
`"x-height"`	The approximate height of non-ascending lowercase letters.
`"baseline"`	The baseline on which the letters rest.
`"bounds"`	The top edge of the glyph's bounding box.

/2.4 **bottom-edge** `length` or `str` *Settable* Default: `"descender"`

The bottom end of the background rectangle.

```
#set highlight(bottom-edge: "descender")
#highlight[a] #highlight[ap]

#set highlight(bottom-edge: "baseline")
#highlight[a] #highlight[ap]
```

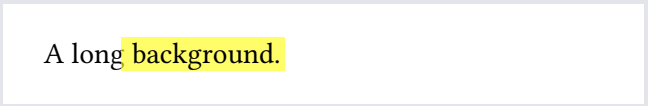


Variant	Details
`"baseline"`	The baseline on which the letters rest.
`"descender"`	The font's descender, which typically exceeds the depth of all glyphs.
`"bounds"`	The bottom edge of the glyph's bounding box.

/2.5 **extent** `length` *Settable* Default: `0pt`

The amount by which to extend the background to the sides beyond (or within if negative) the content.

```
A long #highlight(extent: 4pt)[background].
```



/2.6 **radius** `relative` or `dictionary` *Settable* Default: `(:)`

How much to round the highlight's corners. See the rectangle's documentation[3.11.10/2.5] for more details.

```
Listen #highlight(
  radius: 5pt, extent: 2pt
)[carefully], it will be on the test.
```



/2.7 **body** `content` *Required* *Positional*

The content that should be highlighted.

3.7.2 `linebreak` Element

Inserts a line break.

Advances the paragraph to the next line. A single trailing line break at the end of a paragraph is ignored, but more than one creates additional empty lines.

/1 Example

```
*Date:* 26.12.2022 \
*Topic:* Infrastructure Test \
*Severity:* High \
```

Date: 26.12.2022
Topic: Infrastructure Test
Severity: High

/2 Syntax

This function also has dedicated syntax: To insert a line break, simply write a backslash followed by whitespace. This always creates an unjustified break.

/3 Parameters

```
linebreak(justify: bool) → content
```

/3.1 `justify` `bool` *Settable* Default: `false`

Whether to justify the line before the break.

This is useful if you found a better line break opportunity in your justified text than Typst did.

```
#set par(justify: true)
#let jb = linebreak(justify: true)

I have manually tuned the #jb
line breaks in this paragraph #jb
for an _interesting_ result. #jb
```

I have manually tuned the line breaks in this paragraph for an *interesting* result.

3.7.3 `lorem`

Creates blind text.

This function yields a Latin-like *Lorem Ipsum* blind text with the given number of words. The sequence of words generated by the function is always the same but randomly chosen. As usual for blind texts, it does not make any sense. Use it as a placeholder to try layouts.

/1 Example

```
= Blind Text
#lorem(30)

= More Blind Text
#lorem(15)
```

Blind Text

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri.

More Blind Text

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.

/2 Parameters

```
lorem( int ) → str
```

/2.1 **words** `int` *Required* *Positional*

The length of the blind text in words.

3.7.4 `lower`

Converts a string or content to lowercase.

/1 Example

```
#lower("ABC") \
#lower[*My Text*] \
#lower[already low]
```

abc
my text
already low

/2 Parameters

```
lower( str content ) → str content
```

/2.1 `text` `str` or `content` *Required* *Positional*

The text to convert to lowercase.

3.7.5 `overline` Element

Adds a line over text.

/1 Example

```
#overline[A line over text.]
```



/2 Parameters

```
overline(
  stroke: auto length color gradient stroke tiling dictionary,
  offset: auto length,
  extent: length,
  evade: bool,
  background: bool,
  content,
) → content
```

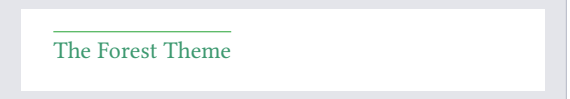
/2.1 `stroke` `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary`

Settable Default: `auto`

How to stroke[3.11.12] the line.

If set to `auto`, takes on the text's color and a thickness defined in the current font.

```
#set text(fill: olive)
#overline(
  stroke: green.darken(20%),
  offset: -12pt,
  [The Forest Theme],
)
```



/2.2 `offset` `auto` or `length` *Settable*

Default: `auto`

The position of the line relative to the baseline. Read from the font tables if `auto`.

```
#overline(offset: -1.2em)[
  The Tale Of A Faraway Line II
]
```

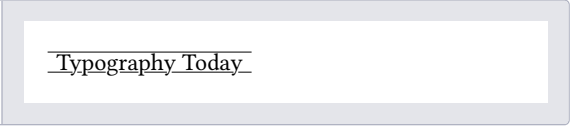


/2.3 `extent` `length` *Settable*

Default: `0pt`

The amount by which to extend the line beyond (or within if negative) the content.

```
#set overline(extent: 4pt)
#set underline(extent: 4pt)
#overline(underline[Typography Today])
```



/2.4 `evade` `bool` *Settable*

Default: `true`

Whether the line skips sections in which it would collide with the glyphs.

```
#overline(
  evade: false,
  offset: -7.5pt,
  stroke: 1pt,
```

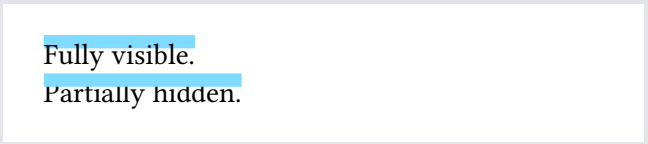


```
  extent: 3pt,
  [Temple],
)
```

/2.5 **background** `bool` *Settable* Default: `false`

Whether the line is placed behind the content it overlines.

```
#set overline(stroke: aqua + 5pt)
#overline(background: true)[Fully visible.] \
#overline(background: false)[Partially hidden.]
```



/2.6 **body** `content` *Required* *Positional*

The content to add a line over.

3.7.6 `raw` Element

Raw text with optional syntax highlighting.

Displays the text verbatim and in a monospace font. This is typically used to embed computer code into a document.

Text given to this element will ignore markup syntax, such as `*strong*` or `_emphasis_`, and will be displayed verbatim. If you would like to display content with a monospace font while still allowing markup syntax, instead of using `raw`[3.7.6], you can explicitly set the text font to a monospace font with the `text.font`[3.7.12/2.1] parameter.

Raw elements are mainly produced with their dedicated syntax[3.7.6/3] by enclosing text with either one or three-plus backtick characters (`) on both sides. When using three or more backticks, text immediately after the initial backticks will be treated as a language tag[3.7.6/4.3] used for syntax highlighting, and the raw text begins after the first whitespace.

/1 Example

```
Adding `rbx` to `rcx` gives
the desired result.

What is ```rust fn main()``` in Rust
would be ```c int main()``` in C.

```rust
fn main() {
    println!("Hello World!");
}
```

This has ``` `backticks` ``` in it
(but the spaces are trimmed). And
``` here``` the leading space is
also trimmed.
```

Adding `rbx` to `rcx` gives the desired result.

What is `fn main()` in Rust would be `int main()` in C.

```rust
fn main() {
    println!("Hello World!");
}
```

This has `` `backticks` `` in it (but the spaces are trimmed). And `here` the leading space is also trimmed.

You can also construct a `raw`[3.7.6] element programmatically from a string (and provide the language tag via the optional `lang`[3.7.6/4.3] parameter).

```
#raw("fn " + "main() {}", lang: "rust")
```

```rust
fn main() {}
```

If no syntax highlighting is available by default for your specified language tag (or if you want to override the built-in definition), you may provide a custom syntax specification file to the `syntaxes`[3.7.6/4.5] parameter.

/2 Styling

By default, the `raw` element uses the `DejaVu Sans Mono` font (included with Typst), with a smaller font size of `0.8em` (that is, 80% of the global font size). This is because monospace fonts tend to be visually larger than non-monospace fonts.

You can customize these properties with show-set rules:

```
// Switch to Cascadia Code for both
// inline and block raw.
#show raw: set text(font: "Cascadia Code")

// Reset raw blocks to the same size as normal text,
// but keep inline raw at the reduced size.
#show raw.where(block: true): set text(1em / 0.8)
```
```

````
Now using the `Cascadia Code` font for raw text.
Here's some Python code. It looks larger now:

```py
def python():
  return 5 + 5
```
````

Now using the `Cascadia Code` font for raw text. Here's some Python code. It looks larger now:

```py
def python():
  return 5 + 5
```

In addition, you can customize the syntax highlighting colors by setting a custom theme through the `theme`[3.7.6/4.6] parameter.

For complete customization of the appearance of a raw block, a show rule on `raw.line`[3.7.6/5.1] could be helpful, such as to add line numbers.

Note that in raw text, typesetting features like hyphenation[3.7.12/2.20], overhang[3.7.12/2.13], CJK-Latin spacing[3.7.12/2.11], and (for raw blocks) justification[3.6.15/4.3] will be disabled by default.

/3 Syntax

This function has dedicated syntax that produces a raw element in both markup and code mode. You can enclose text in one or three-plus backtick characters (`) on both sides to make it raw. The number of backticks must be the same on both sides, and the enclosed text cannot contain a group of that many backticks in a row. Writing just two backticks (``) produces empty raw text.

Notable differences from Markdown include that single backticks can enclose text spanning multiple lines without removing indentation, and that the three-plus backtick syntax still interprets language tags when used inline.

Raw text enclosed in *single* backticks has no way to specify a language tag and is always treated as inline for use within a paragraph, i.e. the `block`[3.7.6/4.2] parameter is `false`.

Raw syntax using *three or more* backticks has the following properties:

- **After the initial backticks, the raw block is only terminated by a sequence of the same number of backticks**

  To include text containing a sequence of backticks, the initial and final backticks must have at least one more backtick than the sequence.

- **If the raw text contains a linebreak, it will be block-level, otherwise it will be inline**

  This sets the `block`[3.7.6/4.2] parameter to `true` or `false` accordingly.

- **Text immediately after the initial backticks, up to the first whitespace, is treated as a *language tag* used for syntax highlighting**

  The specific rules for which text can be treated as the language tag are planned to change, and are explained in detail below.[3.7.6/3.2]

- **The initial and final lines have special trimming behavior**

  For the initial line, if all characters following the initial backticks or language tag are whitespace, the entire line will be trimmed. However, if there are non-whitespace characters on that line, only a single space immediately following the initial backticks or language tag will be trimmed if present.

If the final line is entirely whitespace up to the closing backticks, it will be trimmed. Otherwise, if the last non-whitespace character of the final line is a backtick, then one space character will be trimmed from the end of the line if present.

– **Common indentation at the beginning of lines is trimmed**

  Typst will remove initial whitespace at the beginning of lines in the raw text that is shared between all lines, i.e. common indentation. Although this excludes text on the line with the initial backticks.

  Typst first finds the line with the fewest initial whitespace characters that contains some non-whitespace characters, including the line with the closing backticks. Then Typst trims characters from every line equal to the number of initial whitespace characters in that line. Lines which are only whitespace will remove the same number of characters until they are empty, but will keep any extra trailing whitespace.

  Note that this check treats tabs and spaces as equivalent characters for simplicity, and that it operates on numbers of Unicode code points[A], i.e. characters, not on byte lengths.

These properties of the three-plus backtick syntax allow for some use cases that may not be obvious:

– To write text containing a sequence of backticks, enclose it with one or more backticks than the sequence: ```` ``enclosed```backticks`` ````

– To write text that starts or ends with a backtick, add a space inside the opening and closing backticks: ``` `backticks` ```

– To write inline text highlighted with a language tag, add a space between the language tag and the text ```` ```rust fn main() {}``` ````

– To write inline text without any language tag, add a space after the initial backticks: ```` ``` text``` ```` or use the single backtick syntax: `` `text` ``

/3.1 Embedding strings with raw syntax

A common use-case for raw syntax is to embed data as strings with formatting by accessing the `.text` field on raw content to get the underlying string. This may also be paired with the bytes[3.5.6] constructor to convert the string to bytes.

````
An inline YAML dictionary via `.text`

#yaml(bytes(
  ```yaml
  Magic:
    limited-by: Mana
  Pokémon:
    limited-by: Energy
  Yu-Gi-Oh:
    limited-by: false
  ```.text
  //  ^^^^ used as a string
))
````

An inline YAML dictionary via .text

```
(
  Magic: (limited-by: "Mana"),
  Pokémon: (limited-by: "Energy"),
  Yu-Gi-Oh: (limited-by: false),
)
```

/3.2 Language tag changes

When using raw syntax with three or more backticks, text immediately after the initial backticks (up to the first whitespace) is treated as a language tag[3.7.6/4.3]. However in the current version of Typst, only text that would be a valid Typst identifier is treated as the language tag. The first character not valid for an identifier will be interpreted as starting the raw text.

For example, in the current verion of Typst, if a raw block starts with `C++`, the identifier `C` will be the language tag, and the raw text will start with `++`. If a raw block starts with `++C`, it will have no language tag and the raw text will start with `++C`.

To use language tags that are not valid as identifiers in the current version of Typst, you must use the lang[3.7.6/4.3] parameter, either by calling the constructor with a string: `#raw("text", lang: "...")`, or by writing a set rule: `#set raw(lang: "...")`.

[A] https://www.unicode.org/glossary/#code_point

In the next version of Typst, *all text* up to the first whitespace or backtick will be treated as the language tag, allowing a wider character set for language tags. Tags including spaces or backticks will still need to be set manually via the `lang`[3.7.6/4.3] parameter.

Typst will alert you if your raw blocks will be interpreted differently in the next Typst version by emitting a warning.

/4 Parameters

```
raw(
  str,
  block: bool,
  lang: none str,
  align: alignment,
  syntaxes: str path bytes array,
  theme: none auto str path bytes,
  tab-size: int,
) → content
```

/4.1 text str *Required* *Positional*

The raw text.

You can also use raw blocks creatively to create custom syntaxes for your automations.

Example · Implementing a DSL using raw and show rules

```
// Parse numbers in raw blocks with the
// `mydsl` tag and sum them up.
#show raw.where(lang: "mydsl"): it => {
  let sum = 0
  for part in it.text.split("+") {
    sum += int(part.trim())
  }
  sum
}

```mydsl
1 + 2 + 3 + 4 + 5
```
```

15

/4.2 block bool *Settable* Default: false

Whether the raw text is displayed as a separate block.

In markup mode, using one-backtick notation makes this `false`. Using three-backtick notation makes it `true` if the enclosed content contains at least one line break.

```
// Display inline code in a small box
// that retains the correct baseline.
#show raw.where(block: false): box.with(
  fill: luma(240),
  inset: (x: 3pt, y: 0pt),
  outset: (y: 3pt),
  radius: 2pt,
)

// Display block code in a larger block
// with more padding.
#show raw.where(block: true): block.with(
  fill: luma(240),
  inset: 10pt,
  radius: 4pt,
```

```
)

With `rg`, you can search through your files quickly.
This example searches the current directory recursively
for the text `Hello World`:

```bash
rg "Hello World"
```
```

With `rg`, you can search through your files quickly. This example searches the current directory recursively for the text `Hello World`:

```bash
rg "Hello World"
```

/4.3 **lang** `none` or `str` *Settable* Default: `none`

The language to interpret the raw text as for syntax highlighting.

In HTML export[3.15], this sets the `data-lang` attribute of the generated `html.code`[3.15.5/2.19] element.

Apart from typical language tags known from Markdown, this supports the `"typ"`, `"typc"`, and `"typm"` tags for Typst markup[3.1.2], Typst code[3.1.4], and Typst math[3.1.3], respectively.

```
```typ
This is *Typst!*
```

This is ```typ also *Typst*```, but inline!
```

This is ***Typst!***

This is `also` ***Typst***, but inline!

/4.4 **align** `alignment` *Settable* Default: `start`

The horizontal alignment that each line in a raw block should have. This option is ignored if this is not a raw block (if specified `block: false` or single backticks were used in markup mode).

By default, this is set to `start`, meaning that raw text is aligned towards the start of the text direction inside the block by default, regardless of the current context's alignment (allowing you to center the raw block itself without centering the text inside it, for example).

```
#set raw(align: center)

```typc
let f(x) = x
code = "centered"
```
```

```
  let f(x) = x
code = "centered"
```

/4.5 **syntaxes** `str` or `path` or `bytes` or `array` *Settable* Default: `()`

Additional syntax definitions to load. The syntax definitions should be in the sublime-syntax file format[A].

You can pass any of the following values:

- A path string or path[3.5.21] to load a syntax file from.
- Raw bytes from which the syntax should be decoded.
- An array where each item is one of the above.

```
#set raw(syntaxes: "SExpressions.sublime-syntax")

```sexp
(defun factorial (x)
  (if (zerop x)
    ; with a comment
    1
    (* x (factorial (- x 1)))))
```
```

```
(defun factorial (x)
  (if (zerop x)
    ; with a comment
    1
    (* x (factorial (- x 1)))))
```

/4.6 **theme** `none` or `auto` or `str` or `path` or `bytes` *Settable* Default: `auto`

The theme to use for syntax highlighting. Themes should be in the tmTheme file format[B].

You can pass any of the following values:

- `none`: Disables syntax highlighting.
- `auto`: Highlights with Typst's default theme.
- A path string or path[3.5.21] to load a theme file from.
- Raw bytes from which the theme should be decoded.

Applying a theme only affects the color of specifically highlighted text. It does not consider the theme's foreground and background properties, so that you retain control over the color of raw text. You can apply the foreground color yourself with the text[3.7.12] function and the background with a filled block[3.10.4/2.4]. You could also use the xml[3.13.7] function to extract these properties from the theme.

```
#set raw(theme: "halcyon.tmTheme")
#show raw: it => block(
  fill: rgb("#1d2433"),
  inset: 8pt,
  radius: 5pt,
  text(fill: rgb("#a2aabc"), it)
)

```typ
= Chapter 1
#let hi = "Hello World"
```
```

```
= Chapter 1
#let hi = "Hello World"
```

/4.7 **tab-size** `int` *Settable* Default: 2

The size for a tab stop in spaces. A tab is replaced with enough spaces to align with the next multiple of the size.

[A] https://www.sublimetext.com/docs/syntax.html
[B] https://www.sublimetext.com/docs/color_schemes_tmtheme.html

```
#set raw(tab-size: 8)
```tsv
Year  Month Day
2000  2 3
2001  2 1
2002  3 10
```
```

Year	Month	Day
2000	2	3
2001	2	1
2002	3	10

/5 Definitions

/5.1 line Element

A highlighted line of raw text.

This is a helper element that is synthesized by raw[3.7.6] elements.

It allows you to access various properties of the line, such as the line number, the raw non-highlighted text, the highlighted text, and whether it is the first or last line of the raw block.

```
raw.line(
  int,
  int,
  str,
  content,
) → content
```

/5.1.1 number int *Required* *Positional*

The line number of the raw line inside of the raw block, starts at 1.

/5.1.2 count int *Required* *Positional*

The total number of lines in the raw block.

/5.1.3 text str *Required* *Positional*

The line of raw text.

/5.1.4 body content *Required* *Positional*

The highlighted raw text.

3.7.7 `smallcaps` Element

Displays text in small capitals.

/1 Example

```
Hello \
#smallcaps[Hello]
```

Hello
HELLO

/2 Smallcaps fonts

By default, this uses the `smcp` and `c2sc` OpenType features on the font. Not all fonts support these features. Sometimes, smallcaps are part of a dedicated font. This is, for example, the case for the *Latin Modern* family of fonts. In those cases, you can use a show-set rule to customize the appearance of the text in smallcaps:

```
#show smallcaps: set text(font: "Latin Modern Roman Caps")
```

In the future, this function will support synthesizing smallcaps from normal letters, but this is not yet implemented.

/3 Smallcaps headings

You can use a show rule[3.2.2] to apply smallcaps formatting to all your headings. In the example below, we also center-align our headings and disable the standard bold font.

```
#set par(justify: true)
#set heading(numbering: "I.")

#show heading: smallcaps
#show heading: set align(center)
#show heading: set text(
  weight: "regular"
)

= Introduction
#lorem(40)
```

I. INTRODUCTION

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere.

/4 Parameters

```
smallcaps(
  all: bool,
  content,
) → content
```

/4.1 `all` `bool` *Settable* Default: `false`

Whether to turn uppercase letters into small capitals as well.

Unless overridden by a show rule, this enables the `c2sc` OpenType feature.

```
#smallcaps(all: true)[UNICEF] is an
agency of #smallcaps(all: true)[UN].
```

UNICEF is an agency of UN.

/4.2 `body` `content` *Required* *Positional*

The content to display in small capitals.

3.7.8 `smartquote` Element

A language-aware quote that reacts to its context.

Automatically turns into an appropriate opening or closing quote based on the active text language[3.7.12/2.16].

/1 Example

```
"This is in quotes."

#set text(lang: "de")
"Das ist in Anführungszeichen."

#set text(lang: "fr")
"C'est entre guillemets."
```

> “This is in quotes.”
>
> „Das ist in Anführungszeichen.“
>
> « C’est entre guillemets. »

/2 Syntax

This function also has dedicated syntax: The normal quote characters (' and "). Typst automatically makes your quotes smart.

/3 Parameters

```
smartquote(
  double: bool,
  enabled: bool,
  alternative: bool,
  quotes: auto str array dictionary,
) → content
```

/3.1 `double` `bool` *Settable* Default: `true`

Whether this should be a double quote.

/3.2 `enabled` `bool` *Settable* Default: `true`

Whether smart quotes are enabled.

To disable smartness for a single quote, you can also escape it with a backslash.

```
#set smartquote(enabled: false)

These are "dumb" quotes.
```

> These are "dumb" quotes.

/3.3 `alternative` `bool` *Settable* Default: `false`

Whether to use alternative quotes.

Does nothing for languages that don’t have alternative quotes, or if explicit quotes were set.

```
#set text(lang: "de")
#set smartquote(alternative: true)

"Das ist in anderen Anführungszeichen."
```

> »Das ist in anderen Anführungszeichen.«

/3.4 `quotes` `auto` or `str` or `array` or `dictionary` *Settable* Default: `auto`

The quotes to use.

- When set to `auto`, the appropriate single quotes for the text language[3.7.12/2.16] will be used. This is the default.
- Custom quotes can be passed as a string, array, or dictionary of either

– string[3.5.27]: a string consisting of two characters containing the opening and closing double quotes (characters here refer to Unicode grapheme clusters)
– array[3.5.2]: an array containing the opening and closing double quotes
– dictionary[3.5.11]: a dictionary containing the double and single quotes, each specified as either `auto`, string, or array

```
#set text(lang: "de")
'Das sind normale Anführungszeichen.'

#set smartquote(quotes: "()")
"Das sind eigene Anführungszeichen."

#set smartquote(quotes: (single: ("[[", "]]"),  double: auto))
'Das sind eigene Anführungszeichen.'
```

‚Das sind normale Anführungszeichen.‘

(Das sind eigene Anführungszeichen.)

[[Das sind eigene Anführungszeichen.]]

3.7.9 `strike` Element

Strikes through text.

/1 Example

```
This is #strike[not] relevant.
```

This is ~~not~~ relevant.

/2 Parameters

```
strike(
  stroke: auto length color gradient stroke tiling dictionary,
  offset: auto length,
  extent: length,
  background: bool,
  content,
) → content
```

/2.1 `stroke` `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary`

Settable Default: `auto`

How to stroke[3.11.12] the line.

If set to `auto`, takes on the text's color and a thickness defined in the current font.

Note: Please don't use this for real redaction as you can still copy paste the text.

```
This is #strike(stroke: 1.5pt + red)[very stricken through]. \
This is #strike(stroke: 10pt)[redacted].
```

This is ~~very stricken through~~.
This is [redacted].

/2.2 `offset` `auto` or `length` *Settable*

Default: `auto`

The position of the line relative to the baseline. Read from the font tables if `auto`.

This is useful if you are unhappy with the offset your font provides.

```
#set text(font: "Inria Serif")
This is #strike(offset: auto)[low-ish]. \
This is #strike(offset: -3.5pt)[on-top].
```

This is ~~low-ish~~.
This is ~~on-top~~.

/2.3 `extent` `length` *Settable*

Default: `0pt`

The amount by which to extend the line beyond (or within if negative) the content.

```
This #strike(extent: -2pt)[skips] parts of the word.
This #strike(extent: 2pt)[extends] beyond the word.
```

This ~~skips~~ parts of the word. This ~~extends~~ beyond the word.

/2.4 background `bool` *Settable* Default: `false`

Whether the line is placed behind the content.

```
#set strike(stroke: red + 2pt)
#strike(background: true)[This is behind.] \
#strike(background: false)[This is in front.]
```

~~This is behind.~~
~~This is in front.~~

/2.5 body `content` *Required* *Positional*

The content to strike through.

3.7.10 sub Element

Renders text in subscript.

The text is rendered smaller and its baseline is lowered.

/1 Example

```
Revenue#sub[yearly]
```

Revenue$_{yearly}$

/2 Parameters

```
sub(
  typographic: bool,
  baseline: auto length,
  size: auto length,
  content,
) → content
```

/2.1 typographic bool *Settable* Default: true

Whether to use subscript glyphs from the font if available.

Ideally, subscripts glyphs are provided by the font (using the `subs` OpenType feature). Otherwise, Typst is able to synthesize subscripts by lowering and scaling down regular glyphs.

When this is set to `false`, synthesized glyphs will be used regardless of whether the font provides dedicated subscript glyphs. When `true`, synthesized glyphs may still be used in case the font does not provide the necessary subscript glyphs.

```
N#sub(typographic: true)[1]
N#sub(typographic: false)[1]
```

N_1 N_1

/2.2 baseline auto or length *Settable* Default: auto

The downward baseline shift for synthesized subscripts.

This only applies to synthesized subscripts. In other words, this has no effect if `typographic` is `true` and the font provides the necessary subscript glyphs.

If set to `auto`, the baseline is shifted according to the metrics provided by the font, with a fallback to `0.2em` in case the font does not define the necessary metrics.

When using multiple fonts, it might be necessary to set `baseline` and `size`[3.7.10/2.3] explicitly. See `super`[3.7.11/2.2] for an example.

/2.3 size auto or length *Settable* Default: auto

The font size for synthesized subscripts.

This only applies to synthesized subscripts. In other words, this has no effect if `typographic` is `true` and the font provides the necessary subscript glyphs.

If set to `auto`, the size is scaled according to the metrics provided by the font, with a fallback to `0.6em` in case the font does not define the necessary metrics.

/2.4 body content *Required* *Positional*

The text to display in subscript.

3.7.11 `super` Element

Renders text in superscript.

The text is rendered smaller and its baseline is raised.

/1 Example

```
1#super[st] try!
```

1st try!

/2 Parameters

```
super(
  typographic: bool,
  baseline: auto length,
  size: auto length,
  content,
) → content
```

/2.1 `typographic` `bool` *Settable* Default: `true`

Whether to use superscript glyphs from the font if available.

Ideally, superscripts glyphs are provided by the font (using the `sups` OpenType feature). Otherwise, Typst is able to synthesize superscripts by raising and scaling down regular glyphs.

When this is set to `false`, synthesized glyphs will be used regardless of whether the font provides dedicated superscript glyphs. When `true`, synthesized glyphs may still be used in case the font does not provide the necessary superscript glyphs.

```
N#super(typographic: true)[1]
N#super(typographic: false)[1]
```

N1 N1

/2.2 `baseline` `auto` or `length` *Settable* Default: `auto`

The downward baseline shift for synthesized superscripts.

This only applies to synthesized superscripts. In other words, this has no effect if `typographic` is `true` and the font provides the necessary superscript glyphs.

If set to `auto`, the baseline is shifted according to the metrics provided by the font, with a fallback to `-0.5em` in case the font does not define the necessary metrics.

Note that, since the baseline shift is applied downward, you will need to provide a negative value for the content to appear as raised above the normal baseline.

Sometimes it is necessary to set `baseline` and `size`[3.7.11/2.3] explicitly. In the following example, the super-scripted text uses multiple fonts with incompatible metrics. To avoid misalignment, we override the metrics for all fonts.

```
#let tz(timezone) = text(
  font: "Roboto",
  smallcaps(timezone, all: true),
)

/ Per-font metrics:
  14:00#super[#tz[UTC], not #tz[CET]]

#set super(
  baseline: -0.4em,
```

Per-font metrics 14:00UTC, not CET

Unified metrics 14:00UTC, not CET

```
  size: 0.6em,
  typographic: false,
)
/ Unified metrics:
  14:00#super[#tz[UTC], not #tz[CET]]
```

/2.3 **size** `auto` or `length` *Settable* Default: `auto`

The font size for synthesized superscripts.

This only applies to synthesized superscripts. In other words, this has no effect if `typographic` is `true` and the font provides the necessary superscript glyphs.

If set to `auto`, the size is scaled according to the metrics provided by the font, with a fallback to `0.6em` in case the font does not define the necessary metrics.

/2.4 **body** `content` *Required* *Positional*

The text to display in superscript.

3.7.12 `text` Element

Customizes the look and layout of text in a variety of ways.

This function is used frequently, both with set rules and directly. While the set rule is often the simpler choice, calling the `text` function directly can be useful when passing text as an argument to another function.

/1 Example

```
#set text(18pt)
With a set rule.

#emph(text(blue)[
  With a function call.
])
```

With a set rule.

With a function call.

/2 Parameters

```
text(
  font: str array dictionary,
  fallback: bool,
  style: str,
  weight: int str,
  stretch: ratio,
  size: length,
  fill: color gradient tiling,
  stroke: none length color gradient stroke tiling dictionary,
  tracking: length,
  spacing: relative,
  cjk-latin-spacing: none auto,
  baseline: length,
  overhang: bool,
  top-edge: length str,
  bottom-edge: length str,
  lang: str,
  region: none str,
  script: auto str,
  dir: auto direction,
  hyphenate: auto bool,
  costs: dictionary,
  kerning: bool,
  alternates: bool int,
  stylistic-set: none int array,
  ligatures: bool,
  discretionary-ligatures: bool,
  historical-ligatures: bool,
  number-type: auto str,
  number-width: auto str,
  slashed-zero: bool,
  fractions: bool,
  features: array dictionary,
  variations: dictionary,
  body: content,
```

```
  str,
) → content
```

/2.1 **font** `str` or `array` or `dictionary` *Settable* Default: `"libertinus serif"`

A font family descriptor or priority list of font family descriptors.

A font family descriptor can be a plain string representing the family name or a dictionary with the following keys:

- `name` (required): The font family name.
- `covers` (optional): Defines the Unicode codepoints for which the family shall be used. This can be:
  - A predefined coverage set:
    - `"latin-in-cjk"` covers all codepoints except for those which exist in Latin fonts, but should preferably be taken from CJK fonts.
  - A regular expression[3.5.23] that defines exactly which codepoints shall be covered. Accepts only the subset of regular expressions which consist of exactly one dot, letter, or character class.

When processing text, Typst tries all specified font families in order until it finds a font that has the necessary glyphs. In the example below, the font `Inria Serif` is preferred, but since it does not contain Arabic glyphs, the arabic text uses `Noto Sans Arabic` instead.

Typst aims to unify different fonts from the same family under a single family name. To that effect, it automatically trims common style suffixes like "Bold" or "Condensed" from font family names. Instead of selecting these through the name, access them through Typst's built-in mechanisms (such as the `weight`[3.7.12/2.4] and `stretch`[3.7.12/2.5] parameters). Similarly, when using a variable font with Typst, the suffixes "Variable", "Var", and "VF" should be omitted as Typst trims them to unify static and variable fonts into a single family.

Between fonts from the same family, Typst picks the one that is the closest match to the configured text `style`[3.7.12/2.3], `weight`[3.7.12/2.4], and `stretch`[3.7.12/2.5]. If both a static and a variable font support a specific configuration, the variable font is preferred.

The collection of available fonts differs by platform:

- In the web app, you can see the list of available fonts by clicking on the "Ag" button. You can provide additional fonts by uploading `.ttf` or `.otf` files into your project. They will be discovered automatically. The priority is: project fonts > server fonts.
- Locally, Typst uses your installed system fonts or embedded fonts in the CLI, which are `Libertinus Serif`, `New Computer Modern`, `New Computer Modern Math`, and `DejaVu Sans Mono`. In addition, you can use the `--font-path` argument or `TYPST_FONT_PATHS` environment variable to add directories that should be scanned for fonts. The priority is: `--font-path` > system fonts > embedded fonts. Run `typst fonts` to see the fonts that Typst has discovered on your system. Note that you can pass the `--ignore-system-fonts` parameter to the CLI to ensure Typst won't search for system fonts.

```
#set text(font: "PT Sans")
This is sans-serif.

#set text(font: (
  "Inria Serif",
  "Noto Sans Arabic",
))

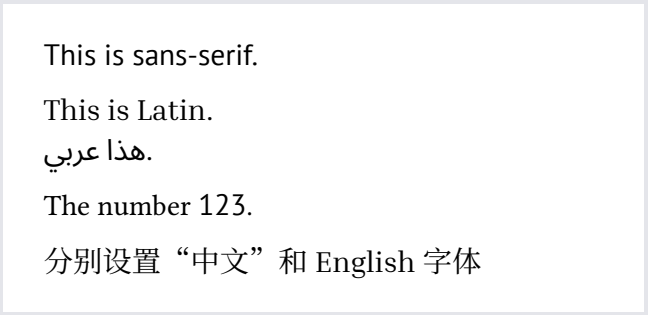
This is Latin. \
هذا عربي.

// Change font only for numbers.
#set text(font: (
  (name: "PT Sans", covers: regex("[0-9]")),
  "Libertinus Serif"
))

The number 123.

// Mix Latin and CJK fonts.
```

```
#set text(font: (
  (name: "Inria Serif", covers: "latin-in-cjk"),
  "Noto Serif CJK SC"
))
分别设置“中文”和English字体
```



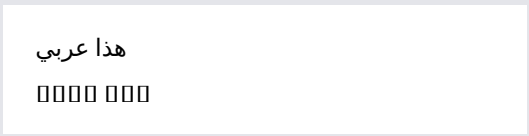
/2.2 **fallback** `bool` *Settable* Default: `true`

Whether to allow last resort font fallback when the primary font list contains no match. This lets Typst search through all available fonts for the most similar one that has the necessary glyphs.

Note: Currently, there are no warnings when fallback is disabled and no glyphs are found. Instead, your text shows up in the form of “tofus”: Small boxes that indicate the lack of an appropriate glyph. In the future, you will be able to instruct Typst to issue warnings so you know something is up.

```
#set text(font: "Inria Serif")
هذا عربي

#set text(fallback: false)
هذا عربي
```



/2.3 **style** `str` *Settable* Default: `"normal"`

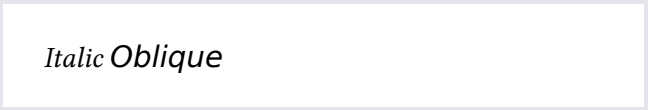
The desired font style.

When an italic style is requested and only an oblique one is available, it is used. Similarly, the other way around, an italic style can stand in for an oblique one. When neither an italic nor an oblique style is available, Typst selects the normal style. Since most fonts are only available either in an italic or oblique style, the difference between italic and oblique style is rarely observable.

When used with a suitable variable font, Typst will automatically configure the `ital` (for an italic style) or `slnt` (for an oblique style) font variation[3.7.12/2.33] based on this property.

Note: If you want to emphasize your text, you should do so using the emph[3.6.7] function instead. This makes it easy to adapt the style later if you change your mind about how to signify the emphasis.

```
#text(font: "Libertinus Serif", style: "italic")[Italic]
#text(font: "DejaVu Sans", style: "oblique")[Oblique]
```



Variant	Details
`"normal"`	The default, typically upright style.
`"italic"`	A cursive style with custom letterform.
`"oblique"`	Just a slanted version of the normal style.

/2.4 **weight** `int` or `str` *Settable* Default: `"regular"`

The desired thickness of the font's glyphs. Accepts an integer between `100` and `900` or one of the predefined weight names. When the desired weight is not available, Typst selects the font from the family that is closest in weight.

When used with a suitable variable font, Typst will automatically configure the `wght` font variation[3.7.12/2.33] based on this property.

Note: If you want to strongly emphasize your text, you should do so using the strong[3.6.19] function instead. This makes it easy to adapt the style later if you change your mind about how to signify the strong emphasis.

```
#set text(font: "IBM Plex Sans")

#text(weight: "light")[Light] \
#text(weight: "regular")[Regular] \
#text(weight: "medium")[Medium] \
#text(weight: 500)[Medium] \
#text(weight: "bold")[Bold]
```

Light
Regular
Medium
Medium
Bold

Variant	Details
`"thin"`	Thin weight (100).
`"extralight"`	Extra light weight (200).
`"light"`	Light weight (300).
`"regular"`	Regular weight (400).
`"medium"`	Medium weight (500).
`"semibold"`	Semibold weight (600).
`"bold"`	Bold weight (700).
`"extrabold"`	Extrabold weight (800).
`"black"`	Black weight (900).

/2.5 **stretch** `ratio` *Settable* Default: `100%`

The desired width of the glyphs. Accepts a ratio between `50%` and `200%`. When the desired width is not available, Typst selects the font from the family that is closest in stretch. This will only stretch the text if a condensed or expanded version of the font is available.

When used with a suitable variable font, Typst will automatically configure the `wdth` font variation[3.7.12/2.33] based on this property.

If you want to adjust the amount of space between characters instead of stretching the glyphs itself, use the `tracking`[3.7.12/2.9] property instead.

```
#set text(font: "IBM Plex Sans")
#text(stretch: 75%)[Condensed] \
#text(stretch: 100%)[Normal]
```

Condensed
Normal

/2.6 **size** `length` *Settable* Default: `11pt`

The size of the glyphs. This value forms the basis of the `em` unit: `1em` is equivalent to the font size.

You can also give the font size itself in `em` units. Then, it is relative to the previous font size.

When used with a suitable variable font, Typst will automatically configure the `opsz` (optical size) font variation[3.7.12/2.33] based on this property, optimizing legibility for the specific size.

```
#set text(size: 20pt)
very #text(1.5em)[big] text
```

very big text

/2.7 **fill** `color` or `gradient` or `tiling` *Settable* Default: `luma(0%)`

The glyph fill paint.

```
#set text(fill: red)
This text is red.
```

This text is red.

/2.8 **stroke** `none` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary`

Settable Default: `none`

How to stroke the text.

```
#text(stroke: 0.5pt + red)[Stroked]
```

Stroked

/2.9 **tracking** `length` *Settable* Default: `0pt`

The amount of space that should be added between characters.

```
#set text(tracking: 1.5pt)
Distant text.
```

Distant text.

/2.10 **spacing** `relative` *Settable* Default: `100% + 0pt`

The amount of space between words.

Can be given as an absolute length, but also relative to the width of the space character in the font.

If you want to adjust the amount of space between characters rather than words, use the tracking[3.7.12/2.9] property instead.

```
#set text(spacing: 200%)
Text with distant words.
```

Text with distant words.

/2.11 **cjk-latin-spacing** `none` or `auto` *Settable* Default: `auto`

Whether to automatically insert spacing between CJK and Latin characters.

```
#set text(cjk-latin-spacing: auto)
第4章介绍了基本的API。

#set text(cjk-latin-spacing: none)
第4章介绍了基本的API。
```

第 4 章介绍了基本的 API.

第4章介绍了基本的API.

/2.12 **baseline** `length` *Settable* Default: `0pt`

An amount to shift the text baseline by.

```
A #text(baseline: 3pt)[lowered]
word.
```

A lowered word.

/2.13 **overhang** `bool` *Settable* Default: `true`

Whether certain glyphs can hang over into the margin in justified text. This can make justification visually more pleasing.

```
#set page(width: 220pt)

#set par(justify: true)
This justified text has a hyphen in
the paragraph's second line. Hanging
the hyphen slightly into the margin
results in a clearer paragraph edge.

#set text(overhang: false)
This justified text has a hyphen in
the paragraph's second line. Hanging
the hyphen slightly into the margin
results in a clearer paragraph edge.
```

This justified text has a hyphen in the paragraph's second line. Hanging the hyphen slightly into the margin results in a clearer paragraph edge.

This justified text has a hyphen in the paragraph's second line. Hanging the hyphen slightly into the margin results in a clearer paragraph edge.

/2.14 **top-edge** `length` or `str` *Settable* Default: `"cap-height"`

The top end of the conceptual frame around the text used for layout and positioning. This affects the size of containers that hold text.

```
#set rect(inset: 0pt)
#set text(size: 20pt)

#set text(top-edge: "ascender")
#rect(fill: aqua)[Typst]

#set text(top-edge: "cap-height")
#rect(fill: aqua)[Typst]
```

Typst

Typst

Variant	Details
`"ascender"`	The font's ascender, which typically exceeds the height of all glyphs.
`"cap-height"`	The approximate height of uppercase letters.
`"x-height"`	The approximate height of non-ascending lowercase letters.
`"baseline"`	The baseline on which the letters rest.
`"bounds"`	The top edge of the glyph's bounding box.

/2.15 **bottom-edge** `length` or `str` *Settable* Default: `"baseline"`

The bottom end of the conceptual frame around the text used for layout and positioning. This affects the size of containers that hold text.

```
#set rect(inset: 0pt)
#set text(size: 20pt)

#set text(bottom-edge: "baseline")
#rect(fill: aqua)[Typst]

#set text(bottom-edge: "descender")
#rect(fill: aqua)[Typst]
```

Typst

Typst

Variant	Details
`"baseline"`	The baseline on which the letters rest.
`"descender"`	The font's descender, which typically exceeds the depth of all glyphs.
`"bounds"`	The bottom edge of the glyph's bounding box.

/2.16 **lang** `str` *Settable* Default: `"en"`

An ISO 639-1/2/3 language code.[A]

[A] https://en.wikipedia.org/wiki/ISO_639

Setting the correct language affects various parts of Typst:

- The text processing pipeline can make more informed choices.
- Hyphenation will use the correct patterns for the language.
- Smart quotes[3.7.8] turns into the correct quotes for the language.
- And all other things which are language-aware.

Choosing the correct language is important for accessibility. For example, screen readers will use it to choose a voice that matches the language of the text. If your document is in another language than English (the default), you should set the text language at the start of your document, before any other content. You can, for example, put it right after the `#set document(/* ... */)` rule that sets your document's title[3.6.6/3.3].

If your document contains passages in a different language than the main language, you should locally change the text language just for those parts, either with a set rule scoped to a block[3.3.2] or using a direct text function call such as `#text(lang: "de")[...]`.

If multiple codes are available for your language, you should prefer the two-letter code (ISO 639-1) over the three-letter codes (ISO 639-2/3). When you have to use a three-letter code and your language differs between ISO 639-2 and ISO 639-3, use ISO 639-2 for PDF 1.7 (Typst's default for PDF export) and below and ISO 639-3 for PDF 2.0 and HTML export.

The language code is case-insensitive, and will be lowercased when accessed through context[3.4].

Example · Setting the text language to German

```
#set text(lang: "de")
#outline()

= Einleitung
In diesem Dokument, ...
```



/2.17 **region** `none` or `str` *Settable* Default: `none`

An ISO 3166-1 alpha-2 region code.[A]

This lets the text processing pipeline make more informed choices.

The region code is case-insensitive, and will be uppercased when accessed through context[3.4].

/2.18 **script** `auto` or `str` *Settable* Default: `auto`

The OpenType writing script.

The combination of `lang` and `script` determine how font features, such as glyph substitution, are implemented. Frequently the value is a modified (all-lowercase) ISO 15924 script identifier, and the `math` writing script is used for features appropriate for mathematical symbols.

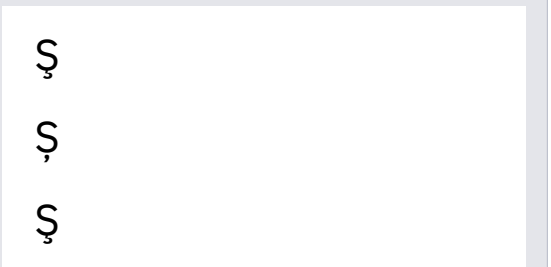
When set to `auto`, the default and recommended setting, an appropriate script is chosen for each block of characters sharing a common Unicode script property.

```
#set text(
  font: "IBM Plex Sans",
  size: 20pt,
)

#let scedilla = [Ş]
#scedilla // S with a cedilla

#set text(lang: "ro", script: "latn")
#scedilla // S with a subscript comma

#set text(lang: "ro", script: "grek")
#scedilla // S with a cedilla
```



[A] https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2

/2.19 **dir** `auto` or `direction` *Settable* Default: `auto`

The dominant direction for text and inline objects. Possible values are:

– `auto`: Automatically infer the direction from the `lang` property.
– `ltr`: Layout text from left to right.
– `rtl`: Layout text from right to left.

When writing in right-to-left scripts like Arabic or Hebrew, you should set the text language[3.7.12/2.16] or direction. While individual runs of text are automatically layouted in the correct direction, setting the dominant direction gives the bidirectional reordering algorithm the necessary information to correctly place punctuation and inline objects. Furthermore, setting the direction affects the alignment values `start` and `end`, which are equivalent to `left` and `right` in `ltr` text and the other way around in `rtl` text.

If you set this to `rtl` and experience bugs or in some way bad looking output, please get in touch with us through the Forum[A], Discord server[B], or our contact form[C].

```
#set text(dir: rtl)
هذا عربي.
```

هذا عربي.

/2.20 **hyphenate** `auto` or `bool` *Settable* Default: `auto`

Whether to hyphenate text to improve line breaking. When `auto`, text will be hyphenated if and only if justification is enabled.

Setting the text language[3.7.12/2.16] ensures that the correct hyphenation patterns are used.

```
#set page(width: 200pt)

#set par(justify: true)
This text illustrates how
enabling hyphenation can
improve justification.

#set text(hyphenate: false)
This text illustrates how
enabling hyphenation can
improve justification.
```

This text illustrates how enabling hyphenation can improve justification.

This text illustrates how enabling hyphenation can improve justification.

/2.21 **costs** `dictionary` *Settable*

Default:
```
(
  hyphenation: 100%,
  runt: 100%,
  widow: 100%,
  orphan: 100%,
)
```

The "cost" of various choices when laying out text. A higher cost means the layout engine will make the choice less often. Costs are specified as a ratio of the default cost, so `50%` will make text layout twice as eager to make a given choice, while `200%` will make it half as eager.

Currently, the following costs can be customized:
– `hyphenation`: splitting a word across multiple lines
– `runt`: ending a paragraph with a line with a single word
– `widow`: leaving a single line of paragraph on the next page
– `orphan`: leaving single line of paragraph on the previous page

Hyphenation is generally avoided by placing the whole word on the next line, so a higher hyphenation cost can result in awkward justification spacing. Note: Hyphenation costs will only be applied when the `linebreaks`[3.6.15/4.5] are set to "optimized". (For example by default implied by `justify`[3.6.15/4.3].)

[A] https://forum.typst.app/
[B] https://discord.gg/2uDybryKPe
[C] https://typst.app/contact

Runts are avoided by placing more or fewer words on previous lines, so a higher runt cost can result in more awkward in justification spacing.

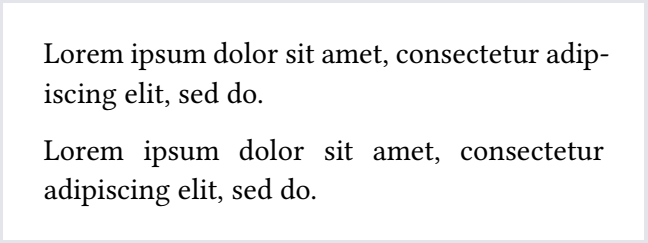
Text layout prevents widows and orphans by default because they are generally discouraged by style guides. However, in some contexts they are allowed because the prevention method, which moves a line to the next page, can result in an uneven number of lines between pages. The `widow` and `orphan` costs allow disabling these modifications. (Currently, `0%` allows widows/orphans; anything else, including the default of `100%`, prevents them. More nuanced cost specification for these modifications is planned for the future.)

```
#set text(hyphenate: true, size: 11.4pt)
#set par(justify: true)

#lorem(10)

// Set hyphenation to ten times the normal cost.
#set text(costs: (hyphenation: 1000%))

#lorem(10)
```



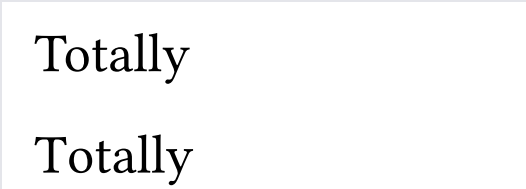
/2.22 **kerning** `bool` *Settable* Default: `true`

Whether to apply kerning.

When enabled, specific letter pairings move closer together or further apart for a more visually pleasing result. The example below demonstrates how decreasing the gap between the “T” and “o” results in a more natural look. Setting this to `false` disables kerning by turning off the OpenType `kern` font feature.

```
#set text(size: 25pt)
Totally

#set text(kerning: false)
Totally
```



/2.23 **alternates** `bool` or `int` *Settable* Default: `0`

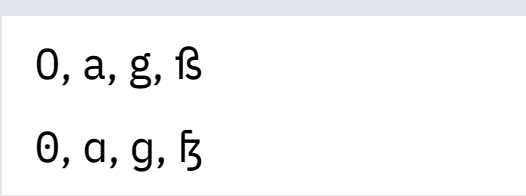
Whether to apply stylistic alternates.

Sometimes fonts contain alternative glyphs for the same codepoint. Setting this to `true` switches to these by enabling the OpenType `salt` font feature. An integer may be used to select between multiple alternates.

```
#set text(
  font: "IBM Plex Sans",
  size: 20pt,
)

0, a, g, ß

#set text(alternates: true)
0, a, g, ß
```



/2.24 **stylistic-set** `none` or `int` or `array` *Settable* Default: ()

Which stylistic sets to apply. Font designers can categorize alternative glyphs forms into stylistic sets. As this value is highly font-specific, you need to consult your font to know which sets are available.

This can be set to an integer or an array of integers, all of which must be between 1 and 20, enabling the corresponding OpenType feature(s) from `ss01` to `ss20`. Setting this to `none` will disable all stylistic sets.

```
#set text(font: "IBM Plex Serif")
ß vs #text(stylistic-set: 5)[ß] \
10 years ago vs #text(stylistic-set: (1, 2, 3))[10 years ago]
```

ß vs ẞ
10 years ago vs 10 years ago

/2.25 **ligatures** `bool` *Settable* Default: `true`

Whether standard ligatures are active.

Certain letter combinations like "fi" are often displayed as a single merged glyph called a *ligature*. Setting this to `false` disables these ligatures by turning off the OpenType `liga` and `clig` font features.

```
#set text(size: 20pt)
A fine ligature.

#set text(ligatures: false)
A fine ligature.
```

A fine ligature.

A fine ligature.

Note that some programming fonts use other OpenType font features to implement "ligatures," including the contextual alternates (`calt`) feature, which is also enabled by default. Use the general `features`[3.7.12/2.32] parameter to control such features.

/2.26 **discretionary-ligatures** `bool` *Settable* Default: `false`

Whether ligatures that should be used sparingly are active. Setting this to `true` enables the OpenType `dlig` font feature.

/2.27 **historical-ligatures** `bool` *Settable* Default: `false`

Whether historical ligatures are active. Setting this to `true` enables the OpenType `hlig` font feature.

/2.28 **number-type** `auto` or `str` *Settable* Default: `auto`

Which kind of numbers / figures to select. When set to `auto`, the default numbers for the font are used.

```
#set text(font: "Noto Sans", 20pt)
#set text(number-type: "lining")
Number 9.

#set text(number-type: "old-style")
Number 9.
```

Number 9.

Number 9.

Variant	Details
`"lining"`	Numbers that fit well with capital text (the OpenType `lnum` font feature).
`"old-style"`	Numbers that fit well into a flow of upper- and lowercase text (the OpenType `onum` font feature).

/2.29 **number-width** `auto` or `str` *Settable* Default: `auto`

The width of numbers / figures. When set to `auto`, the default numbers for the font are used.

```
#set text(font: "Noto Sans", 20pt)
#set text(number-width: "proportional")
A 12 B 34. \
A 56 B 78.

#set text(number-width: "tabular")
A 12 B 34. \
A 56 B 78.
```

A 12 B 34.
A 56 B 78.

A 12 B 34.
A 56 B 78.

Variant	Details
`"proportional"`	Numbers with glyph-specific widths (the OpenType `pnum` font feature).
`"tabular"`	Numbers of equal width (the OpenType `tnum` font feature).

/2.30 **slashed-zero** `bool` *Settable* Default: `false`

Whether to have a slash through the zero glyph. Setting this to `true` enables the OpenType `zero` font feature.

```
0, #text(slashed-zero: true)[0]
```

0, 0

/2.31 **fractions** `bool` *Settable* Default: `false`

Whether to turn numbers into fractions. Setting this to `true` enables the OpenType `frac` font feature.

It is not advisable to enable this property globally as it will mess with all appearances of numbers after a slash (e.g., in URLs). Instead, enable it locally when you want a fraction.

```
1/2 \
#text(fractions: true)[1/2]
```

1/2
1/2

/2.32 **features** `array` or `dictionary` *Settable* Default: `(:)`

Raw OpenType features to apply.

- If given an array of strings, sets the features identified by the strings to `1`.
- If given a dictionary mapping to numbers, sets the features identified by the keys to the values. This allows interacting with non-boolean features such as `swsh`.

Example · Give an array of strings

```
// Enable the `frac` feature manually.
#set text(features: ("frac",))
1/2
```

1/2

Example · Give a dictionary mapping to numbers

```
#set text(font: "Cascadia Code")
=>
// Disable the contextual alternates (`calt`) feature.
#set text(features: (calt: 0))
=>
```

```
⇒ =>
```

/2.33 **variations** `dictionary` *Settable* Default: `(:)`

Raw OpenType font variations to apply.

While classic static fonts require a separate font file for each style combination, variable fonts have *variation axes* from which many different styles can be instanced. Variation axes are identified by case-sensitive four-letter strings.

There are a few well-known variation axes, for which Typst will automatically set suitable values based on the text `weight`[3.7.12/2.4], `stretch`[3.7.12/2.5], `style`[3.7.12/2.3], and `size`[3.7.12/2.6]. This includes:
- `wght`: Weight (e.g., 400 for regular, 700 for bold)
- `wdth`: Width (percentage, e.g., 100 for normal)
- `slnt`: Slant (degrees, negative for right-leaning)
- `ital`: Italic (0 for upright, 1 for italic)
- `opsz`: Optical size (in points)

Fonts can also define custom variation axes to realize arbitrary visual effects. For example, a font's appearance could become more whimsical the higher a particular axis value is set.

With the `variations` parameter, you can directly set values for the axes supported by the active font. It only has an effect when used with a suitable variable font that supports the specified axes. You can use the parameter both to override automatically set values for the well-known axes and to set values for custom axes.

The value should be a dictionary mapping axis tags (four-character strings[3.5.27]) to their values (floating-point numbers[3.5.14]).

Example · Setting values for custom axes

```
#set text(font: "Fraunces", size: 60pt)

#text(variations: (SOFT: 0))[Soft? No.] \
#text(variations: (SOFT: 100))[Soft? Yes.]
```

Soft? No.

Soft? Yes.

Example · Overriding automatically set values

```
#set text(
  font: "Roboto Flex",
  weight: 900,
  stretch: 150%,
)

Wide and Heavy
```

Wide and Heavy

Forced back to normal, but still wide.

```
#text(variations: (wght: 400))[
  Forced back to normal,
  but still wide.
]
```

/2.34 **body** `content` Default: []

Content in which all text is styled according to the other arguments.

/2.35 **text** `str` *Required* *Positional*

The text.

3.7.13 `underline` Element

Underlines text.

/1 Example

```
This is #underline[important].
```

This is important.

/2 Parameters

```
underline(
  stroke: auto length color gradient stroke tiling dictionary,
  offset: auto length,
  extent: length,
  evade: bool,
  background: bool,
  content,
) → content
```

/2.1 `stroke` `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary`

Settable Default: `auto`

How to stroke[3.11.12] the line.

If set to `auto`, takes on the text's color and a thickness defined in the current font.

```
Take #underline(
  stroke: 1.5pt + red,
  offset: 2pt,
  [care],
)
```

Take care

/2.2 `offset` `auto` or `length` *Settable* Default: `auto`

The position of the line relative to the baseline, read from the font tables if `auto`.

```
#underline(offset: 5pt)[
  The Tale Of A Faraway Line I
]
```

The Tale Of A Faraway Line I

/2.3 `extent` `length` *Settable* Default: `0pt`

The amount by which to extend the line beyond (or within if negative) the content.

```
#align(center,
  underline(extent: 2pt)[Chapter 1]
)
```

Chapter 1

/2.4 `evade` `bool` *Settable* Default: `true`

Whether the line skips sections in which it would collide with the glyphs.

```
This #underline(evade: true)[is great].
This #underline(evade: false)[is less great].
```

This is great. This is less great.

/2.5 **background** `bool` *Settable* Default: `false`

Whether the line is placed behind the content it underlines.

```
#set underline(stroke: aqua + 5pt, evade: false)
#underline(background: true)[Fully visible.] \
#underline(background: false)[Partially hidden.]
```

Fully visible.
Partially hidden.

/2.6 **body** `content` *Required* *Positional*

The content to underline.

3.7.14 upper

Converts a string or content to uppercase.

/1 Example

```
#upper("abc") \
#upper[*my text*] \
#upper[ALREADY HIGH]
```

ABC
MY TEXT
ALREADY HIGH

/2 Parameters

```
upper( str content ) → str content
```

/2.1 **text** `str` or `content` *Required* *Positional*

The text to convert to uppercase.

3.8 Math

Typst has special syntax[3.1.3] and library functions to typeset mathematical formulas. Math formulas can be displayed inline with text or as separate blocks. They will be typeset into their own block if they start and end with at least one space (e.g. `$ x^2 $`).

3.8.1 Variables

In math, single letters are always displayed as is. Multiple letters, however, are interpreted as variables and functions. To display multiple letters verbatim, you can place them into quotes and to access single letter variables, you can use the hash syntax[3.3.1].

```
$ A = pi r^2 $
$ "area" = pi dot "radius"^2 $
$ cal(A) :=
    { x in RR | x "is natural" } $
#let x = 5
$ #x < 17 $
```

$$A = \pi r^2$$

$$\text{area} = \pi \cdot \text{radius}^2$$

$$\mathcal{A} := \{x \in \mathbb{R} \mid x \text{ is natural}\}$$

$$5 < 17$$

3.8.2 Symbols

Math mode makes a wide selection of symbols[3.9.2] like `pi`, `dot`, or `RR` available. Many mathematical symbols are available in different variants. You can select between different variants by applying modifiers[3.5.28] to the symbol. Typst further recognizes a number of shorthand sequences like `=>` that approximate a symbol. When such a shorthand exists, the symbol's documentation lists it.

```
$ x < y => x gt.eq.not y $
```

$$x < y \Rightarrow x \gneq y$$

3.8.3 Line Breaks

Formulas can also contain line breaks. Each line can contain one or multiple *alignment points* (`&`) which are then aligned.

```
$ sum_(k=0)^n k
    &= 1 + ... + n \
    &= (n(n+1)) / 2 $
```

$$\sum_{k=0}^{n} k = 1 + \ldots + n$$

$$= \frac{n(n+1)}{2}$$

3.8.4 Function calls

Math mode supports special function calls without the hash prefix. In these "math calls", the argument list works a little differently than in code:

- Within them, Typst is still in "math mode". Thus, you can write math directly into them, but need to use hash syntax to pass code expressions (except for strings, which are available in the math syntax).
- They support positional and named arguments, as well as argument spreading, but don't support trailing content blocks.
- They provide additional syntax for 2-dimensional argument lists. The semicolon (`;`) merges preceding arguments separated by commas into an array argument.

```
$ frac(a^2, 2) $
$ vec(1, 2, delim: "[") $
$ mat(1, 2; 3, 4) $
$ mat(..#range(1, 5).chunks(2)) $
$ lim_x =
    op("lim", limits: #true)_x $
```

$$\frac{a^2}{2}$$

$$\begin{bmatrix}1\\2\end{bmatrix}$$

$$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$$

$$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$$

$$\lim_x = \lim_x$$

To write a verbatim comma or semicolon in a math call, escape it with a backslash. The colon on the other hand is only recognized in a special way if directly preceded by an identifier, so to display it verbatim in those cases, you can just insert a space before it.

Functions calls preceded by a hash are normal code function calls and not affected by these rules.

3.8.5 Alignment

When equations include multiple *alignment points* (&), this creates blocks of alternatingly right- and left-aligned columns. In the example below, the expression `(3x + y) / 7` is right-aligned and `= 9` is left-aligned. The word "given" is also left-aligned because `&&` creates two alignment points in a row, alternating the alignment twice. `& &` and `&&` behave exactly the same way. Meanwhile, "multiply by 7" is right-aligned because just one `&` precedes it. Each alignment point simply alternates between right-aligned/left-aligned.

```
$ (3x + y) / 7 &= 9 && "given" \
  3x + y &= 63 & "multiply by 7" \
  3x &= 63 - y && "subtract y" \
  x &= 21 - y/3 & "divide by 3" $
```

$$\begin{aligned}\frac{3x+y}{7} &= 9 && \text{given}\\ 3x+y &= 63 && \text{multiply by 7}\\ 3x &= 63-y && \text{subtract y}\\ x &= 21-\frac{y}{3} && \text{divide by 3}\end{aligned}$$

3.8.6 Math fonts

You can set the math font by with a show-set rule[3.2.2] as demonstrated below. Note that only special OpenType math fonts are suitable for typesetting maths.

```
#show math.equation: set text(font: "Pennstander Math")
$ sum_(i in NN) 1 + i $
```

$$\sum_{i\in\mathbb{N}} 1 + i$$

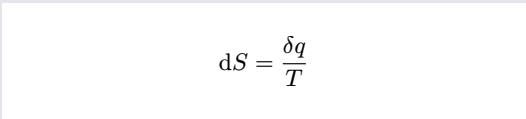
3.8.7 Math module

All math functions are part of the `math` module[3.3.8], which is available by default in equations. Outside of equations, they can be accessed with the `math.` prefix.

3.8.8 Accessibility

To make math accessible, you must provide alternative descriptions of equations in natural language using the `alt` parameter of `math.equation`[3.8.15/3.5]. For more information, see the Textual Representations section of the Accessibility Guide[4.4.7].

```
#math.equation(
  alt: "d S equals delta q divided by T",
  block: true,
  $ dif S = (delta q) / T $,
)
```



In the future, Typst will automatically make equations without alternative descriptions accessible in HTML and PDF 2.0 export.

Definitions

3.8.9 Accent Attaches an accent to a base. 267
3.8.10 Attach Subscript, superscripts, and limits. 269
3.8.11 Binomial A binomial expression. 271
3.8.12 Cancel Displays a diagonal line over a part of an equation. 272
3.8.13 Cases A case distinction. 274
3.8.14 Class Forced use of a certain math class. 275
3.8.15 Equation A mathematical equation. 276
3.8.16 Fraction A mathematical fraction. 279
3.8.17 Left/Right Delimiter matching. 281
3.8.18 Matrix A matrix. 285
3.8.19 Primes Grouped primes. 288
3.8.20 Roots Square and non-square roots. 289
3.8.21 Sizes Forced size styles for expressions within formulas. 290
3.8.22 Stretch Stretches a glyph. 292
3.8.23 Styles Alternate letterforms within formulas. 293
3.8.24 Text Operator A text operator in an equation. 294
3.8.25 Under/Over Delimiters above or below parts of an equation. 295
3.8.26 Variants Alternate typefaces within formulas. 299
3.8.27 Vector A column vector. 302

3.8.9 accent Element

Attaches an accent to a base.

In math mode, common accents are also available as named symbols[3.5.28] that can be directly called (like functions[3.5.15]) to attach them to some content.

/1 Example

```
$grave(a) = accent(a, `)$ \
$arrow(a) = accent(a, arrow)$ \
$tilde(a) = accent(a, \u{0303})$
```

$$\grave{a} = \grave{a}$$
$$\vec{a} = \vec{a}$$
$$\tilde{a} = \tilde{a}$$

/2 Parameters

```
accent(
  content,
  str content,
  size: relative,
  dotless: bool,
) → content
```

/2.1 base content *Required* *Positional*

The base to which the accent is applied. May consist of multiple letters.

```
$arrow(A B C)$
```

$$\overrightarrow{ABC}$$

/2.2 accent str or content *Required* *Positional*

The accent to apply to the base.

Supported accents include:

Accent	Name	Codepoint
Grave	`grave`	`
Acute	`acute`	´
Circumflex	`hat`	^
Tilde	`tilde`	~
Macron	`macron`	¯
Dash	`dash`	‾
Breve	`breve`	˘
Dot	`dot`	.
Double dot, Diaeresis	`dot.double`, `diaer`	¨
Triple dot	`dot.triple`	\u{20db}
Quadruple dot	`dot.quad`	\u{20dc}
Circle	`circle`	∘
Double acute	`acute.double`	˝
Caron	`caron`	ˇ
Right arrow	`arrow`, `->`	→
Left arrow	`arrow.l`, `<-`	←

Accent	Name	Codepoint
Left/Right arrow	`arrow.l.r`	↔
Right harpoon	`harpoon`	⇀
Left harpoon	`harpoon.lt`	↼

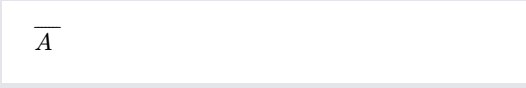
/2.3 **size** `relative` *Settable*

Default: `100% + 0pt`

The size of the accent, relative to the width of the base.

Example · Basic usage

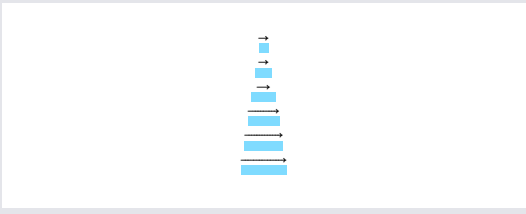
```
$dash(A, size: #150%)$
```



Note that the resulting accent may not have the exact desired size. For example, an arrow may be either a pre-defined short glyph, or a long glyph assembled from building blocks (arrowhead + line) provided by the font. The sizes of the two possibilities may not cover the entire span. Consequently, arrows of certain intermediate sizes cannot be constructed.

Example · Size of arrow growing discontinuously

```
#for i in range(6) {
  $ arrow(#box(
    width: 0.4em + 0.3em * i,
    fill: aqua,
    height: 0.4em,
  )) $
}
```



/2.4 **dotless** `bool` *Settable*

Default: `true`

Whether to remove the dot on top of lowercase i and j when adding a top accent.

This enables the `dtls` OpenType feature.

```
$hat(dotless: #false, i)$
```

3.8.10 Attach

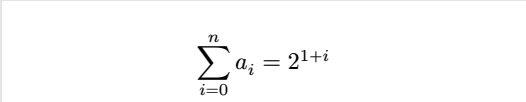
Subscript, superscripts, and limits.

Attachments can be displayed either as sub/superscripts, or limits. Typst automatically decides which is more suitable depending on the base, but you can also control this manually with the `scripts` and `limits` functions.

If you want the base to stretch to fit long top and bottom attachments (for example, an arrow with text above it), use the `stretch`[3.8.22] function.

/1 Example

```
$ sum_(i=0)^n a_i = 2^(1+i) $
```



/2 Syntax

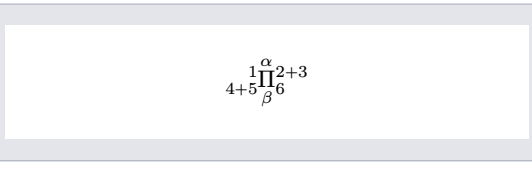
This function also has dedicated syntax for attachments after the base: Use the underscore (_) to indicate a subscript i.e. bottom attachment and the hat (^) to indicate a superscript i.e. top attachment.

/3 Functions

/3.1 attach Element

A base with optional attachments.

```
$ attach(
  Pi, t: alpha, b: beta,
  tl: 1, tr: 2+3, bl: 4+5, br: 6,
) $
```



If you want to add accents (hats, tildes, arrows, etc.) instead of scripts or corner attachments, use the `accent`[3.8.9] function instead.

```
math.attach(
  content,
  t: none content,
  b: none content,
  tl: none content,
  bl: none content,
  tr: none content,
  br: none content,
) → content
```

/3.1.1 base content *Required* *Positional*

The base to which things are attached.

/3.1.2 t none or content *Settable* Default: none

The top attachment, smartly positioned at top-right or above the base.

You can wrap the base in `limits()` or `scripts()` to override the smart positioning.

/3.1.3 b none or content *Settable* Default: none

The bottom attachment, smartly positioned at the bottom-right or below the base.

You can wrap the base in `limits()` or `scripts()` to override the smart positioning.

/3.1.4 **tl** `none` or `content` *Settable* Default: `none`

The top-left attachment (before the base).

/3.1.5 **bl** `none` or `content` *Settable* Default: `none`

The bottom-left attachment (before base).

/3.1.6 **tr** `none` or `content` *Settable* Default: `none`

The top-right attachment (after the base).

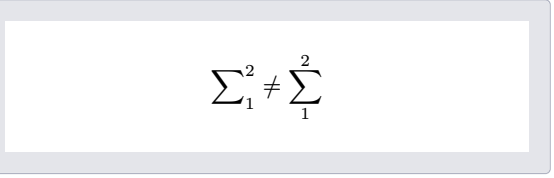
/3.1.7 **br** `none` or `content` *Settable* Default: `none`

The bottom-right attachment (after the base).

/3.2 scripts Element

Forces a base to display attachments as scripts.

```
$ scripts(sum)_1^2 != sum_1^2 $
```



```
math.scripts(content) → content
```

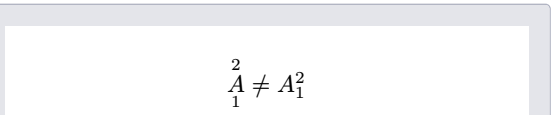
/3.2.1 **body** `content` *Required* *Positional*

The base to attach the scripts to.

/3.3 limits Element

Forces a base to display attachments as limits.

```
$ limits(A)_1^2 != A_1^2 $
```



```
math.limits(
  content,
  inline: bool,
) → content
```

/3.3.1 **body** `content` *Required* *Positional*

The base to attach the limits to.

/3.3.2 **inline** `bool` *Settable* Default: `true`

Whether to also force limits in inline equations.

When applying limits globally (e.g., through a show rule), it is typically a good idea to disable this.

3.8.11 `binom` Element

A binomial expression.

/1 Example

```
$ binom(n, k) $
$ binom(n, k_1, k_2, k_3, ..., k_m) $
```

$$\binom{n}{k}$$

$$\binom{n}{k_1, k_2, k_3, ..., k_m}$$

/2 Parameters

```
binom(
  content,
  ..content,
) → content
```

/2.1 **upper** `content` *Required* *Positional*

The binomial's upper index.

/2.2 **lower** `content` *Required* *Positional* *Variadic*

The binomial's lower index.

3.8.12 `cancel` Element

Displays a diagonal line over a part of an equation.

This is commonly used to show the elimination of a term.

/1 Example

```
Here, we can simplify:
$ (a dot b dot cancel(x)) /
    cancel(x) $
```

Here, we can simplify:

$$\frac{a \cdot b \cdot \cancel{x}}{\cancel{x}}$$

/2 Parameters

```
cancel(
  content,
  length: relative,
  inverted: bool,
  cross: bool,
  angle: auto angle function,
  stroke: length color gradient stroke tiling dictionary,
) → content
```

/2.1 **body** `content` *Required* *Positional*

The content over which the line should be placed.

/2.2 **length** `relative` *Settable* Default: `100% + 0.3em`

The length of the line, relative to the length of the diagonal spanning the whole element being "cancelled". A value of `100%` would then have the line span precisely the element's diagonal.

```
$ a + cancel(x, length: #200%)
    - cancel(x, length: #200%) $
```

$$a + \cancel{x} - \cancel{x}$$

/2.3 **inverted** `bool` *Settable* Default: `false`

Whether the cancel line should be inverted (flipped along the y-axis). For the default angle setting, inverted means the cancel line points to the top left instead of top right.

```
$ (a cancel((b + c), inverted: #true)) /
    cancel(b + c, inverted: #true) $
```

$$\frac{a\cancel{(b + c)}}{\cancel{b + c}}$$

/2.4 **cross** `bool` *Settable* Default: `false`

Whether two opposing cancel lines should be drawn, forming a cross over the element. Overrides `inverted`.

```
$ cancel(Pi, cross: #true) $
```

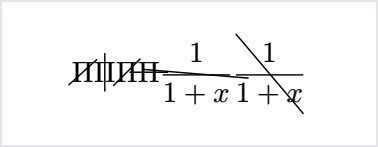
$$\cancel{\Pi}$$

/2.5 **angle** `auto` or `angle` or `function` *Settable* Default: `auto`

How much to rotate the cancel line.

- If given an angle, the line is rotated by that angle clockwise with respect to the y-axis.
- If `auto`, the line assumes the default angle; that is, along the rising diagonal of the content box.
- If given a function `angle => angle`, the line is rotated, with respect to the y-axis, by the angle returned by that function. The function receives the default angle as its input.

```
$ cancel(Pi)
  cancel(Pi, angle: #0deg)
  cancel(Pi, angle: #45deg)
  cancel(Pi, angle: #90deg)
  cancel(1/(1+x), angle: #(a => a + 45deg))
  cancel(1/(1+x), angle: #(a => a + 90deg)) $
```



/2.6 **stroke** `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable*

Default: `0.05em`

How to stroke[3.11.12] the cancel line.

```
$ cancel(
  sum x,
  stroke: #(
    paint: red,
    thickness: 1.5pt,
    dash: "dashed",
  ),
) $
```



3.8.13 `cases` Element

A case distinction.

Content across different branches can be aligned with the `&` symbol.

/1 Example

```
$ f(x, y) := cases(
  1 "if" (x dot y)/2 <= 0,
  2 "if" x "is even",
  3 "if" x in NN,
  4 "else",
) $
```

$$f(x, y) := \begin{cases} 1 \text{ if } \frac{x \cdot y}{2} \leq 0 \\ 2 \text{ if } x \text{ is even} \\ 3 \text{ if } x \in \mathbb{N} \\ 4 \text{ else} \end{cases}$$

/2 Parameters

```
cases(
  delim: none str symbol array,
  reverse: bool,
  gap: relative,
  ..content,
) → content
```

/2.1 `delim` `none` or `str` or `symbol` or `array` *Settable* Default: `("{", "}")`

The delimiter to use.

Can be a single character specifying the left delimiter, in which case the right delimiter is inferred. Otherwise, can be an array containing a left and a right delimiter.

```
#set math.cases(delim: "[")
$ x = cases(1, 2) $
```

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

/2.2 `reverse` `bool` *Settable* Default: `false`

Whether the direction of cases should be reversed.

```
#set math.cases(reverse: true)
$ cases(1, 2) = x $
```

$$\left.\begin{matrix} 1 \\ 2 \end{matrix}\right\} = x$$

/2.3 `gap` `relative` *Settable* Default: `0% + 0.2em`

The gap between branches.

```
#set math.cases(gap: 1em)
$ x = cases(1, 2) $
```

$$x = \begin{cases} 1 \\ 2 \end{cases}$$

/2.4 `children` `content` *Required* *Positional* *Variadic*

The branches of the case distinction.

3.8.14 `class` Element

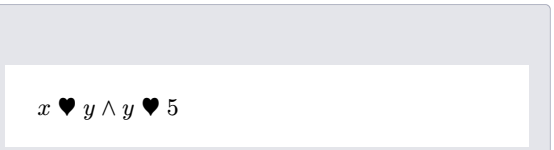
Forced use of a certain math class.

This is useful to treat certain symbols as if they were of a different class, e.g. to make a symbol behave like a relation. The class of a symbol defines the way it is laid out, including spacing around it, and how its scripts are attached by default. Note that the latter can always be overridden using `limits`[3.8.10/3.3] and `scripts`[3.8.10/3.2].

/1 Example

```
#let loves = math.class(
  "relation",
  sym.suit.heart,
)

$x loves y and y loves 5$
```



/2 Parameters

```
class(
  str,
  content,
) → content
```

/2.1 `class` `str` *Required* *Positional*

The class to apply to the content.

Variant	Details
`"normal"`	The default class for non-special things.
`"punctuation"`	Punctuation, e.g. a comma.
`"opening"`	An opening delimiter, e.g. `(`.
`"closing"`	A closing delimiter, e.g. `)`.
`"fence"`	A delimiter that is the same on both sides, e.g. `|`.
`"large"`	A large operator like `sum`. If the body is a single glyph, this class vertically centers it on the math axis (where the fraction line sits) and stretches it vertically when in `display` style.
`"relation"`	A relation like = or `prec`.
`"unary"`	A unary operator like `not`.
`"binary"`	A binary operator like `times`.
`"vary"`	An operator that can be both unary or binary like +.

/2.2 `body` `content` *Required* *Positional*

The content to which the class is applied.

3.8.15 `equation` Element

A mathematical equation.

Can be displayed inline with text or as a separate block. An equation becomes block-level through the presence of whitespace after the opening dollar sign and whitespace before the closing dollar sign.

/1 Example

```
#set text(font: "New Computer Modern")

Let $a$, $b$, and $c$ be the side
lengths of a right-angled triangle.
Then, we know that:
$ a^2 + b^2 = c^2 $

Prove by induction:
$ sum_(k=1)^n k = (n(n+1)) / 2 $
```

Let a, b, and c be the side lengths of a right-angled triangle. Then, we know that:

$$a^2 + b^2 = c^2$$

Prove by induction:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

By default, block-level equations will not break across pages. This can be changed through `show math.equation: set block(breakable: true)`.

/2 Syntax

This function also has dedicated syntax: Write mathematical markup within dollar signs to create an equation. Starting and ending the equation with whitespace lifts it into a separate block that is centered horizontally. For more details about math syntax, see the main math page[3.8].

/3 Parameters

```
equation(
  block: bool,
  numbering: none str function,
  number-align: alignment,
  supplement: none auto content function,
  alt: none str,
  content,
) → content
```

/3.1 `block` `bool` *Settable* Default: `false`

Whether the equation is displayed as a separate block.

/3.2 `numbering` `none` or `str` or `function` *Settable* Default: `none`

How to number block-level equations. Accepts a numbering pattern or function[3.6.13] taking a single number.

```
#set math.equation(numbering: "(1)")

We define:
$ phi.alt := (1 + sqrt(5)) / 2 $ <ratio>

With @ratio, we get:
$ F_n = floor(1 / sqrt(5) phi.alt^n) $
```

We define:

$$\phi := \frac{1 + \sqrt{5}}{2} \tag{1}$$

With Equation 1, we get:

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor \tag{2}$$

/3.3 **number-align** `alignment` *Settable* Default: `end + horizon`

The alignment of the equation numbering.

By default, the alignment is `end + horizon`. For the horizontal component, you can use `right`, `left`, or `start` and `end` of the text direction; for the vertical component, you can use `top`, `horizon`, or `bottom`.

```
#set math.equation(numbering: "(1)", number-align: bottom)

We can calculate:
$ E &= sqrt(m_0^2 + p^2) \
    &approx 125 "GeV" $
```

We can calculate:

$$E = \sqrt{m_0^2 + p^2} \approx 125\ \text{GeV} \quad (1)$$

/3.4 **supplement** `none` or `auto` or `content` or `function` *Settable* Default: `auto`

A supplement for the equation.

For references to equations, this is added before the referenced number.

If a function is specified, it is passed the referenced equation and should return content.

```
#set math.equation(numbering: "(1)", supplement: [Eq.])

We define:
$ phi.alt := (1 + sqrt(5)) / 2 $ <ratio>

With @ratio, we get:
$ F_n = floor(1 / sqrt(5) phi.alt^n) $
```

We define:

$$\phi := \frac{1 + \sqrt{5}}{2} \quad (1)$$

With Eq. 1, we get:

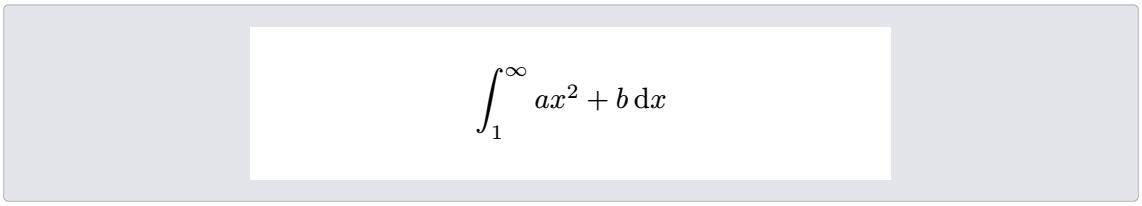
$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor \quad (2)$$

/3.5 **alt** `none` or `str` *Settable* Default: `none`

An alternative description of the mathematical equation.

This should describe the full equation in natural language and will be made available to Assistive Technology. You can learn more in the Textual Representations section of the Accessibility Guide[4.4.7].

```
#math.equation(
  alt: "integral from 1 to infinity of a x squared plus b with respect to x",
  block: true,
  $ integral_1^oo a x^2 + b dif x $,
)
```



/3.6 **body** `content` *Required* *Positional*

The contents of the equation.

3.8.16 `frac` Element

A mathematical fraction.

/1 Example

```
$ 1/2 < (x+1)/2 $
$ ((x+1)) / 2 = frac(a, b) $
```

$$\frac{1}{2} < \frac{x+1}{2}$$

$$\frac{(x+1)}{2} = \frac{a}{b}$$

/2 Syntax

This function also has dedicated syntax: Use a slash to turn neighbouring expressions into a fraction. Multiple atoms can be grouped into a single expression using round grouping parentheses. Such parentheses are removed from the output, but you can nest multiple to force them.

/3 Parameters

```
frac(
  content,
  content,
  style: str,
) → content
```

/3.1 **num** `content` *Required* *Positional*

The fraction's numerator.

/3.2 **denom** `content` *Required* *Positional*

The fraction's denominator.

/3.3 **style** `str` *Settable* Default: `"vertical"`

How the fraction should be laid out.

Example · Styles

```
$ frac(x, y, style: "vertical") $
$ frac(x, y, style: "skewed") $
$ frac(x, y, style: "horizontal") $
```

$$\frac{x}{y}$$

$$x/y$$

$$x/y$$

Example · Setting the default

```
#set math.frac(style: "skewed")
$ a / b $
```

$$a/b$$

Example · Handling of grouping parentheses

```
// Grouping parentheses are removed.
#set math.frac(style: "vertical")
$ (a + b) / b $

// Grouping parentheses are removed.
#set math.frac(style: "skewed")
$ (a + b) / b $

// Grouping parentheses are retained.
#set math.frac(style: "horizontal")
$ (a + b) / b $
```

$$\frac{a+b}{b}$$

$$a+b\,/\,b$$

$$(a+b)/b$$

Example · Different styles in inline vs block equations

```
// This changes the style for inline equations only.
#show math.equation.where(block: false): set math.frac(style: "horizontal")

This $(x-y)/z = 3$ is inline math, and this is block math:
$ (x-y)/z = 3 $
```

This $(x-y)/z = 3$ is inline math, and this is block math:

$$\frac{x-y}{z} = 3$$

Example · Use LaTeX-like convention

```
// Change the default style.
#set math.frac(style: "horizontal")
// Define a shorthand with the original style.
#let frac = math.frac.with(style: "vertical")

$ p/q = frac(p, q) $

// The shadowed definition can still be accessed.
#assert.eq($p/q$, $std.math.frac(p, q)$)
```

$$p/q = \frac{p}{q}$$

Variant	Details
`"vertical"`	Stacked numerator and denominator with a bar.
`"skewed"`	Numerator and denominator separated by a slash.
`"horizontal"`	Numerator and denominator placed inline and parentheses are not absorbed.

3.8.17 Left/Right

Delimiter matching.

The `lr` function allows you to match two delimiters and scale them with the content they contain. While this also happens automatically for delimiters that match syntactically, `lr` allows you to match two arbitrary delimiters and control their size exactly. Apart from the `lr` function, Typst provides a few more functions that create delimiter pairings for absolute, ceiled, and floored values as well as norms.

To prevent a delimiter from being matched by Typst, and thus auto-scaled, escape it with a backslash. To instead disable auto-scaling completely, use `set math.lr(size: 1em)`.

/1 Example

```
$ [a, b/2] $
$ lr(]sum_(x=1)^n], size: #50%) x $
$ abs((x + y) / 2) $
$ \{ (x / y) \} $
#set math.lr(size: 1em)
$ { (a / b), a, b in (0; 1/2] } $
```

$$\left[a, \frac{b}{2}\right]$$

$$\left]\sum_{x=1}^{n}\right] x$$

$$\left|\frac{x+y}{2}\right|$$

$$\{\left(\frac{x}{y}\right)\}$$

$$\{(\frac{a}{b}), a, b \in (0; \frac{1}{2}]\}$$

/2 Functions

/2.1 `lr` Element

Scales delimiters.

While matched delimiters scale by default, this can be used to scale unmatched delimiters and to control the delimiter scaling more precisely.

```
math.lr(
  size: relative,
  content,
) → content
```

/2.1.1 `size` `relative` *Settable* Default: `100% + 0pt`

The size of the brackets, relative to the height of the wrapped content.

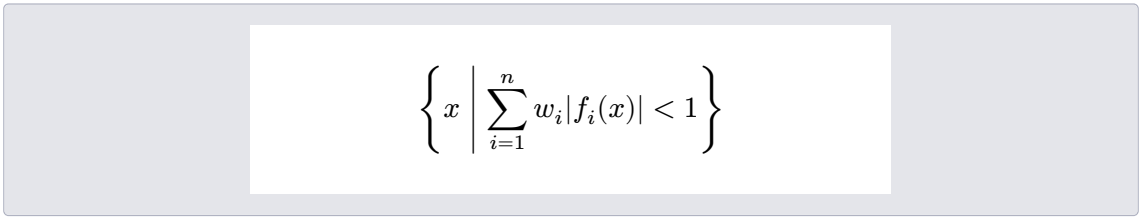
/2.1.2 `body` `content` *Required* *Positional*

The delimited content, including the delimiters.

/2.2 `mid` Element

Scales delimiters vertically to the nearest surrounding `lr()` group.

```
$ { x mid(|) sum_(i=1)^n w_i abs(f_i (x)) < 1 } $
```



```
math.mid(content) → content
```

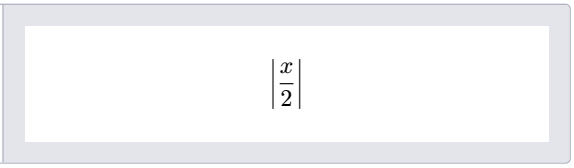
/2.2.1 **body** `content` *Required* *Positional*

The content to be scaled.

/2.3 abs

Takes the absolute value of an expression.

```
$ abs(x/2) $
```



```
math.abs(
  size: relative,
  content,
) → content
```

/2.3.1 **size** `relative`

The size of the brackets, relative to the height of the wrapped content.

Default: The current value of `lr.size`[3.8.17/2.1.1].

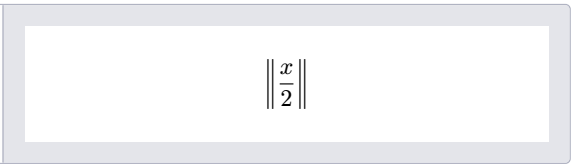
/2.3.2 **body** `content` *Required* *Positional*

The expression to take the absolute value of.

/2.4 norm

Takes the norm of an expression.

```
$ norm(x/2) $
```



```
math.norm(
  size: relative,
  content,
) → content
```

/2.4.1 **size** `relative`

The size of the brackets, relative to the height of the wrapped content.

Default: The current value of `lr.size`[3.8.17/2.1.1].

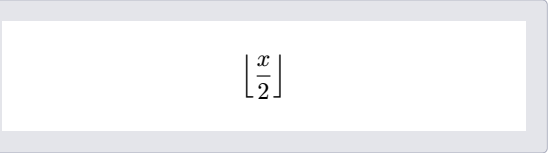
/2.4.2 **body** `content` *Required* *Positional*

The expression to take the norm of.

/2.5 floor

Floors an expression.

```
$ floor(x/2) $
```



```
math.floor(
  size: relative,
  content,
) → content
```

/2.5.1 **size** `relative`

The size of the brackets, relative to the height of the wrapped content.

Default: The current value of `lr.size`[3.8.17/2.1.1].

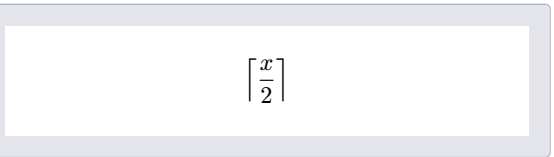
/2.5.2 **body** `content` *Required* *Positional*

The expression to floor.

/2.6 ceil

Ceils an expression.

```
$ ceil(x/2) $
```



```
math.ceil(
  size: relative,
  content,
) → content
```

/2.6.1 **size** `relative`

The size of the brackets, relative to the height of the wrapped content.

Default: The current value of `lr.size`[3.8.17/2.1.1].

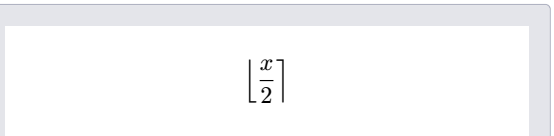
/2.6.2 **body** `content` *Required* *Positional*

The expression to ceil.

/2.7 round

Rounds an expression.

```
$ round(x/2) $
```



```
math.round(
  size: relative,
  content,
) → content
```

/2.7.1 **size** `relative`

The size of the brackets, relative to the height of the wrapped content.

Default: The current value of `lr.size`[3.8.17/2.1.1].

/2.7.2 **body** `content` *Required* *Positional*

The expression to round.

3.8.18 `mat` Element

A matrix.

The elements of a row should be separated by commas, while the rows themselves should be separated by semicolons. The semicolon syntax merges preceding arguments separated by commas into an array. You can also use this special syntax of math function calls to define custom functions that take 2D data.

Content in cells can be aligned with the `align`[3.8.18/2.2] parameter, or content in cells that are in the same row can be aligned with the & symbol.

/1 Example

```
$ mat(
  1, 2, ..., 10;
  2, 2, ..., 10;
  dots.v, dots.v, dots.down, dots.v;
  10, 10, ..., 10;
) $
```

$$\begin{pmatrix} 1 & 2 & \dots & 10 \\ 2 & 2 & \dots & 10 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix}$$

/2 Parameters

```
mat(
  delim: none str symbol array,
  align: alignment,
  augment: none int dictionary,
  gap: relative,
  row-gap: relative,
  column-gap: relative,
  ..array,
) → content
```

/2.1 `delim` `none` or `str` or `symbol` or `array` *Settable* Default: `("(", ")")`

The delimiter to use.

Can be a single character specifying the left delimiter, in which case the right delimiter is inferred. Otherwise, can be an array containing a left and a right delimiter.

```
#set math.mat(delim: "[")
$ mat(1, 2; 3, 4) $
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

/2.2 `align` `alignment` *Settable* Default: `center`

The horizontal alignment that each cell should have.

```
#set math.mat(align: right)
$ mat(-1, 1, 1; 1, -1, 1; 1, 1, -1) $
```

$$\begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix}$$

/2.3 `augment` `none` or `int` or `dictionary` *Settable* Default: `none`

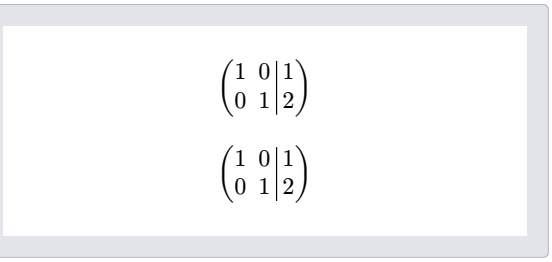
Draws augmentation lines in a matrix.

– `none`: No lines are drawn.

- A single number: A vertical augmentation line is drawn after the specified column number. Negative numbers start from the end.
- A dictionary: With a dictionary, multiple augmentation lines can be drawn both horizontally and vertically. Additionally, the style of the lines can be set. The dictionary can contain the following keys:
  - `hline`: The offsets at which horizontal lines should be drawn. For example, an offset of 2 would result in a horizontal line being drawn after the second row of the matrix. Accepts either an integer for a single line, or an array of integers for multiple lines. Like for a single number, negative numbers start from the end.
  - `vline`: The offsets at which vertical lines should be drawn. For example, an offset of 2 would result in a vertical line being drawn after the second column of the matrix. Accepts either an integer for a single line, or an array of integers for multiple lines. Like for a single number, negative numbers start from the end.
  - `stroke`: How to stroke[3.11.12] the line. If set to `auto`, takes on a thickness of 0.05 em and square line caps.

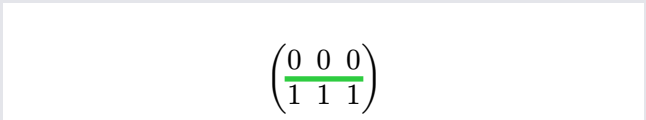
Example · Basic usage

```
$ mat(1, 0, 1; 0, 1, 2; augment: #2) $
// Equivalent to:
$ mat(1, 0, 1; 0, 1, 2; augment: #(-1)) $
```



Example · Customizing the augmentation line

```
$ mat(0, 0, 0; 1, 1, 1; augment: #(hline: 1, stroke: 2pt + green)) $
```

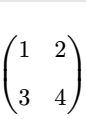


/2.4 **gap** `relative` Default: `0% + 0pt`

The gap between rows and columns.

This is a shorthand to set `row-gap` and `column-gap` to the same value.

```
#set math.mat(gap: 1em)
$ mat(1, 2; 3, 4) $
```



/2.5 **row-gap** `relative` *Settable* Default: `0% + 0.2em`

The gap between rows.

```
#set math.mat(row-gap: 1em)
$ mat(1, 2; 3, 4) $
```



/2.6 **column-gap** `relative` *Settable* Default: `0% + 0.5em`

The gap between columns.

```
#set math.mat(column-gap: 1em)
$ mat(1, 2; 3, 4) $
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

/2.7 **rows** `array` *Required* *Positional* *Variadic*

An array of arrays with the rows of the matrix.

```
#let data = ((1, 2, 3), (4, 5, 6))
#let matrix = math.mat(..data)
$ v := matrix $
```

$$v := \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

3.8.19 `primes` Element

Grouped primes.

```
$ a'''_b = a^''''_b $
```

$$a_b''' = a_b'''$$

/1 Syntax

This function has dedicated syntax: use apostrophes instead of primes. They will automatically attach to the previous element, moving superscripts to the next level.

/2 Parameters

```
primes( int ) → content
```

/2.1 `count` `int` *Required* *Positional*

The number of grouped primes.

3.8.20 Roots

Square and non-square roots.

/1 Example

```
$ sqrt(3 - 2 sqrt(2)) = sqrt(2) - 1 $
$ root(3, x) $
```

$$\sqrt{3 - 2\sqrt{2}} = \sqrt{2} - 1$$

$$\sqrt[3]{x}$$

/2 Functions

/2.1 **root** Element

A general root.

```
$ root(3, x) $
```

$$\sqrt[3]{x}$$

```
math.root(
  none content,
  content,
) → content
```

/2.1.1 **index** `none` or `content` *Positional* *Settable*

Default: `none`

Which root of the radicand to take.

/2.1.2 **radicand** `content` *Required* *Positional*

The expression to take the root of.

/2.2 **sqrt**

A square root.

```
$ sqrt(3 - 2 sqrt(2)) = sqrt(2) - 1 $
```

$$\sqrt{3 - 2\sqrt{2}} = \sqrt{2} - 1$$

```
math.sqrt(content) → content
```

/2.2.1 **radicand** `content` *Required* *Positional*

The expression to take the square root of.

3.8.21 Sizes

Forced size styles for expressions within formulas.

These functions allow manual configuration of the size of equation elements to make them look as in a display/inline equation or as if used in a root or sub/superscripts.

/1 Functions

/1.1 display

Forced display style in math.

This is the normal size for block equations.

```
$sum_i x_i/2 = display(sum_i x_i/2)$
```



```
math.display(
  content,
  cramped: bool,
) → content
```

/1.1.1 **body** `content` *Required* *Positional*

The content to size.

/1.1.2 **cramped** `bool` Default: `false`

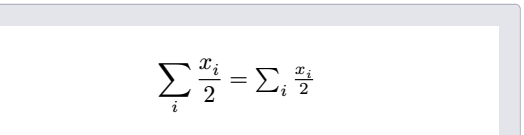
Whether to impose a height restriction for exponents, like regular sub- and superscripts do.

/1.2 inline

Forced inline (text) style in math.

This is the normal size for inline equations.

```
$ sum_i x_i/2
    = inline(sum_i x_i/2) $
```



```
math.inline(
  content,
  cramped: bool,
) → content
```

/1.2.1 **body** `content` *Required* *Positional*

The content to size.

/1.2.2 **cramped** `bool` Default: `false`

Whether to impose a height restriction for exponents, like regular sub- and superscripts do.

/1.3 script

Forced script style in math.

This is the smaller size used in powers or sub- or superscripts.

```
$sum_i x_i/2 = script(sum_i x_i/2)$
```



```
math.script(
  content,
  cramped: bool,
) → content
```

/1.3.1 **body** `content` *Required* *Positional*

The content to size.

/1.3.2 **cramped** `bool` Default: `true`

Whether to impose a height restriction for exponents, like regular sub- and superscripts do.

/1.4 sscript

Forced second script style in math.

This is the smallest size, used in second-level sub- and superscripts (script of the script).

```
$sum_i x_i/2 = sscript(sum_i x_i/2)$
```



```
math.sscript(
  content,
  cramped: bool,
) → content
```

/1.4.1 **body** `content` *Required* *Positional*

The content to size.

/1.4.2 **cramped** `bool` Default: `true`

Whether to impose a height restriction for exponents, like regular sub- and superscripts do.

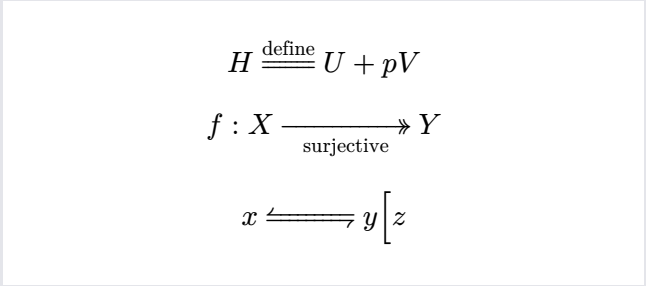
3.8.22 `stretch` Element

Stretches a glyph.

This function can also be used to automatically stretch the base of an attachment, so that it fits the top and bottom attachments.

Note that only some glyphs can be stretched, and which ones can depend on the math font being used. However, most math fonts are the same in this regard.

```
$ H stretch(=)^"define" U + p V $
$ f : X stretch(->>, size: #150%)_"surjective" Y $
$ x stretch(harpoons.ltrb, size: #3em) y
    stretch(\[, size: #150%) z $
```



/1 Parameters

```
stretch(
  content,
  size: relative,
) → content
```

/1.1 **body** `content` *Required* *Positional*

The glyph to stretch.

/1.2 **size** `relative` *Settable*

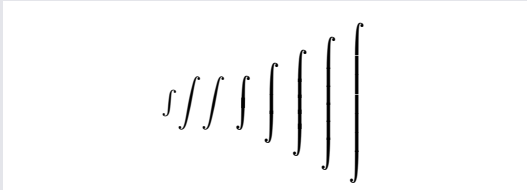
Default: `100% + 0pt`

The size to stretch to, relative to the maximum size of the glyph and its attachments.

Note that the resulting glyph may not have the exact desired size. A stretched glyph may be either a pre-defined glyph, or a glyph assembled from building blocks provided by the font. The possible sizes may not cover the entire span. In the example below, when the `size` parameter is increased from `101%` to `200%`, the selected glyph remains the same, so the actual size does not change.

Example · Size of ∫ growing discontinuously

```
#for size in (
  100%, // short
  101%, 200%, // tall
  201%, 300%, 400%, 500%, 600%, // taller
) {
  $stretch(integral, size: #size)$
}
```



3.8.23 Styles

Alternate letterforms within formulas.

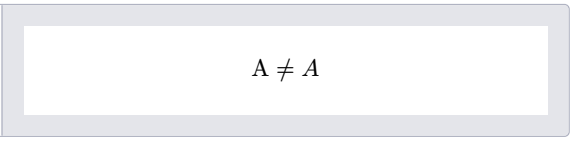
These functions are distinct from the `text`[3.7.12] function because math fonts contain multiple variants of each letter.

/1 Functions

/1.1 upright

Upright (non-italic) font style in math.

```
$ upright(A) != A $
```



```
math.upright( content ) → content
```

/1.1.1 **body** `content` *Required* *Positional*

The content to style.

/1.2 italic

Italic font style in math.

For roman letters and greek lowercase letters, this is already the default.

```
math.italic( content ) → content
```

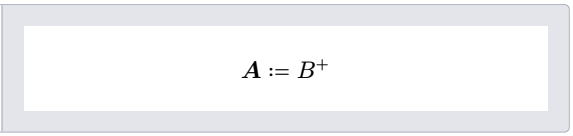
/1.2.1 **body** `content` *Required* *Positional*

The content to style.

/1.3 bold

Bold font style in math.

```
$ bold(A) := B^+ $
```



```
math.bold( content ) → content
```

/1.3.1 **body** `content` *Required* *Positional*

The content to style.

3.8.24 op Element

A text operator in an equation.

/1 Example

```
$ tan x = (sin x)/(cos x) $
$ op("custom",
     limits: #true)_(n->oo) n $
```

$$\tan x = \frac{\sin x}{\cos x}$$

$$\operatorname{custom}_{n \to \infty} n$$

/2 Predefined Operators

Typst predefines the operators `arccos`, `arcsin`, `arctan`, `arg`, `cos`, `cosh`, `cot`, `coth`, `csc`, `csch`, `ctg`, `deg`, `det`, `dim`, `exp`, `gcd`, `lcm`, `hom`, `id`, `im`, `inf`, `ker`, `lg`, `lim`, `liminf`, `limsup`, `ln`, `log`, `max`, `min`, `mod`, `Pr`, `sec`, `sech`, `sin`, `sinc`, `sinh`, `sup`, `tan`, `tanh`, `tg` and `tr`.

/3 Parameters

```
op(
  content,
  limits: bool,
) → content
```

/3.1 text `content` *Required* *Positional*

The operator's text.

/3.2 limits `bool` *Settable* Default: `false`

Whether the operator should show attachments as limits in display mode.

3.8.25 Under/Over

Delimiters above or below parts of an equation.

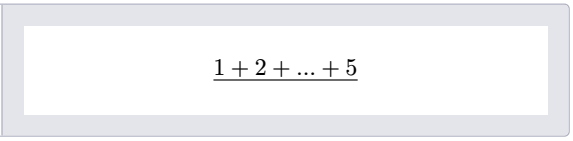
The braces and brackets further allow you to add an optional annotation below or above themselves.

/1 Functions

/1.1 underline Element

A horizontal line under content.

```
$ underline(1 + 2 + ... + 5) $
```



```
math.underline(content) → content
```

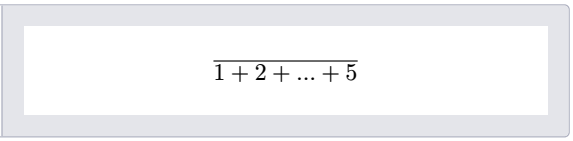
/1.1.1 body content *Required* *Positional*

The content above the line.

/1.2 overline Element

A horizontal line over content.

```
$ overline(1 + 2 + ... + 5) $
```



```
math.overline(content) → content
```

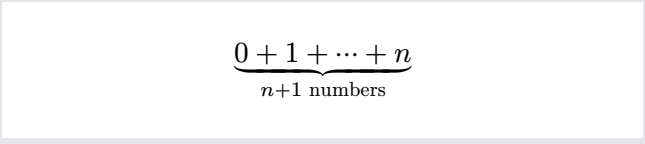
/1.2.1 body content *Required* *Positional*

The content below the line.

/1.3 underbrace Element

A horizontal brace under content, with an optional annotation below.

```
$ underbrace(0 + 1 + dots.c + n, n + 1 "numbers") $
```



```
math.underbrace(
  content,
  none content,
) → content
```

/1.3.1 body content *Required* *Positional*

The content above the brace.

/1.3.2 annotation none or content *Positional* *Settable*

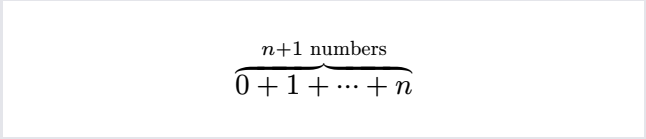
Default: none

The optional content below the brace.

/1.4 overbrace Element

A horizontal brace over content, with an optional annotation above.

```
$ overbrace(0 + 1 + dots.c + n, n + 1 "numbers") $
```



```
math.overbrace(
  content,
  none content,
) → content
```

/1.4.1 body content *Required* *Positional*

The content below the brace.

/1.4.2 annotation none or content *Positional* *Settable*

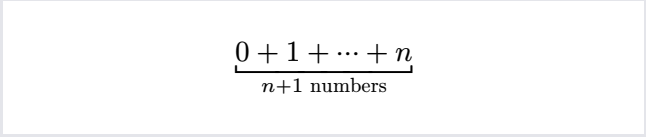
Default: none

The optional content above the brace.

/1.5 underbracket Element

A horizontal bracket under content, with an optional annotation below.

```
$ underbracket(0 + 1 + dots.c + n, n + 1 "numbers") $
```



```
math.underbracket(
  content,
  none content,
) → content
```

/1.5.1 body content *Required* *Positional*

The content above the bracket.

/1.5.2 annotation none or content *Positional* *Settable*

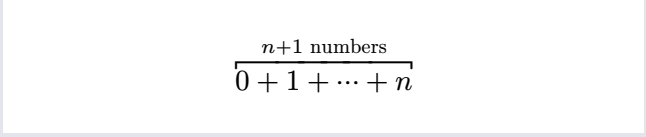
Default: none

The optional content below the bracket.

/1.6 overbracket Element

A horizontal bracket over content, with an optional annotation above.

```
$ overbracket(0 + 1 + dots.c + n, n + 1 "numbers") $
```



```
math.overbracket(
  content,
  none content,
) → content
```

/1.6.1 **body** `content` *Required* *Positional*

The content below the bracket.

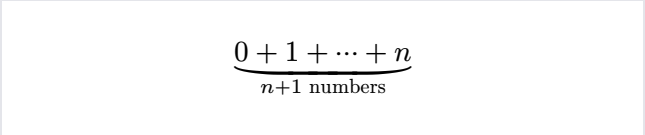
/1.6.2 **annotation** `none` or `content` *Positional* *Settable* Default: `none`

The optional content above the bracket.

/1.7 underparen Element

A horizontal parenthesis under content, with an optional annotation below.

```
$ underparen(0 + 1 + dots.c + n, n + 1 "numbers") $
```



```
math.underparen(
  content,
  none content,
) → content
```

/1.7.1 **body** `content` *Required* *Positional*

The content above the parenthesis.

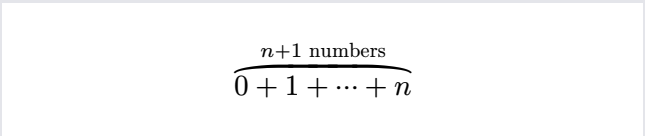
/1.7.2 **annotation** `none` or `content` *Positional* *Settable* Default: `none`

The optional content below the parenthesis.

/1.8 overparen Element

A horizontal parenthesis over content, with an optional annotation above.

```
$ overparen(0 + 1 + dots.c + n, n + 1 "numbers") $
```



```
math.overparen(
  content,
  none content,
) → content
```

/1.8.1 **body** `content` *Required* *Positional*

The content below the parenthesis.

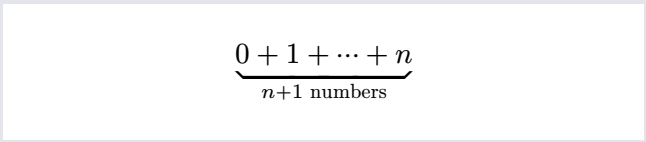
/1.8.2 **annotation** `none` or `content` *Positional* *Settable* Default: `none`

The optional content above the parenthesis.

/1.9 undershell Element

A horizontal tortoise shell bracket under content, with an optional annotation below.

```
$ undershell(0 + 1 + dots.c + n, n + 1 "numbers") $
```



```
math.undershell(
  content,
  none content,
) → content
```

/1.9.1 body content *Required* *Positional*

The content above the tortoise shell bracket.

/1.9.2 annotation none or content *Positional* *Settable*

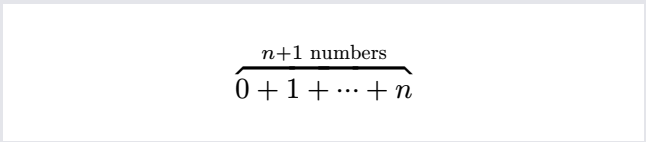
Default: none

The optional content below the tortoise shell bracket.

/1.10 overshell Element

A horizontal tortoise shell bracket over content, with an optional annotation above.

```
$ overshell(0 + 1 + dots.c + n, n + 1 "numbers") $
```



```
math.overshell(
  content,
  none content,
) → content
```

/1.10.1 body content *Required* *Positional*

The content below the tortoise shell bracket.

/1.10.2 annotation none or content *Positional* *Settable*

Default: none

The optional content above the tortoise shell bracket.

3.8.26 Variants

Alternate typefaces within formulas.

These functions are distinct from the `text`[3.7.12] function because math fonts contain multiple variants of each letter.

/1 Functions

/1.1 serif

Serif (roman) font style in math.

This is already the default.

```
math.serif(content) → content
```

/1.1.1 **body** `content` *Required* *Positional*

The content to style.

/1.2 sans

Sans-serif font style in math.

```
$ sans(A B C) $
```

$ sans(A B C) $

```
math.sans(content) → content
```

/1.2.1 **body** `content` *Required* *Positional*

The content to style.

/1.3 frak

Fraktur font style in math.

```
$ frak(P) $
```

$ frak(P) $

```
math.frak(content) → content
```

/1.3.1 **body** `content` *Required* *Positional*

The content to style.

/1.4 mono

Monospace font style in math.

```
$ mono(x + y = z) $
```

$ mono(x + y = z) $

```
math.mono(content) → content
```

/1.4.1 **body** `content` *Required* *Positional*

The content to style.

/1.5 bb

Blackboard bold (double-struck) font style in math.

For uppercase latin letters, blackboard bold is additionally available through symbols[3.9.2] of the form `NN` and `RR`.

```
$ bb(b) $
$ bb(N) = NN $
$ f: NN -> RR $
```

$$\mathbb{b}$$
$$\mathbb{N} = \mathbb{N}$$
$$f : \mathbb{N} \to \mathbb{R}$$

```
math.bb( content ) → content
```

/1.5.1 body `content` *Required* *Positional*

The content to style.

/1.6 cal

Calligraphic (chancery) font style in math.

```
Let $cal(P)$ be the set of ...
```

Let $\mathcal{P}$ be the set of ...

This is the default calligraphic/script style for most math fonts. See `scr`[3.8.26/1.7] for more on how to get the other style (roundhand).

```
math.cal( content ) → content
```

/1.6.1 body `content` *Required* *Positional*

The content to style.

/1.7 scr

Script (roundhand) font style in math.

```
$scr(L)$ is not the set of linear
maps $cal(L)$.
```

$\mathscr{L}$ is not the set of linear maps $\mathcal{L}$.

There are two ways that fonts can support differentiating `cal` and `scr`. The first is using Unicode variation sequences. This works out of the box in Typst, however only a few math fonts currently support this.

The other way is using font features[3.7.12/2.32]. For example, the roundhand style might be available in a font through the *stylistic set*[3.7.12/2.24] *1* (`ss01`) feature. To use it in Typst, you could then define your own version of `scr` like in the example below.

Example · Recreation using stylistic set 1

```
#let scr(it) = text(
  stylistic-set: 1,
  $cal(it)$,
)

We establish $cal(P) != scr(P)$.
```

We establish $\mathcal{P} \neq \mathscr{P}$.

```
math.scr( content ) → content
```

/1.7.1 **body** `content` *Required* *Positional*

The content to style.

3.8.27 `vec` Element

A column vector.

Content in the vector's elements can be aligned with the `align`[3.8.27/2.2] parameter, or the `&` symbol.

This function is for typesetting vector components. To typeset a symbol that represents a vector, `arrow`[3.8.9] and `bold`[3.8.23/1.3] are commonly used.

/1 Example

```
$ vec(a, b, c) dot vec(1, 2, 3)
    = a + 2b + 3c $
```

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = a + 2b + 3c$$

/2 Parameters

```
vec(
  delim: none str symbol array,
  align: alignment,
  gap: relative,
  ..content,
) → content
```

/2.1 `delim` `none` or `str` or `symbol` or `array` *Settable* Default: `("(", ")")`

The delimiter to use.

Can be a single character specifying the left delimiter, in which case the right delimiter is inferred. Otherwise, can be an array containing a left and a right delimiter.

```
#set math.vec(delim: "[")
$ vec(1, 2) $
```

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

/2.2 `align` `alignment` *Settable* Default: `center`

The horizontal alignment that each element should have.

```
#set math.vec(align: right)
$ vec(-1, 1, -1) $
```

$$\begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}$$

/2.3 `gap` `relative` *Settable* Default: `0% + 0.2em`

The gap between elements.

```
#set math.vec(gap: 1em)
$ vec(1, 2) $
```

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

/2.4 **children** `content` *Required* *Positional* *Variadic*

The elements of the vector.

3.9 Symbols

The General Symbols[3.9.2] and Emoji[3.9.3] modules give names to symbols and emoji to make them easy to insert with a normal keyboard. Alternatively, you can also always directly enter Unicode symbols into your text and formulas. In addition to the symbols listed below, math mode defines `dif` and `Dif`. These are not normal symbol values because they also affect spacing and font style.

You can define custom symbols with the constructor function of the `symbol`[3.5.28] type.

3.9.1 Shorthands

Shorthands are concise sequences of characters that evoke specific glyphs. Shorthands and other ways to produce symbols can be used interchangeably. You can use different sets of shorthands in math and markup mode. Some shorthands, like `~` for a non-breaking space produce non-printing symbols, which are indicated with gray placeholder text.

You can deactivate a shorthand's interpretation by escaping any of its characters. If you escape a single character in a shorthand, the remaining unescaped characters may form a different shorthand.

/1 Within Markup Mode

…	`...`
nbsp	`~`
–	`-`
–	`--`
—	`---`
shy	`-?`

/2 Within Math Mode

…	`...`
–	`-`
∗	`*`
∼	`~`
≠	`!=`
≔	`:=`
⩴	`::=`
≕	`=:`
≪	`<<`
⋘	`<<<`
≫	`>>`
⋙	`>>>`
≤	`<=`
≥	`>=`
→	`->`
⟶	`-->`
↦	`|->`
↣	`>->`
↠	`->>`
←	`<-`
⟵	`<--`
↢	`<-<`
↞	`<<-`
↔	`<->`
⟷	`<-->`
⇝	`~>`
⇜	`<~`
⇄	`<~~`
⇒	`=>`
⇨	`|=>`
⟹	`==>`
⟸	`<==`
⇔	`<=>`
⟺	`<==>`
⟦	`[|`
⟧	`|]`
‖	`||`

3.9.2 General Symbols

Named general symbols.

For example, `#sym.arrow` produces the → symbol. Within math[3.8], these symbols can be used without the `#sym.` prefix.

The d in an integral's dx can be written as `$dif x$`. Outside math formulas, `dif` can be accessed as `math.dif`.

𝔸	`AA`	𝕏	`XX`
A	`Alpha`	Ξ	`Xi`
𝔹	`BB`	𝕐	`YY`
B	`Beta`	ℤ	`ZZ`
ℂ	`CC`	Z	`Zeta`
X	`Chi`	´	`acute`
𝔻	`DD`	˝	`acute.double`
Δ	`Delta`	؋	`afghani`
F	`Digamma`	א	`aleph`
𝔼	`EE`	α	`alpha`
E	`Epsilon`	&	`amp`
H	`Eta`	⅋	`amp.inv`
𝔽	`FF`	∧	`and`
𝔾	`GG`	⋀	`and.big`
Γ	`Gamma`	⅄	`and.curly`
ℍ	`HH`	⟑	`and.dot`
𝕀	`II`	⩓	`and.double`
ℑ	`Im`	∠	`angle`
I	`Iota`	∡	`angle.acute`
𝕁	`JJ`	∢	`angle.arc`
𝕂	`KK`	⦛	`angle.arc.rev`
K	`Kappa`	⍼	`angle.azimuth`
𝕃	`LL`	⦠	`angle.obtuse`
Λ	`Lambda`	⦜	`angle.rev`
𝕄	`MM`	∟	`angle.right`
M	`Mu`	⌟	`angle.right.rev`
ℕ	`NN`	⊾	`angle.right.arc`
N	`Nu`	⦝	`angle.right.dot`
𝕆	`OO`	⦞	`angle.right.square`
Ω	`Omega`	⦣	`angle.s`
℧	`Omega.inv`	⦟	`angle.spatial`
O	`Omicron`	⦡	`angle.spheric`
ℙ	`PP`	⦢	`angle.spheric.rev`
Φ	`Phi`	⦤	`angle.spheric.t`
Π	`Pi`	Å	`angstrom`
Ψ	`Psi`	⍼	`angzarr`
ℚ	`QQ`	≈	`approx`
ℝ	`RR`	≊	`approx.eq`
ℜ	`Re`	≉	`approx.not`
P	`Rho`	≋	`approx.hat`
𝕊	`SS`	→	`arrow.r`
Ш	`Sha`	⟼	`arrow.r.long.bar`
Σ	`Sigma`	↦	`arrow.r.bar`
𝕋	`TT`	↳	`arrow.r.curve`
T	`Tau`	↻	`arrow.r.turn`
Θ	`Theta`	⇢	`arrow.r.dashed`
ϴ	`Theta.alt`	⇢	`arrow.r.dotted`
𝕌	`UU`	⇒	`arrow.r.double`
ϒ	`Upsilon`	⤇	`arrow.r.double.bar`
𝕍	`VV`	⟹	`arrow.r.double.long`
𝕎	`WW`	⟾	`arrow.r.double.long.bar`

⇏	arrow.r.double.not
⇻	arrow.r.double.struck
➡	arrow.r.filled
↪	arrow.r.hook
⟶	arrow.r.long
⟿	arrow.r.long.squiggly
↬	arrow.r.loop
↛	arrow.r.not
⇉	arrow.r.quad
⇝	arrow.r.squiggly
⇥	arrow.r.stop
⇨	arrow.r.stroked
↦	arrow.r.struck
⇸	arrow.r.dstruck
↣	arrow.r.tail
⤔	arrow.r.tail.struck
⤕	arrow.r.tail.dstruck
⇗	arrow.r.tilde
⇶	arrow.r.triple
↠	arrow.r.twohead
⤅	arrow.r.twohead.bar
⤀	arrow.r.twohead.struck
⤁	arrow.r.twohead.dstruck
⤖	arrow.r.twohead.tail
⤗	arrow.r.twohead.tail.struck
⤘	arrow.r.twohead.tail.dstruck
→	arrow.r.open
⇜	arrow.r.wave
←	arrow.l
↤	arrow.l.bar
↵	arrow.l.curve
↩	arrow.l.turn
⇠	arrow.l.dashed
⇠	arrow.l.dotted
⇐	arrow.l.double
⇤	arrow.l.double.bar
⟸	arrow.l.double.long
⟽	arrow.l.double.long.bar
⇍	arrow.l.double.not
⇷	arrow.l.double.struck
⬅	arrow.l.filled
↩	arrow.l.hook
⟵	arrow.l.long
⟻	arrow.l.long.bar
⟿	arrow.l.long.squiggly
↫	arrow.l.loop
↚	arrow.l.not
⇚	arrow.l.quad
↜	arrow.l.squiggly
⇤	arrow.l.stop
⇦	arrow.l.stroked
↤	arrow.l.struck
⇷	arrow.l.dstruck
↢	arrow.l.tail
⤙	arrow.l.tail.struck
⤚	arrow.l.tail.dstruck
⇜	arrow.l.tilde
⇚	arrow.l.triple
↞	arrow.l.twohead
⬵	arrow.l.twohead.bar
⤆	arrow.l.twohead.struck
⤇	arrow.l.twohead.dstruck
⬻	arrow.l.twohead.tail
⬺	arrow.l.twohead.tail.struck
⬼	arrow.l.twohead.tail.dstruck
←	arrow.l.open
↜	arrow.l.wave
↑	arrow.t
↥	arrow.t.bar
↱	arrow.t.curve
↺	arrow.t.turn
⇡	arrow.t.dashed
⇑	arrow.t.double
⬆	arrow.t.filled
⟰	arrow.t.quad
⤒	arrow.t.stop
⇧	arrow.t.stroked
↟	arrow.t.struck
⇞	arrow.t.dstruck
⟰	arrow.t.triple
↟	arrow.t.twohead
↓	arrow.b
↧	arrow.b.bar
↴	arrow.b.curve
↻	arrow.b.turn
⇣	arrow.b.dashed
⇓	arrow.b.double
⬇	arrow.b.filled
⟱	arrow.b.quad
⤓	arrow.b.stop
⇩	arrow.b.stroked
↡	arrow.b.struck
⇟	arrow.b.dstruck
⟱	arrow.b.triple
↡	arrow.b.twohead
↔	arrow.l.r
⇔	arrow.l.r.double
⟺	arrow.l.r.double.long
⇎	arrow.l.r.double.not
⇼	arrow.l.r.double.struck
⬌	arrow.l.r.filled
⟷	arrow.l.r.long
↮	arrow.l.r.not
⇔	arrow.l.r.stroked
↔	arrow.l.r.struck
⇹	arrow.l.r.dstruck
↔	arrow.l.r.open
↭	arrow.l.r.wave
↕	arrow.t.b
⇳	arrow.t.b.double
⬍	arrow.t.b.filled
⇕	arrow.t.b.stroked
↗	arrow.tr
⤤	arrow.tr.bar
⇗	arrow.tr.double
⬈	arrow.tr.filled

- ↗ arrow.tr.hook
- ⇗ arrow.tr.stroked
- ↘ arrow.br
- ⤵ arrow.br.bar
- ⇘ arrow.br.double
- ➘ arrow.br.filled
- ⤷ arrow.br.hook
- ⤳ arrow.br.stroked
- ↖ arrow.tl
- ⤴ arrow.tl.bar
- ⇖ arrow.tl.double
- ➖ arrow.tl.filled
- ↰ arrow.tl.hook
- ⤣ arrow.tl.stroked
- ↙ arrow.bl
- ⤶ arrow.bl.bar
- ⇙ arrow.bl.double
- ➙ arrow.bl.filled
- ↲ arrow.bl.hook
- ⤢ arrow.bl.stroked
- ⤡ arrow.tl.br
- ⤢ arrow.tr.bl
- ↺ arrow.ccw
- ↶ arrow.ccw.half
- ↻ arrow.cw
- ↷ arrow.cw.half
- ↯ arrow.zigzag
- ^ arrowhead.t
- ˅ arrowhead.b
- ⇉ arrows.rr
- ⇇ arrows.ll
- ⇈ arrows.tt
- ⇊ arrows.bb
- ⇆ arrows.lr
- ↹ arrows.lr.stop
- ⇄ arrows.rl
- ⇄ arrows.rl.long
- ⇅ arrows.tb
- ⇵ arrows.bt
- ⇉ arrows.rrr
- ⇇ arrows.lll
- ∗ ast.op
- ⊛ ast.op.o
- * ast.basic
- ⁎ ast.low
- ⁑ ast.double
- ⁂ ast.triple
- ⊛ ast.square
- ≍ asymp
- ≭ asymp.not
- @ at
- \ backslash
- ⦸ backslash.o
- ⍀ backslash.not
- ⟅ bag.l
- ⟆ bag.r
- ฿ baht
- ☐ ballot
- ☒ ballot.cross
- ☑ ballot.check
- 🗹 ballot.check.heavy
- | bar.v
- ‖ bar.v.double
- ⦀ bar.v.triple
- ¦ bar.v.broken
- ⦶ bar.v.o
- — bar.h
- ∵ because
- β beta
- ϐ beta.alt
- ב beth
- ฿ bitcoin
- ⊥ bot
- ⋈ bowtie.stroked
- ⨝ bowtie.stroked.big
- ⋈ bowtie.stroked.big.l
- ⋈ bowtie.stroked.big.r
- ⋈ bowtie.stroked.big.l.r
- ⧓ bowtie.filled
- ⧑ bowtie.filled.l
- ⧒ bowtie.filled.r
- { brace.l
- ⦃ brace.l.stroked
- } brace.r
- ⦄ brace.r.stroked
- ⏞ brace.t
- ⏟ brace.b
- [bracket.l
- ⎡ bracket.l.tick.t
- ⎣ bracket.l.tick.b
- ⟦ bracket.l.stroked
-] bracket.r
- ⎤ bracket.r.tick.t
- ⎦ bracket.r.tick.b
- ⟧ bracket.r.stroked
- ⎴ bracket.t
- ⎵ bracket.b
- ˘ breve
- ▪ bullet
- • bullet.op
- ⦿ bullet.o
- ○ bullet.stroked
- ◎ bullet.stroked.o
- ◘ bullet.hole
- ⋅ bullet.hyph
- ‣ bullet.tri
- ◖ bullet.l
- ◗ bullet.r
- ^ caret
- ˇ caron
- ㏄ cc
- ㏒ cc.by
- ㏑ cc.nc
- ⊜ cc.nd
- ㏓ cc.public

⭮	cc.sa
⓪	cc.zero
₵	cedi
⌈	ceil.l
⌉	ceil.r
¢	cent
✓	checkmark
✓	checkmark.light
✔	checkmark.heavy
⟨	chevron.l
⟨	chevron.l.curly
⟨	chevron.l.dot
⟨	chevron.l.closed
⟪	chevron.l.double
⟩	chevron.r
⟩	chevron.r.curly
⟩	chevron.r.dot
⟩	chevron.r.closed
⟫	chevron.r.double
χ	chi
○	circle.stroked
∘	circle.stroked.tiny
○	circle.stroked.small
◯	circle.stroked.big
●	circle.filled
•	circle.filled.tiny
●	circle.filled.small
⬤	circle.filled.big
◌	circle.dotted
℅	co
:	colon
₡	colon.currency
∷	colon.double
⁝	colon.tri
⁝	colon.tri.op
≔	colon.eq
⩴	colon.double.eq
,	comma
‚	comma.inv
‚	comma.rev
∁	complement
∘	compose
⊚	compose.o
␆	control.ack
␇	control.bel
␈	control.bs
␘	control.can
␍	control.cr
␑	control.dc.one
␒	control.dc.two
␓	control.dc.three
␔	control.dc.four
␡	control.del
␐	control.dle
␙	control.em
␅	control.enq
␄	control.eot
␛	control.esc
␗	control.etb
␃	control.etx
␌	control.ff
␜	control.fs
␝	control.gs
␉	control.ht
␊	control.lf
␕	control.nak
␤	control.nl
␀	control.nul
␞	control.rs
␏	control.si
␎	control.so
␁	control.soh
␠	control.sp
␂	control.stx
␚	control.sub
␖	control.syn
␟	control.us
␋	control.vt
∗	convolve
⊛	convolve.o
℗	copyleft
©	copyright
℗	copyright.sound
⌜	corner.l.t
⌞	corner.l.b
⌝	corner.r.t
⌟	corner.r.b
╳	crossmark
✘	crossmark.heavy
¤	currency
†	dagger
‡	dagger.double
⧺	dagger.triple
↿	dagger.l
↾	dagger.r
⇂	dagger.inv
ד	daleth
–	dash.en
—	dash.em
⸺	dash.em.two
⸻	dash.em.three
‒	dash.fig
–:	dash.colon
⊖	dash.o
~	dash.wave
~~	dash.wave.double
°	degree
δ	delta
¨	diaer

⌀	diameter
◇	diamond.stroked
⋄	diamond.stroked.small
◇	diamond.stroked.medium
⟐	diamond.stroked.dot
◆	diamond.filled
◆	diamond.filled.medium
◆	diamond.filled.small
⚅	die.six
⚄	die.five
⚃	die.four
⚂	die.three
⚁	die.two
⚀	die.one
ϝ	digamma
÷	div
⨸	div.o
⟌	div.slanted.o
∣	divides
∤	divides.not
⫮	divides.not.rev
⫶	divides.struck
$	dollar
đ	dong
□	dorome
⋅	dot.op
.	dot.basic
·	dot.c
⊙	dot.o
⨀	dot.o.big
⊡	dot.square
¨	dot.double
⃛	dot.triple
⃜	dot.quad
ɪ	dotless.i
ȷ	dotless.j
⋯	dots.h.c
…	dots.h
⋮	dots.v
⋱	dots.down
⋰	dots.up
ϯ	dram
□	earth
ð	earth.alt
ℓ	ell
⬭	ellipse.stroked.h
⬯	ellipse.stroked.v
⬬	ellipse.filled.h
⬮	ellipse.filled.v
∅	emptyset
∅	emptyset.zero
⃗∅	emptyset.arrow.r
⃖∅	emptyset.arrow.l
⃥∅	emptyset.bar
⃠∅	emptyset.circle
⦸	emptyset.rev
ε	epsilon
ϵ	epsilon.alt
ɜ	epsilon.alt.rev
=	eq
≛	eq.ast
≛	eq.star
⊜	eq.o
⩵	eq.colon
≐	eq.dot
≒	eq.dots
≒	eq.dots.down
≒	eq.dots.up
≝	eq.def
≜	eq.delta
≚	eq.equi
≙	eq.est
⋝	eq.gt
⋜	eq.lt
≞	eq.m
≠	eq.not
⋞	eq.prec
≟	eq.quest
⋟	eq.succ
≡	eq.triple
≢	eq.triple.not
≣	eq.quad
≡	equiv
≢	equiv.not
⫠	errorbar.square.stroked
⫠	errorbar.square.filled
⫡	errorbar.diamond.stroked
⫡	errorbar.diamond.filled
⫯	errorbar.circle.stroked
⫯	errorbar.circle.filled
η	eta
€	euro
!	excl
‼	excl.double
¡	excl.inv
⁉	excl.quest
∃	exists
∄	exists.not
⦚	fence.l
⧙	fence.l.double
⦚	fence.r
⧙	fence.r.double
⦙	fence.dotted
♭	flat
▱	flat.t
▱	flat.b
▱	flat.double
▱	flat.quarter
⌊	floor.l
⌋	floor.r
❦	floral
❧	floral.l
❧	floral.r
∀	forall
⊩	forces
⊮	forces.not

⌒ frown
ℽ gamma
♀ gender.female
⚢ gender.female.double
⚤ gender.female.male
⚥ gender.intersex
♂ gender.male
⚣ gender.male.double
⚤ gender.male.female
⚦ gender.male.stroke
⚨ gender.male.stroke.t
⚩ gender.male.stroke.r
⚲ gender.neuter
⚧ gender.trans
ℷ gimel
∇ gradient
` grave
> gt
⦸ gt.o
⩾ gt.dot
⩼ gt.quest
≳ gt.approx
◁ gt.arc
⋟ gt.arc.eq
▷ gt.closed
⊵ gt.closed.eq
⋭ gt.closed.eq.not
⋫ gt.closed.not
≫ gt.double
⪢ gt.double.nested
≥ gt.eq
⩾ gt.eq.slant
⋛ gt.eq.lt
≱ gt.eq.not
≧ gt.equiv
≷ gt.lt
≯ gt.lt.not
≩ gt.neq
≳ gt.napprox
≩ gt.nequiv
>/ gt.not
≳ gt.ntilde
≳ gt.tilde
≯ gt.tilde.not
▷ ~~gt.tri~~
⊵ ~~gt.tri.eq~~
⋭ ~~gt.tri.eq.not~~
⋫ ~~gt.tri.not~~
⋙ gt.triple
⫸ gt.triple.nested
₲ guarani
⇀ harpoon.rt
⥓ harpoon.rt.bar
⇀| harpoon.rt.stop
⇁ harpoon.rb
⥟ harpoon.rb.bar
⇁| harpoon.rb.stop
↼ harpoon.lt
↽| harpoon.lt.bar
|↼ harpoon.lt.stop
↼ harpoon.lb
↽| harpoon.lb.bar
|↼ harpoon.lb.stop
↿ harpoon.tl
⥠ harpoon.tl.bar
|↿ harpoon.tl.stop
↾ harpoon.tr
⥤ harpoon.tr.bar
|↾ harpoon.tr.stop
⇂ harpoon.bl
⥡ harpoon.bl.bar
⇂| harpoon.bl.stop
⇃ harpoon.br
⥥ harpoon.br.bar
⇃| harpoon.br.stop
⥎ harpoon.lt.rt
⥋ harpoon.lb.rb
⇄ harpoon.lb.rt
⥊ harpoon.lt.rb
⇃ harpoon.tl.bl
⇂ harpoon.tr.br
⇃ harpoon.tl.br
⇂ harpoon.tr.bl
⇉ harpoons.rtrb
⥯ harpoons.blbr
⥮ harpoons.bltr
⥋ harpoons.lbrb
⇇ harpoons.ltlb
⇋ harpoons.ltrb
⥎ harpoons.ltrt
⇄ harpoons.rblb
⇌ harpoons.rtlb
⥤ harpoons.rtlt
⥯ harpoons.tlbr
⥮ harpoons.tltr
hash
^ hat
⬡ hexa.stroked
⬢ hexa.filled
⧖ hourglass.stroked
⧗ hourglass.filled
₴ hryvnia
- hyph
- hyph.minus
- hyph.nobreak
· hyph.point
shy hyph.soft
⊷ image
∈ in
∉ in.not
∋ in.rev
∌ in.rev.not
∍ in.rev.small
∈ in.small
∞ infinity
⧜ infinity.bar

∞	infinity.incomplete
⧜	infinity.tie
∫	integral
⨍	integral.arrow.hook
⨑	integral.ccw
∮	integral.cont
∳	integral.cont.ccw
∲	integral.cont.cw
⨎	integral.cw
⨏	integral.dash
⨐	integral.dash.double
∬	integral.double
⨌	integral.quad
⨙	integral.inter
⨏	integral.slash
⨑	integral.square
⨋	integral.surf
⨘	integral.times
∭	integral.triple
⨊	integral.union
⨋	integral.vol
∩	inter
⋒	inter.serif
⩕	inter.and
⋂	inter.big
⩀	inter.dot
⋒	inter.double
⊓	inter.sq
⊓	inter.sq.serif
⨅	inter.sq.big
⩎	inter.sq.double
⫴	interleave
⫼	interleave.big
⫳	interleave.struck
⁉	interrobang
⸘	interrobang.inv
ι	iota
ɩ	iota.inv
⋈	~~join~~
⋉	~~join.r~~
⋊	~~join.l~~
⟕	~~join.l.r~~
♃	jupiter
ϰ	kappa
𝜘	kappa.alt
₭	kip
λ	lambda
Δ	laplace
₾	lari
⩾	lat
≥	lat.eq
₺	lira
◊	lozenge.stroked
⋄	lozenge.stroked.small
◊	lozenge.stroked.medium
◆	lozenge.filled
⧫	lozenge.filled.small
♦	lozenge.filled.medium
lrm	lrm
<	lt
⊛	lt.o
⋖	lt.dot
⩻	lt.quest
⪅	lt.approx
◁	lt.arc
◁	lt.arc.eq
◁	lt.closed
⊴	lt.closed.eq
⋬	lt.closed.eq.not
⋪	lt.closed.not
≪	lt.double
⪡	lt.double.nested
≤	lt.eq
⩽	lt.eq.slant
⪋	lt.eq.gt
≰	lt.eq.not
≦	lt.equiv
≶	lt.gt
≸	lt.gt.not
≨	lt.neq
⪉	lt.napprox
≨	lt.nequiv
</	lt.not
⪅	lt.ntilde
≲	lt.tilde
≴	lt.tilde.not
◁	~~lt.tri~~
⊴	~~lt.tri.eq~~
⋬	~~lt.tri.eq.not~~
⋪	~~lt.tri.not~~
⋘	lt.triple
⪡	lt.triple.nested
¯	macron
✠	maltese
₼	manat
⟼	mapsfrom
⟻	mapsfrom.long
↦	mapsto
⟼	mapsto.long
♂	mars
☿	mercury
−	minus
⊖	minus.o
∸	minus.dot
∓	minus.plus
⊟	minus.square
≂	minus.tilde
⨺	minus.triangle
⫟	miny
⊨	models
μ	mu
⊸	multimap
⧟	multimap.double
⎰	mustache.l
⎱	mustache.r
∇	nabla

₦	naira
♮	natural
♮	natural.t
♮	natural.b
♆	neptune
♆	neptune.alt
¬	not
𝅘𝅥	note.up
𝅘𝅥	note.down
𝅝	note.whole
𝅗𝅥	note.half
♩	note.quarter
♩	note.quarter.alt
♪	note.eighth
♪	note.eighth.alt
♫	note.eighth.beamed
𝅘𝅥𝅯	note.sixteenth
♫	note.sixteenth.beamed
𝅘𝅥	note.grace
𝅘𝅥	note.grace.slash
∅	nothing
∅	nothing.zero
∅⃗	nothing.arrow.r
∅⃖	nothing.arrow.l
∅̄	nothing.bar
∅̊	nothing.circle
⦸	nothing.rev
ν	nu
№	numero
ω	omega
ο	omicron
∞	oo
∨	or
⋁	or.big
⋎	or.curly
⩒	or.dot
⩔	or.double
⊶	original
∥	parallel
⫲	parallel.struck
⦶	parallel.o
⋕	parallel.eq
⩕	parallel.equiv
∦	parallel.not
⫽	parallel.slanted.eq
⫽̃	parallel.slanted.eq.tilde
⫻	parallel.slanted.equiv
⫴	parallel.tilde
▱	parallelogram.stroked
▰	parallelogram.filled
(	paren.l
⟮	paren.l.flat
⦇	paren.l.closed
⦇	paren.l.stroked
)	paren.r
⟯	paren.r.flat
⦈	paren.r.closed
⦈	paren.r.stroked
⏜	paren.t
⏝	paren.b
∂	partial
$	pataca
℘	pee
⬠	penta.stroked
⬟	penta.filled
%	percent
‰	permille
‱	permyriad
⊥	perp
⦫	perp.o
$	peso
₱	peso.philippine
φ	phi
ϕ	phi.alt
π	pi
ϖ	pi.alt
¶	pilcrow
⁋	pilcrow.rev
ℏ	planck
+	plus
⊕	plus.o
⨁	plus.o.l
⨁	plus.o.r
⨁	plus.o.arrow
⨁	plus.o.big
∔	plus.dot
⧺	plus.double
±	plus.minus
⊞	plus.square
⧋	plus.triangle
⧻	plus.triple
⨤	plus.hat
£	pound
⏻	power.standby
⏽	power.on
⏾	power.off
⏼	power.on.off
⏾	power.sleep
≺	prec
≾	prec.approx
⊀	prec.curly.eq
⋠	prec.curly.eq.not
⊀	prec.double
⪯	prec.eq
⪯̸	prec.equiv
⪹	prec.napprox
⪵	prec.neq
⪱	prec.nequiv
⊀	prec.not
⪷	prec.ntilde
≾̸	prec.tilde
′	prime
‵	prime.rev
″	prime.double
‶	prime.double.rev
‴	prime.triple

‴ prime.triple.rev
⁗ prime.quad
∏ product
∐ product.co
∝ prop
ψ psi
■ qed
? quest
⁇ quest.double
⁈ quest.excl
¿ quest.inv
" quote.double
' quote.single
“ quote.l.double
‘ quote.l.single
” quote.r.double
’ quote.r.single
« quote.chevron.l.double
‹ quote.chevron.l.single
» quote.chevron.r.double
› quote.chevron.r.single
‟ quote.high.double
‛ quote.high.single
„ quote.low.double
‚ quote.low.single
∶ ratio
▭ rect.stroked.h
▯ rect.stroked.v
▬ rect.filled.h
▮ rect.filled.v
※ refmark
𝅝 rest.whole
𝅗𝅥 rest.multiple
𝄺 rest.multiple.measure
𝄼 rest.half
𝄽 rest.quarter
𝄾 rest.eighth
𝄿 rest.sixteenth
ρ rho
ϱ rho.alt
﷼ riel
₱ riyal
rlm rlm
₽ ruble
₹ rupee.indian
₨ rupee.generic
𑿝 rupee.tamil
𞋿 rupee.wancho
♄ saturn
§ section
; semi
⁏ semi.inv
; semi.rev
Ш sha
♯ sharp
𝆯 sharp.t
𝆰 sharp.b
𝆱 sharp.double
𝆲 sharp.quarter
₪ shekel
⟨ shell.l
⟪ shell.l.stroked
❨ shell.l.filled
⟩ shell.r
⟫ shell.r.stroked
❩ shell.r.filled
⏠ shell.t
⏡ shell.b
σ sigma
ς sigma.alt
/ slash
⊘ slash.o
// slash.double
/// slash.triple
╱ slash.big
※ smash
⌣ smile
⩽ smt
≦ smt.eq
₱ som
␣ space
nbsp space.nobreak
nnbsp space.nobreak.narrow
ensp space.en
emsp space.quad
⅓emsp space.third
¼emsp space.quarter
⅙emsp space.sixth
mmsp space.med
numsp space.fig
puncsp space.punct
thinsp space.thin
hairsp space.hair
␣ spacebar
☐ square.stroked
▫ square.stroked.tiny
□ square.stroked.small
□ square.stroked.medium
☐ square.stroked.big
⬚ square.stroked.dotted
▢ square.stroked.rounded
■ square.filled
▪ square.filled.tiny
■ square.filled.small
■ square.filled.medium
■ square.filled.big
⋆ star.op
☆ star.stroked
★ star.filled
⊂ subset
⫋ subset.approx
⊏ subset.closed
⊑ subset.closed.eq
⊍ subset.dot
⫅ subset.double
⊆ subset.eq

⫃ subset.eq.dot
⊈ subset.eq.not
⊑ subset.eq.sq
⋢ subset.eq.sq.not
⫅ subset.equiv
⊊ subset.neq
⫋ subset.nequiv
⊄ subset.not
⫀ subset.plus
⊏ subset.sq
⋤ subset.sq.neq
⫉ subset.tilde
⫃ subset.times
≻ succ
⪸ succ.approx
≽ succ.curly.eq
⋡ succ.curly.eq.not
⪼ succ.double
≿ succ.eq
⪴ succ.equiv
⪺ succ.napprox
⪶ succ.neq
⪴̸ succ.nequiv
⊁ succ.not
⪹ succ.ntilde
≿ succ.tilde
♣ suit.club.filled
♧ suit.club.stroked
♦ suit.diamond.filled
♢ suit.diamond.stroked
♥ suit.heart.filled
♡ suit.heart.stroked
♠ suit.spade.filled
♤ suit.spade.stroked
Σ sum
∳ sum.integral
⊙ sun
⊃ supset
⪈ supset.approx
⟃ supset.closed
⊇ supset.closed.eq
⪾ supset.dot
⫘ supset.double
⊇ supset.eq
⫄ supset.eq.dot
⊉ supset.eq.not
⊒ supset.eq.sq
⋣ supset.eq.sq.not
⫆ supset.equiv
⊋ supset.neq
⫌ supset.nequiv
⊅ supset.not
⫀ supset.plus
⊐ supset.sq
⋥ supset.sq.neq
⫈ supset.tilde
⫂ supset.times
⊢ tack.r
⊬ tack.r.not
⟝ tack.r.long
⊦ tack.r.short
~~⊨ tack.r.double~~
⊨ tack.rr
~~⊭ tack.r.double.not~~
⊭ tack.rr.not
⊩ tack.rrr
⊣ tack.l
⟞ tack.l.long
⫞ tack.l.short
~~⫤ tack.l.double~~
⫤ tack.ll
⊥ tack.t
⟘ tack.t.big
~~⫫ tack.t.double~~
⫫ tack.tt
⫠ tack.t.short
⊤ tack.b
⟙ tack.b.big
~~⫪ tack.b.double~~
⫪ tack.bb
⫟ tack.b.short
⫦ tack.l.r
□ taka
□ taman
τ tau
T̄ tenge
∴ therefore
θ theta
ϑ theta.alt
∼ tilde.op
~ tilde.basic
⩪ tilde.dot
≃ tilde.eq
≄ tilde.eq.not
≲ tilde.eq.rev
≅ tilde.equiv
≇ tilde.equiv.not
≆ tilde.nequiv
≁ tilde.not
∽ tilde.rev
⋍ tilde.rev.equiv
≋ tilde.triple
× times
⨉ times.big
⊗ times.o
⟕ times.o.l
⟖ times.o.r
⊛ times.o.hat
⨂ times.o.big
⋇ times.div
⋋ times.three.l
⋌ times.three.r
⋉ times.l
⋊ times.r
⊠ times.square
⟑ times.triangle

⨦	tiny
₮	togrog
⊤	top
™	trademark
®	trademark.registered
℠	trademark.service
△	triangle.stroked.t
▽	triangle.stroked.b
▷	triangle.stroked.r
◁	triangle.stroked.l
◺	triangle.stroked.bl
◿	triangle.stroked.br
◸	triangle.stroked.tl
◹	triangle.stroked.tr
▵	triangle.stroked.small.t
▿	triangle.stroked.small.b
▹	triangle.stroked.small.r
◃	triangle.stroked.small.l
🛆	triangle.stroked.rounded
⟁	triangle.stroked.nested
◬	triangle.stroked.dot
▲	triangle.filled.t
▼	triangle.filled.b
▶	triangle.filled.r
◀	triangle.filled.l
◣	triangle.filled.bl
◢	triangle.filled.br
◤	triangle.filled.tl
◥	triangle.filled.tr
▲	triangle.filled.small.t
▼	triangle.filled.small.b
▶	triangle.filled.small.r
◀	triangle.filled.small.l
_	underscore
∪	union
⋓	union.serif
⊎	union.arrow
⋃	union.big
⊍	union.dot
⨃	union.dot.big
⋓	union.double
⊎	union.minus
⩒	union.or
⊎	union.plus
⨄	union.plus.big
⊔	union.sq
⊔	union.sq.serif
⨆	union.sq.big
⩏	union.sq.double
ϒ	upsilon
⛢	uranus
♅	uranus.alt
♀	venus
∖	without
wjoin	wj
₩	won
≀	wreath
ξ	xi
⊕	xor
⨁	xor.big
¥	yen
¥	yuan
0	zero
0	zero.slashed
ζ	zeta
zwj	zwj
zwnj	zwnj
zwsp	zws

3.9.3 Emoji

Named emojis.

For example, `#emoji.face` produces the 😀 emoji. If you frequently use certain emojis, you can also import them from the `emoji` module (`#import emoji: face`) to use them without the `emoji.` prefix.

- 🆎 ABCD
- 🅰 a
- 🆎 ab
- 🧮 abacus
- 🔤 abc
- 🔡 abcd
- 🪗 accordion
- ⚕ aesculapius
- ✈ airplane
- 🛬 airplane.landing
- 🛩 airplane.small
- 🛫 airplane.takeoff
- ⚗ alembic
- 👽 alien
- 👾 alien.monster
- 🚑 ambulance
- 🏺 amphora
- ⚓ anchor
- 💢 anger
- 🐜 ant
- 🍏 apple.green
- 🍎 apple.red
- 🦾 arm.mech
- 💪 arm.muscle
- 🤳 arm.selfie
- ➡ arrow.r.filled
- ↪ arrow.r.hook
- SOON arrow.r.soon
- ⬅ arrow.l.filled
- ↩ arrow.l.hook
- BACK arrow.l.back
- END arrow.l.end
- ⬆ arrow.t.filled
- ⤴ arrow.t.curve
- TOP arrow.t.top
- ⬇ arrow.b.filled
- ⤵ arrow.b.curve
- ↔ arrow.l.r
- ON! arrow.l.r.on
- ↕ arrow.t.b
- ↙ arrow.bl
- ↘ arrow.br
- ↖ arrow.tl
- ↗ arrow.tr
- 🔄 arrows.cycle
- * ast
- ✳ ast.box
- 🏧 atm
- ⚛ atom
- 🍆 aubergine
- 🥑 avocado
- 🪓 axe
- 🅱 b
- 👶 baby
- 👼 baby.angel
- 🚸 baby.box
- 🍼 babybottle
- 🎒 backpack
- 🥓 bacon
- 🦡 badger
- 🏸 badminton
- 🥯 bagel
- 🛄 baggageclaim
- 🥖 baguette
- 🎈 balloon
- ☑ ballot.check
- 🗳 ballotbox
- 🍌 banana
- 🪕 banjo
- 🏦 bank
- 💈 barberpole
- ⚾ baseball
- 🧢 basecap
- 🧺 basket
- ⛹ basketball
- 🏀 basketball.ball
- 🦇 bat
- 🛁 bathtub
- 🛁 bathtub.foam
- 🔋 battery
- 🪫 battery.low
- 🏝 beach.palm
- 🏖 beach.umbrella
- 📿 beads
- 🫘 beans
- 🐻 bear
- 🦫 beaver
- 🛏 bed
- 🛌 bed.person
- 🐝 bee
- 🍺 beer
- 🍻 beer.clink
- ▫ beet
- 🪲 beetle
- 🐞 beetle.lady
- 🔔 bell
- 🛎 bell.ding
- 🔕 bell.not

bento
bicyclist
bicyclist.mountain
bigfoot
bike
bike.not
bikini
billiards
bin
biohazard
bird
bison
blood
blouse
blowfish
blueberries
boar
boat.sail
boat.row
boat.motor
boat.speed
boat.canoe
bolt
bomb
bone
book.red
book.blue
book.green
book.orange
book.spiral
book.open
bookmark
books
boomerang
bordercontrol
bouquet
bow
bowl.spoon
bowl.steam
bowling
boxing
boy
brain
bread
brick
bride
bridge.fog
bridge.night
briefcase
briefs
brightness.high
brightness.low
broccoli
broom
brush
bubble.speech.r
bubble.speech.l
bubble.thought
bubble.anger.r
bubbles
bubbletea
bucket
buffalo.water
bug
builder
burger
burrito
bus
bus.front
bus.small
bus.stop
bus.trolley
butter
butterfly
button
button.alt
button.radio
cabinet.file
cablecar
cablecar.small
cactus
cake
cake.fish
cake.moon
cake.slice
calendar
calendar.spiral
calendar.tearoff
camel
camel.dromedar
camera
camera.flash
camera.movie
camera.movie.box
camera.video
camping
can
candle
candy
cane
car
car.front
car.pickup
car.police
car.police.front
car.racing
car.rickshaw
car.suv

- card.credit
- card.id
- cardindex
- carrot
- cart
- cassette
- castle.eu
- castle.jp
- cat
- cat.face
- cat.face.angry
- cat.face.cry
- cat.face.heart
- cat.face.joy
- cat.face.kiss
- cat.face.laugh
- cat.face.shock
- cat.face.smile
- cat.face.smirk
- chain
- chains
- chair
- champagne
- chart.bar
- chart.up
- chart.down
- chart.yen.up
- checkmark.heavy
- checkmark.box
- cheese
- cherries
- chess
- chestnut
- chicken
- chicken.baby
- chicken.baby.egg
- chicken.baby.head
- chicken.leg
- chicken.male
- child
- chipmunk
- chocolate
- chopsticks
- church
- church.love
- cigarette
- cigarette.not
- circle.black
- circle.blue
- circle.brown
- circle.green
- circle.orange
- circle.purple
- circle.white
- circle.red
- circle.yellow
- circle.stroked
- circus
- city
- city.dusk
- city.night
- city.sunset
- cl
- clamp
- clapperboard
- climbing
- clip
- clipboard
- clips
- clock.one
- clock.one.thirty
- clock.two
- clock.two.thirty
- clock.three
- clock.three.thirty
- clock.four
- clock.four.thirty
- clock.five
- clock.five.thirty
- clock.six
- clock.six.thirty
- clock.seven
- clock.seven.thirty
- clock.eight
- clock.eight.thirty
- clock.nine
- clock.nine.thirty
- clock.ten
- clock.ten.thirty
- clock.eleven
- clock.eleven.thirty
- clock.twelve
- clock.twelve.thirty
- clock.alarm
- clock.old
- clock.timer
- cloud
- cloud.dust
- cloud.rain
- cloud.snow
- cloud.storm
- cloud.sun
- cloud.sun.hidden
- cloud.sun.rain
- cloud.thunder
- coat
- coat.lab
- cockroach

- cocktail.martini
- cocktail.tropical
- coconut
- coffee
- coffin
- coin
- comet
- compass
- computer
- computermouse
- confetti
- construction
- controller
- cookie
- cookie.fortune
- cooking
- cool
- copyright
- coral
- corn
- couch
- couple
- cow
- cow.face
- crab
- crane
- crayon
- cricket
- cricketbat
- crocodile
- croissant
- crossmark
- crossmark.box
- crown
- crutch
- crystal
- cucumber
- cup.straw
- cupcake
- curling
- curry
- custard
- customs
- cutlery
- cyclone
- dancing.ballet
- dancing.man
- dancing.woman
- dancing.bunny
- dancing.bunny.men
- dancing.bunny.women
- darts
- dash.wave.double
- deer
- desert
- detective
- diamond.blue
- diamond.blue.small
- diamond.orange
- diamond.orange.small
- diamond.dot
- die
- dino.pod
- dino.rex
- disc.cd
- disc.dvd
- disc.mini
- discoball
- diving
- dodo
- dog
- dog.face
- dog.guide
- dog.poodle
- dollar
- dolphin
- donkey
- donut
- door
- dove.peace
- dragon
- dragon.face
- dress
- dress.kimono
- dress.sari
- drop
- drops
- drum
- drum.big
- duck
- dumpling
- eagle
- ear
- ear.aid
- egg
- eighteen.not
- elephant
- elevator
- elf
- email
- excl
- excl.white
- excl.double
- excl.quest
- explosion
- extinguisher
- eye
- eyes

- face.grin
- face.angry
- face.angry.red
- face.anguish
- face.astonish
- face.bandage
- face.beam
- face.blank
- face.clown
- face.cold
- face.concern
- face.cool
- face.cover
- face.cowboy
- face.cry
- face.devil.smile
- face.devil.frown
- face.diagonal
- face.disguise
- face.distorted
- face.distress
- face.dizzy
- face.dotted
- face.down
- face.down.sweat
- face.drool
- face.explode
- face.eyeroll
- face.friendly
- face.fear
- face.fear.sweat
- face.fever
- face.flush
- face.frown
- face.frown.slight
- face.frust
- face.goofy
- face.halo
- face.happy
- face.heart
- face.hearts
- face.heat
- face.hug
- face.inv
- face.joy
- face.kiss
- face.kiss.smile
- face.kiss.heart
- face.kiss.blush
- face.lick
- face.lie
- face.mask
- face.meh
- face.melt
- face.money
- face.monocle
- face.nausea
- face.nerd
- face.neutral
- face.open
- face.party
- face.peek
- face.plead
- face.relief
- face.rofl
- face.sad
- face.salute
- face.shaking
- face.shock
- face.shush
- face.skeptic
- face.sleep
- face.sleepy
- face.smile
- face.smile.slight
- face.smile.sweat
- face.smile.tear
- face.smirk
- face.sneeze
- face.speak.not
- face.squint
- face.stars
- face.straight
- face.suffer
- face.surprise
- face.symbols
- face.tear
- face.tear.relief
- face.tear.withheld
- face.teeth
- face.think
- face.tired
- face.tongue
- face.tongue.squint
- face.tongue.wink
- face.triumph
- face.unhappy
- face.vomit
- face.weary
- face.wink
- face.woozy
- face.worry
- face.wow
- face.yawn
- face.zip
- factory
- fairy
- faith.christ

- faith.dharma
- faith.khanda
- faith.islam
- faith.judaism
- faith.menorah
- faith.om
- faith.orthodox
- faith.peace
- faith.star.dot
- faith.worship
- faith.yinyang
- falafel
- family
- fax
- feather
- feeding.breast
- fencing
- ferriswheel
- fightcloud
- filebox
- filedividers
- film
- finger.r
- finger.l
- finger.t
- finger.t.alt
- finger.b
- finger.front
- finger.m
- fingerprint
- fingers.cross
- fingers.pinch
- fingers.snap
- fire
- firecracker
- fireengine
- fireworks
- fish
- fish.tropical
- fishing
- fist.front
- fist.r
- fist.l
- fist.raised
- flag.black
- flag.white
- flag.goal
- flag.golf
- flag.red
- flags.jp.crossed
- flamingo
- flashlight
- flatbread
- fleur
- floppy
- flower.hibiscus
- flower.hyacinth
- flower.lotus
- flower.pink
- flower.rose
- flower.sun
- flower.tulip
- flower.white
- flower.wilted
- flower.yellow
- flute
- fly
- fog
- folder
- folder.open
- fondue
- foot
- football
- football.am
- forex
- fountain
- fox
- free
- fries
- frisbee
- frog.face
- fuelpump
- gachi
- garlic
- gear
- gem
- genie
- ghost
- ginger
- giraffe
- girl
- glass.clink
- glass.milk
- glass.pour
- glass.tumbler
- glasses
- glasses.sun
- globe.am
- globe.as.au
- globe.eu.af
- globe.meridian
- gloves
- go
- goal
- goat
- goggles
- golfing
- goose

- gorilla
- grapes
- guard.man
- guitar
- gymnastics
- haircut
- hairpick
- hammer
- hammer.pick
- hammer.wrench
- hamsa
- hamster.face
- hand.raised
- hand.raised.alt
- hand.r
- hand.l
- hand.t
- hand.b
- hand.ok
- hand.call
- hand.love
- hand.part
- hand.peace
- hand.pinch
- hand.pushing.l
- hand.pushing.r
- hand.rock
- hand.splay
- hand.wave
- hand.write
- handbag
- handball
- handfan
- handholding
- handholding.man.man
- handholding.woman.man
- handholding.woman.woman
- hands.folded
- hands.palms
- hands.clap
- hands.heart
- hands.open
- hands.raised
- hands.shake
- harp
- hash
- hat.ribbon
- hat.top
- headphone
- heart
- heart.arrow
- heart.beat
- heart.black
- heart.blue
- heart.box
- heart.broken
- heart.brown
- heart.double
- heart.excl
- heart.gray
- heart.green
- heart.grow
- heart.lightblue
- heart.orange
- heart.pink
- heart.purple
- heart.real
- heart.revolve
- heart.ribbon
- heart.spark
- heart.white
- heart.yellow
- hedgehog
- helicopter
- helix
- helmet.cross
- helmet.military
- hi
- hippo
- hockey
- hole
- honey
- hongbao
- hook
- horn.postal
- horse
- horse.carousel
- horse.face
- horse.race
- hospital
- hotdog
- hotel
- hotel.love
- hotspring
- hourglass
- hourglass.flow
- house
- house.derelict
- house.garden
- house.multiple
- hundred
- hut
- ice
- icecream
- icecream.shaved
- icecream.soft
- icehockey
- id

ℹ️ info
🏮 izakaya
🫙 jar
👖 jeans
🪼 jellyfish
🧩 jigsaw
🕹️ joystick
🤹 juggling
🧃 juice
🈯 ka
🕋 kaaba
🈹 kachi
🎍 kadomatsu
🦘 kangaroo
🈳 kara
🥙 kebab
🔑 key
🗝️ key.old
⌨️ keyboard
💏 kiss
💋 kissmark
🪁 kite
🥝 kiwi
🔪 knife
🗡️ knife.dagger
🪢 knot
🐨 koala
🎏 koinobori
🈁 koko
🈲 kon
🏷️ label
🥍 lacrosse
🪜 ladder
🪔 lamp.diya
🛖 landslide
💻 laptop
☘️ leaf.clover.three
🍀 leaf.clover.four
🍂 leaf.fall
🌿 leaf.herb
🍁 leaf.maple
🍃 leaf.wind
🛄 leftluggage
🦵 leg
🦿 leg.mech
🍋 lemon
🐆 leopard
💌 letter.love
🗽 liberty
💡 lightbulb
⚡ lightning
🦁 lion
💄 lipstick
🚮 litter
🚯 litter.not
🦎 lizard
🦙 llama
🦞 lobster
🔒 lock
🔐 lock.key
🔓 lock.open
🔏 lock.pen
🍭 lollipop
🧴 lotion
🧳 luggage
🫁 lungs
🧙 mage
🧲 magnet
🔎 magnify.r
🔍 magnify.l
🀄 mahjong.dragon.red
✉️ mail
📩 mail.arrow
📪 mailbox.closed.empty
📫 mailbox.closed.full
📬 mailbox.open.empty
📭 mailbox.open.full
🦣 mammoth
👨 man
🚹 man.box
🤴 man.crown
👳 man.guapimao
🕴️ man.levitate
👴 man.old
🤰 man.pregnant
👳 man.turban
🤵 man.tuxedo
🥭 mango
🗺️ map.world
🗾 map.jp
🪇 maracas
🥋 martialarts
🎭 masks
🧉 mate
🪆 matryoshka
🥩 meat
🍖 meat.bone
🥇 medal.first
🥈 medal.second
🥉 medal.third
🏅 medal.sports
🎖️ medal.military
📣 megaphone
📢 megaphone.simple
🍈 melon
🧜 merperson
Ⓜ️ metro
🦠 microbe

microphone
microphone.studio
microscope
milkyway
mirror
mixer
money.bag
money.dollar
money.euro
money.pound
money.yen
money.wings
monkey
monkey.face
monkey.hear.not
monkey.see.not
monkey.speak.not
moon.crescent
moon.full
moon.full.face
moon.new
moon.new.face
moon.wane.one
moon.wane.two
moon.wane.three.face
moon.wane.three
moon.wax.one
moon.wax.two
moon.wax.two.face
moon.wax.three
moose
mortarboard
mosque
mosquito
motorcycle
motorway
mountain
mountain.fuji
mountain.snow
mountain.sunrise
mouse
mouse.face
mousetrap
mouth
mouth.bite
moyai
muryo
museum
mushroom
musicalscore
nails.polish
namebadge
nazar
necktie
needle
nest.empty
nest.eggs
new
newspaper
newspaper.rolled
ng
ningyo
ninja
noentry
nose
notebook
notebook.deco
notepad
notes
notes.triple
numbers
o
octopus
office
oil
ok
olive
oni
onion
orangutan
orca
otter
owl
ox
oyster
package
paella
page
page.curl
page.pencil
pager
pages.tabs
painting
palette
pancakes
panda
parachute
park
parking
parrot
partalteration
party
peach
peacock
peanuts
peapod
pear
pedestrian

- pedestrian.not
- pen.ball
- pen.fountain
- pencil
- penguin
- pepper
- pepper.hot
- person
- person.angry
- person.beard
- person.blonde
- person.bow
- person.crown
- person.deaf
- person.facepalm
- person.frown
- person.hijab
- person.kneel
- person.lotus
- person.massage
- person.no
- person.ok
- person.old
- person.pregnant
- person.raise
- person.sassy
- person.shrug
- person.stand
- person.steam
- petri
- phone
- phone.arrow
- phone.classic
- phone.not
- phone.off
- phone.receiver
- phone.signal
- phone.vibrate
- piano
- pick
- pie
- pig
- pig.face
- pig.nose
- pill
- pin
- pin.round
- pinata
- pineapple
- pingpong
- pistol
- pizza
- placard
- planet
- plant
- plaster
- plate.cutlery
- playback.down
- playback.eject
- playback.forward
- playback.pause
- playback.record
- playback.repeat
- playback.repeat.once
- playback.repeat.v
- playback.restart
- playback.rewind
- playback.shuffle
- playback.skip
- playback.stop
- playback.toggle
- playback.up
- playingcard.flower
- playingcard.joker
- plunger
- policeofficer
- poo
- popcorn
- post.eu
- post.jp
- postbox
- potato
- potato.sweet
- pouch
- powerplug
- present
- pretzel
- printer
- prints.foot
- prints.paw
- prohibited
- projector
- pumpkin.lantern
- purse
- quest
- quest.white
- rabbit
- rabbit.face
- raccoon
- radio
- radioactive
- railway
- rainbow
- ram
- rat
- razor
- receipt
- recycling
- reg

- restroom
- rhino
- ribbon
- ribbon.remind
- rice
- rice.cracker
- rice.ear
- rice.onigiri
- ring
- ringbuoy
- robot
- rock
- rocket
- rollercoaster
- rosette
- rugby
- ruler
- ruler.triangle
- running
- sa
- safetypin
- safetyvest
- sake
- salad
- salt
- sandwich
- santa.man
- santa.woman
- satdish
- satellite
- saw
- saxophone
- scales
- scarf
- school
- scissors
- scooter
- scooter.motor
- scorpion
- screwdriver
- scroll
- seal
- seat
- seedling
- shark
- sheep
- shell.spiral
- shield
- shin
- ship
- ship.cruise
- ship.ferry
- shirt.sports
- shirt.t
- shoe
- shoe.ballet
- shoe.flat
- shoe.heel
- shoe.hike
- shoe.ice
- shoe.roller
- shoe.sandal.heel
- shoe.ski
- shoe.sneaker
- shoe.tall
- shoe.thong
- shopping
- shorts
- shoshinsha
- shovel
- shower
- shrimp
- shrimp.fried
- shrine
- shuku
- sign.crossing
- sign.stop
- silhouette
- silhouette.double
- silhouette.hug
- silhouette.speak
- siren
- skateboard
- skewer.dango
- skewer.oden
- skiing
- skull
- skull.bones
- skunk
- sled
- slide
- slider
- sloth
- slots
- snail
- snake
- snowboarding
- snowflake
- snowman
- snowman.snow
- soap
- socks
- softball
- sos
- soup
- spaghetti
- sparkle.box
- sparkler

- ✨ sparkles
- 🔈 speaker
- 🔇 speaker.not
- 🔉 speaker.wave
- 🔊 speaker.waves
- 🕷 spider
- 🕸 spiderweb
- 🥬 spinach
- ▫ splatter
- 🧽 sponge
- 🥄 spoon
- ■ square.black
- ▪ square.black.tiny
- ■ square.black.small
- ■ square.black.medium
- □ square.white
- ▫ square.white.tiny
- □ square.white.small
- □ square.white.medium
- 🟦 square.blue
- 🟫 square.brown
- 🟩 square.green
- 🟧 square.orange
- 🟪 square.purple
- 🟥 square.red
- 🟨 square.yellow
- 🦑 squid
- 🏟 stadium
- ⋆ star
- 🌠 star.arc
- ✴ star.box
- 🌟 star.glow
- 🌠 star.shoot
- 🩺 stethoscope
- 🏬 store.big
- 🏪 store.small
- 🍓 strawberry
- ♣ suit.club
- ♦ suit.diamond
- ♥ suit.heart
- ♠ suit.spade
- ☀ sun
- 🌤 sun.cloud
- 🌞 sun.face
- 🌅 sunrise
- 🦸 superhero
- 🦹 supervillain
- 🏄 surfing
- 🍣 sushi
- 🦢 swan
- 🏊 swimming
- 🩱 swimsuit
- ⚔ swords
- 🔣 symbols
- 🕍 synagogue
- 💉 syringe
- 🌮 taco
- 🥡 takeout
- 🫔 tamale
- 🎋 tanabata
- 🍊 tangerine
- 🚰 tap
- 🚱 tap.not
- 🚕 taxi
- 🚖 taxi.front
- 🍵 teacup
- 🫖 teapot
- 🧸 teddy
- 🔭 telescope
- 🛕 temple
- 🔟 ten
- 👺 tengu
- 🎾 tennis
- ⛺ tent
- 🧪 testtube
- 🌡 thermometer
- 🧵 thread
- 👍 thumb.up
- 👎 thumb.down
- 🎫 ticket.event
- 🎟 ticket.travel
- 🐅 tiger
- 🐯 tiger.face
- ™ tm
- 🚽 toilet
- 🧻 toiletpaper
- 🉐 toku
- 🍅 tomato
- 🪦 tombstone
- 👅 tongue
- 🧰 toolbox
- 🦷 tooth
- 🪥 toothbrush
- 🌪 tornado
- 🗼 tower.tokyo
- 🖲 trackball
- 🚜 tractor
- 🚦 trafficlight.v
- 🚥 trafficlight.h
- 🚆 train
- 🚃 train.car
- 🚈 train.light
- 🚇 train.metro
- 🚝 train.mono
- 🚞 train.mountain
- 🚄 train.speed
- 🚅 train.speed.bullet
- 🚂 train.steam
- 🚊 train.stop

train.suspend
train.tram
train.tram.car
transgender
tray.inbox
tray.mail
tray.outbox
treasure
tree.deciduous
tree.evergreen
tree.leafless
tree.palm
tree.xmas
triangle.r
triangle.l
triangle.t
triangle.b
triangle.t.red
triangle.b.red
trident
troll
trombone
trophy
truck
truck.trailer
trumpet
tsukimi
turkey
turtle
tv
ufo
umbrella.open
umbrella.closed
umbrella.rain
umbrella.sun
unicorn
unknown
up
urn
vampire
violin
volcano
volleyball
vs
waffle
wand
warning
watch
watch.stop
watermelon
waterpolo
wave
wc
weightlifting
whale
whale.spout
wheel
wheelchair
wheelchair.box
wheelchair.motor
wind
windchime
window
wine
wing
wireless
wolf
woman
woman.box
woman.crown
woman.old
woman.pregnant
wood
worm
wrench
wrestling
xray
yarn
yo
yoyo
yubi
yuryo
zebra
zodiac.aquarius
zodiac.aries
zodiac.cancer
zodiac.capri
zodiac.gemini
zodiac.leo
zodiac.libra
zodiac.ophi
zodiac.pisces
zodiac.sagit
zodiac.scorpio
zodiac.taurus
zodiac.virgo
zombie
zzz

3.10 Layout

Arranging elements on the page in different ways.

By combining layout functions, you can create complex and automatic layouts.

Definitions

3.10.1 Align Aligns content horizontally and vertically. . . . 330
3.10.2 Alignment Where to align something along an axis. . . . 332
3.10.3 Angle An angle describing a rotation. . . . 334
3.10.4 Block A block-level container. . . . 335
3.10.5 Box An inline-level container that sizes content. . . . 339
3.10.6 Column Break Forces a column break. . . . 342
3.10.7 Columns Separates a region into multiple equally sized columns. . . . 343
3.10.8 Direction The four directions into which content can be laid out. . . . 345
3.10.9 Fraction Defines how the remaining space in a layout is distributed. . . . 348
3.10.10 Grid Arranges content in a grid. . . . 349
3.10.11 Hide Hides content without affecting layout. . . . 361
3.10.12 Layout Provides access to the current outer container's (or page's, if none) dimensions (width and height). . . . 362
3.10.13 Length A size or distance, possibly expressed with contextual units. . . . 364
3.10.14 Measure Measures the layouted size of content. . . . 366
3.10.15 Move Moves content without affecting layout. . . . 368
3.10.16 Padding Adds spacing around content. . . . 369
3.10.17 Page Layouts its child onto one or multiple pages. . . . 371
3.10.18 Page Break A manual page break. . . . 381
3.10.19 Place Places content relatively to its parent container. . . . 382
3.10.20 Ratio A ratio of a whole. . . . 386
3.10.21 Relative Length A length in relation to some known length. . . . 387
3.10.22 Repeat Repeats content to the available space. . . . 389
3.10.23 Rotate Rotates content without affecting layout. . . . 390
3.10.24 Scale Scales content without affecting layout. . . . 392
3.10.25 Skew Skews content. . . . 394
3.10.26 Spacing (H) Inserts horizontal spacing into a paragraph. . . . 396
3.10.27 Spacing (V) Inserts vertical spacing into a flow of blocks. . . . 398
3.10.28 Stack Arranges content and spacing horizontally or vertically. . . . 399

3.10.1 `align` Element

Aligns content horizontally and vertically.

/1 Example

Let's start with centering our content horizontally:

```
#set page(height: 120pt)
#set align(center)

Centered text, a sight to see \
In perfect balance, visually \
Not left nor right, it stands alone \
A work of art, a visual throne
```

Centered text, a sight to see
In perfect balance, visually
Not left nor right, it stands alone
A work of art, a visual throne

To center something vertically, use *horizon* alignment:

```
#set page(height: 120pt)
#set align(horizon)

Vertically centered, \
the stage had entered, \
a new paragraph.
```

Vertically centered,
the stage had entered,
a new paragraph.

/2 Combining alignments

You can combine two alignments with the + operator. Let's also only apply this to one piece of content by using the function form instead of a set rule:

```
#set page(height: 120pt)
Though left in the beginning ...

#align(right + bottom)[
  ... they were right in the end, \
  and with addition had gotten, \
  the paragraph to the bottom!
]
```

Though left in the beginning ...

... they were right in the end,
and with addition had gotten,
the paragraph to the bottom!

/3 Nested alignment

You can use varying alignments for layout containers and the elements within them. This way, you can create intricate layouts:

```
#align(center, block[
  #set align(left)
  Though centered together \
  alone \
  we \
  are \
  left.
])
```

Though centered together
alone
we
are
left.

/4 Alignment within the same line

The `align` function performs block-level alignment and thus always interrupts the current paragraph. To have different alignment for parts of the same line, you should use fractional spacing[3.10.26] instead:

```
Start #h(1fr) End
```

Start End

/5 Parameters

```
align(
  alignment,
  content,
) → content
```

/5.1 alignment `alignment` *Positional* *Settable*

Default: `start + top`

The alignment[3.10.2] along both axes.

```
#set page(height: 6cm)
#set text(lang: "ar")

مثال
#align(
  end + horizon,
  rect(inset: 12pt)[ركن]
)
```

مثال

ركن

/5.2 body `content` *Required* *Positional*

The content to align.

3.10.2 `alignment`

Where to align something along an axis.

Possible values are:
- `start`: Aligns at the start[3.10.8/1.5] of the text direction[3.7.12/2.19].
- `end`: Aligns at the end[3.10.8/1.6] of the text direction[3.7.12/2.19].
- `left`: Align at the left.
- `center`: Aligns in the middle, horizontally.
- `right`: Aligns at the right.
- `top`: Aligns at the top.
- `horizon`: Aligns in the middle, vertically.
- `bottom`: Align at the bottom.

These values are available globally and also in the alignment type's scope, so you can write either of the following two:

```
#align(center)[Hi]
#align(alignment.center)[Hi]
```

Hi

Hi

/1 2D alignments

To align along both axes at the same time, add the two alignments using the `+` operator. For example, `top + right` aligns the content to the top right corner.

```
#set page(height: 3cm)
#align(center + bottom)[Hi]
```

Hi

/2 Fields

The `x` and `y` fields hold the alignment's horizontal and vertical components, respectively (as yet another `alignment`). They may be `none`.

```
#(top + right).x \
#left.x \
#left.y (none)
```

right
left
(none)

/3 Definitions

/3.1 `axis`

The axis this alignment belongs to.
- `"horizontal"` for `start`, `left`, `center`, `right`, and `end`
- `"vertical"` for `top`, `horizon`, and `bottom`
- `none` for 2-dimensional alignments

```
#left.axis() \
#bottom.axis()
```

horizontal
vertical

```
self.axis() → none str
```

/3.2 inv

The inverse alignment.

```
#top.inv() \
#left.inv() \
#center.inv() \
#(left + bottom).inv()
```

```
bottom
right
center
right + top
```

```
self.inv() → alignment
```

3.10.3 `angle`

An angle describing a rotation.

Typst supports the following angular units:

- Degrees: `180deg`
- Radians: `3.14rad`

/1 Example

```
#rotate(10deg)[Hello there!]
```



/2 Definitions

/2.1 `rad`

Converts this angle to radians.

```
self.rad() → float
```

/2.2 `deg`

Converts this angle to degrees.

```
self.deg() → float
```

3.10.4 `block` Element

A block-level container.

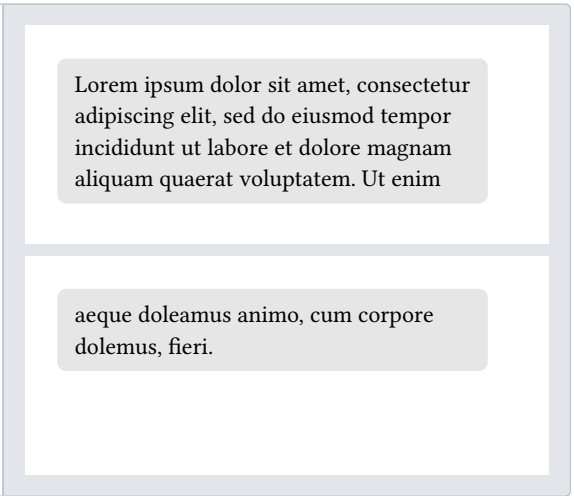
Such a container can be used to separate content, size it, and give it a background or border.

Blocks are also the primary way to control whether text becomes part of a paragraph or not. See the paragraph documentation[3.6.15/2] for more details.

/1 Examples

With a block, you can give a background to content while still allowing it to break across multiple pages.

```
#set page(height: 100pt)
#block(
  fill: luma(230),
  inset: 8pt,
  radius: 4pt,
  lorem(30),
)
```



Blocks are also useful to force elements that would otherwise be inline to become block-level, especially when writing show rules.

```
#show heading: it => it.body
= Blockless
More text.

#show heading: it => block(it.body)
= Blocky
More text.
```

Blockless More text.

Blocky

More text.

/2 Parameters

```
block(
  width: auto relative,
  height: auto relative fraction,
  breakable: bool,
  fill: none color gradient tiling,
  stroke: none length color gradient stroke tiling dictionary,
  radius: relative dictionary,
  inset: relative dictionary,
  outset: relative dictionary,
  spacing: auto relative fraction,
  above: auto relative fraction,
  below: auto relative fraction,
  clip: bool,
  sticky: bool,
  none content,
) → content
```

/2.1 **width** `auto` or `relative` *Settable* Default: `auto`

The block's width.

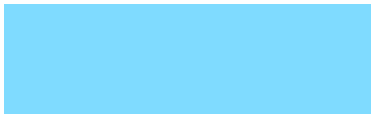
```
#set align(center)
#block(
  width: 60%,
  inset: 8pt,
  fill: silver,
  lorem(10),
)
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

/2.2 **height** `auto` or `relative` or `fraction` *Settable* Default: `auto`

The block's height. When the height is larger than the remaining space on a page and `breakable`[3.10.4/2.3] is `true`, the block will continue on the next page with the remaining height.

```
#set page(height: 80pt)
#set align(center)
#block(
  width: 80%,
  height: 150%,
  fill: aqua,
)
```





/2.3 **breakable** `bool` *Settable* Default: `true`

Whether the block can be broken and continue on the next page.

```
#set page(height: 80pt)
The following block will
jump to its own page.
#block(
  breakable: false,
  lorem(15),
)
```

The following block will jump to its own page.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.

/2.4 **fill** `none` or `color` or `gradient` or `tiling` *Settable* Default: `none`

The block's background color. See the rectangle's documentation[3.11.10/2.3] for more details.

/2.5 **stroke** `none` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable* Default: `(:)`

The block's border color. See the rectangle's documentation[3.11.10/2.4] for more details.

/2.6 **radius** `relative` or `dictionary` *Settable* Default: `(:)`

How much to round the block's corners. See the rectangle's documentation[3.11.10/2.5] for more details.

/2.7 **inset** `relative` or `dictionary` *Settable* Default: `(:)`

How much to pad the block's content. See the box's documentation[3.10.5/2.7] for more details.

/2.8 **outset** `relative` or `dictionary` *Settable* Default: `(:)`

How much to expand the block's size without affecting the layout. See the box's documentation[3.10.5/2.8] for more details.

/2.9 **spacing** `auto` or `relative` or `fraction` Default: `1.2em`

The spacing around the block. When `auto`, inherits the paragraph `spacing`[3.6.15/4.2].

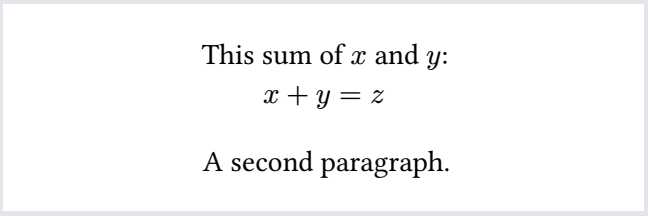
For two adjacent blocks, the larger of the first block's `below` and the second block's `above` spacing wins. Moreover, block spacing takes precedence over paragraph `spacing`[3.6.15/4.2].

Note that this is only a shorthand to set `above` and `below` to the same value. Since the values for `above` and `below` might differ, a context[3.4] block only provides access to `block.above` and `block.below`, not to `block.spacing` directly.

This property can be used in combination with a show rule to adjust the spacing around arbitrary block-level elements.

```
#set align(center)
#show math.equation: set block(above: 8pt, below: 16pt)

This sum of $x$ and $y$:
$ x + y = z $
A second paragraph.
```



/2.10 **above** `auto` or `relative` or `fraction` *Settable* Default: `auto`

The spacing between this block and its predecessor.

/2.11 **below** `auto` or `relative` or `fraction` *Settable* Default: `auto`

The spacing between this block and its successor.

/2.12 **clip** `bool` *Settable* Default: `false`

Whether to clip the content inside the block.

Clipping is useful when the block's content is larger than the block itself, as any content that exceeds the block's bounds will be hidden.

```
#block(
  width: 50pt,
  height: 50pt,
  clip: true,
  image("tiger.jpg", width: 100pt, height: 100pt)
)
```



/2.13 **sticky** `bool` *Settable* Default: `false`

Whether this block must stick to the following one, with no break in between.

This is, by default, set on heading blocks to prevent orphaned headings at the bottom of the page.

```
// Disable stickiness of headings.
#show heading: set block(sticky: false)
#lorem(20)

= Chapter
#lorem(10)
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Chapter

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

/2.14 **body** `none` or `content` *Positional* *Settable* Default: `none`

The contents of the block.

3.10.5 box Element

An inline-level container that sizes content.

All elements except inline math, text, and boxes are block-level and cannot occur inside of a paragraph[3.6.15]. The box function can be used to integrate such elements into a paragraph. Boxes take the size of their contents by default but can also be sized explicitly.

/1 Example

```
Refer to the docs
#box(
  height: 9pt,
  image("docs.svg")
)
for more information.
```

Refer to the docs for more information.

/2 Parameters

```
box(
  width: auto relative fraction,
  height: auto relative,
  baseline: auto relative dictionary alignment,
  fill: none color gradient tiling,
  stroke: none length color gradient stroke tiling dictionary,
  radius: relative dictionary,
  inset: relative dictionary,
  outset: relative dictionary,
  clip: bool,
  none content,
) → content
```

/2.1 width auto or relative or fraction *Settable* Default: auto

The width of the box.

Boxes can have fractional[3.10.9] widths, as the example below demonstrates.

Note: Currently, only boxes and only their widths might be fractionally sized within paragraphs. Support for fractionally sized images, shapes, and more might be added in the future.

```
Line in #box(width: 1fr, line(length: 100%)) between.
```

Line in _____ between.

/2.2 height auto or relative *Settable* Default: auto

The height of the box.

/2.3 baseline auto or relative or dictionary or alignment *Settable*

Default: (at: auto, shift: 0% + 0pt)

The vertical position of the box's baseline. This is used to align the box with the text surrounding it in a paragraph, as the baseline is meant to go right below text by default.

By default, the box's baseline will match the baseline of its contents (for example, of the text or equation inside it) - this is the auto option. However, the baseline can be adjusted in two ways. The first one is to simply pick a vertical alignment[3.10.2], such as top, horizon or bottom, to place the baseline at that position, relative to the total height of the box (including inset).

The other way to adjust it is to shift the default baseline vertically by some amount, specified as a relative length[3.10.21]. For example, a value of `2pt` will move it up by that exact length (causing the contents to go *down*, as the alignment point moves *up*), whereas `-40%` will shift the baseline *down* by 40% of the box's total height, including inset (thus causing the contents to move *up*).

Both options can be specified at the same time through a dictionary with the keys `at` and `shift`, respectively. For example, when specifying `(at: bottom, shift: 10pt)`, the box's baseline will be set to the height exactly `10pt` above the bottom of its contents.

```
Image: #box(baseline: 40%, image("tiger.jpg", width: 2cm)).
```

Image: .

/2.4 **fill** `none` or `color` or `gradient` or `tiling` *Settable* Default: `none`

The box's background color. See the rectangle's documentation[3.11.10/2.3] for more details.

/2.5 **stroke** `none` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable* Default: `(:)`

The box's border color. See the rectangle's documentation[3.11.10/2.4] for more details.

/2.6 **radius** `relative` or `dictionary` *Settable* Default: `(:)`

How much to round the box's corners. See the rectangle's documentation[3.11.10/2.5] for more details.

/2.7 **inset** `relative` or `dictionary` *Settable* Default: `(:)`

How much to pad the box's content.

This can be a single length for all sides or a dictionary of lengths for individual sides. When passing a dictionary, it can contain the following keys in order of precedence: `top`, `right`, `bottom`, `left` (controlling the respective cell sides), `x`, `y` (controlling vertical and horizontal insets), and `rest` (covers all insets not styled by other dictionary entries). All keys are optional; omitted keys will use their previously set value, or the default value if never set.

Relative lengths[3.10.21] for this parameter are relative to the box size excluding outset[3.10.5/2.8]. Note that relative insets and outsets are different from relative widths[3.10.5/2.1] and heights[3.10.5/2.2], which are relative to the container.

Note: When the box contains text, its exact size depends on the current text edges[3.7.12/2.14].

```
#rect(inset: 0pt)[Tight]
```

Tight

/2.8 **outset** `relative` or `dictionary` *Settable* Default: `(:)`

How much to expand the box's size without affecting the layout.

This can be a single length for all sides or a dictionary of lengths for individual sides. Relative lengths[3.10.21] for this parameter are relative to the box size excluding outset. See the documentation for inset[3.10.5/2.7] above for further details.

This is useful to prevent padding from affecting line layout. For a generalized version of the example below, see the documentation for the raw text's block parameter[3.7.6/4.2].

```
An inline
#box(
  fill: luma(235),
```

An inline rectangle.

```
  inset: (x: 3pt, y: 0pt),
  outset: (y: 3pt),
  radius: 2pt,
)[rectangle].
```

/2.9 **clip** `bool` *Settable* Default: `false`

Whether to clip the content inside the box.

Clipping is useful when the box's content is larger than the box itself, as any content that exceeds the box's bounds will be hidden.

```
#box(
  width: 50pt,
  height: 50pt,
  clip: true,
  image("tiger.jpg", width: 100pt, height: 100pt)
)
```



/2.10 **body** `none` or `content` *Positional* *Settable* Default: `none`

The contents of the box.

3.10.6 `colbreak` Element

Forces a column break.

The function will behave like a page break[3.10.18] when used in a single column layout or the last column on a page. Otherwise, content after the column break will be placed in the next column.

/1 Example

```
#set page(columns: 2)
Preliminary findings from our
ongoing research project have
revealed a hitherto unknown
phenomenon of extraordinary
significance.

#colbreak()
Through rigorous experimentation
and analysis, we have discovered
a hitherto uncharacterized process
that defies our current
understanding of the fundamental
laws of nature.
```

Preliminary findings from our ongoing research project have revealed a hitherto unknown phenomenon of extraordinary significance.

Through rigorous experimentation and analysis, we have discovered a hitherto uncharacterized process that defies our current understanding of the fundamental laws of nature.

/2 Parameters

```
colbreak(weak: bool) → content
```

/2.1 `weak` `bool` *Settable* Default: `false`

If `true`, the column break is skipped if the current column is already empty.

3.10.7 `columns` Element

Separates a region into multiple equally sized columns.

The `column` function lets you separate the interior of any container into multiple columns. It will currently not balance the height of the columns. Instead, the columns will take up the height of their container or the remaining height on the page. Support for balanced columns is planned for the future.

When arranging content across multiple columns, use `colbreak`[3.10.6] to explicitly continue in the next column.

/1 Example

```
#columns(2, gutter: 8pt)[
  This text is in the
  first column.

  #colbreak()

  This text is in the
  second column.
]
```

This text is in the first column. This text is in the second column.

/2 Page-level columns

If you need to insert columns across your whole document, use the `page` function's `columns` parameter[3.10.17/4.8] instead. This will create the columns directly at the page-level rather than wrapping all of your content in a layout container. As a result, things like pagebreaks[3.10.18], footnotes[3.6.9], and line numbers[3.6.15/5.1] will continue to work as expected. For more information, also read the relevant part of the page setup guide[4.2.4].

/3 Breaking out of columns

To temporarily break out of columns (e.g. for a paper's title), use parent-scoped floating placement:

```
#set page(columns: 2, height: 150pt)

#place(
  top + center,
  scope: "parent",
  float: true,
  text(1.4em, weight: "bold")[
    My document
  ],
)

#lorem(40)
```

My document

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod

/4 Parameters

```
columns(
  int,
  gutter: relative,
  content,
) → content
```

/4.1 `count` `int` *Positional* *Settable* Default: 2

The number of columns.

/4.2 `gutter` `relative` *Settable* Default: `4% + 0pt`

The size of the gutter space between each column.

/4.3 **body** `content` *Required* *Positional*

The content that should be layouted into the columns.

3.10.8 direction

The four directions into which content can be laid out.

Possible values are:
– `ltr`: Left to right.
– `rtl`: Right to left.
– `ttb`: Top to bottom.
– `btt`: Bottom to top.

These values are available globally and also in the direction type's scope, so you can write either of the following two:

```
#stack(dir: rtl)[A][B][C]
#stack(dir: direction.rtl)[A][B][C]
```

CBA

CBA

/1 Definitions

/1.1 from

Returns a direction from a starting point.

```
#direction.from(left) \
#direction.from(right) \
#direction.from(top) \
#direction.from(bottom)
```

```
ltr
rtl
ttb
btt
```

```
direction.from(alignment) → direction
```

/1.1.1 **side** `alignment` *Required* *Positional*

/1.2 to

Returns a direction from an end point.

```
#direction.to(left) \
#direction.to(right) \
#direction.to(top) \
#direction.to(bottom)
```

```
rtl
ltr
btt
ttb
```

```
direction.to(alignment) → direction
```

/1.2.1 **side** `alignment` *Required* *Positional*

/1.3 axis

The axis this direction belongs to, either `"horizontal"` or `"vertical"`.

```
#ltr.axis() \
#ttb.axis()
```

horizontal
vertical

```
self.axis() → str
```

/1.4 sign

The corresponding sign, for use in calculations.

```
#ltr.sign() \
#rtl.sign() \
#ttb.sign() \
#btt.sign()
```

```
1
−1
1
−1
```

```
self.sign() → int
```

/1.5 start

The start point of this direction, as an alignment.

```
#ltr.start() \
#rtl.start() \
#ttb.start() \
#btt.start()
```

```
left
right
top
bottom
```

```
self.start() → alignment
```

/1.6 end

The end point of this direction, as an alignment.

```
#ltr.end() \
#rtl.end() \
#ttb.end() \
#btt.end()
```

```
right
left
bottom
top
```

```
self.end() → alignment
```

/1.7 inv

The inverse direction.

```
#ltr.inv() \
#rtl.inv() \
#ttb.inv() \
#btt.inv()
```

```
rtl
ltr
btt
ttb
```

```
self.inv() → direction
```

3.10.9 `fraction`

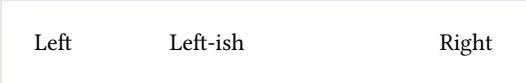
Defines how the remaining space in a layout is distributed.

Each fractionally sized element gets space based on the ratio of its fraction to the sum of all fractions.

For more details, also see the `h`[3.10.26] and `v`[3.10.27] functions and the `grid` function[3.10.10].

/1 Example

```
Left #h(1fr) Left-ish #h(2fr) Right
```



3.10.10 `grid` Element

Arranges content in a grid.

The grid element allows you to arrange content in a grid. You can define the number of rows and columns, as well as the size of the gutters between them. There are multiple sizing modes for columns and rows that can be used to create complex layouts.

While the grid and table elements work very similarly, they are intended for different use cases and carry different semantics. The grid element is intended for presentational and layout purposes, while the `table`[3.6.20] element is intended for, in broad terms, presenting multiple related data points. Set and show rules on one of these elements do not affect the other. Refer to the Accessibility Section[3.10.10/4] to learn how grids and tables are presented to users of Assistive Technology (AT) like screen readers.

/1 Sizing the tracks

A grid's sizing is determined by the track sizes specified in the arguments. There are multiple sizing parameters: `columns`[3.10.10/5.1], `rows`[3.10.10/5.2] and `gutter`[3.10.10/5.3]. Because each of the sizing parameters accepts the same values, we will explain them just once, here. Each sizing argument accepts an array of individual track sizes. A track size is either:

- `auto`: The track will be sized to fit its contents. It will be at most as large as the remaining space. If there is more than one `auto` track width, and together they claim more than the available space, the `auto` tracks will fairly distribute the available space among themselves.
- A fixed or relative length (e.g. `10pt` or `20% - 1cm`): The track will be exactly of this size.
- A fractional length (e.g. `1fr`): Once all other tracks have been sized, the remaining space will be divided among the fractional tracks according to their fractions. For example, if there are two fractional tracks, each with a fraction of `1fr`, they will each take up half of the remaining space.

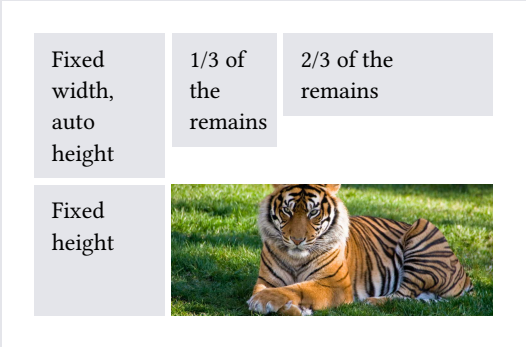
To specify a single track, the array can be omitted in favor of a single value. To specify multiple `auto` tracks, enter the number of tracks instead of an array. For example, `columns: 3` is equivalent to `columns: (auto, auto, auto)`.

/2 Examples

The example below demonstrates the different track sizing options. It also shows how you can use `grid.cell`[3.10.10/6.1] to make an individual cell span two grid tracks.

```
// We use `rect` to emphasize the
// area of cells.
#set rect(
  inset: 8pt,
  fill: rgb("e4e5ea"),
  width: 100%,
)

#grid(
  columns: (60pt, 1fr, 2fr),
  rows: (auto, 60pt),
  gutter: 3pt,
  rect[Fixed width, auto height],
  rect[1/3 of the remains],
  rect[2/3 of the remains],
  rect(height: 100%)[Fixed height],
  grid.cell(
    colspan: 2,
    image("tiger.jpg", width: 100%),
  ),
)
```



You can also spread[3.5.1/2] an array of strings or content into a grid to populate its cells.

```
#grid(
  columns: 5,
  gutter: 5pt,
  ..range(25).map(str)
)
```

0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24

/3 Styling the grid

The grid and table elements work similarly. For a hands-on explanation, refer to the Table Guide[4.3.4]; for a quick overview, continue reading.

The grid's appearance can be customized through different parameters. These are the most important ones:

- `align`[3.10.10/5.7] to change how cells are aligned
- `inset`[3.10.10/5.6] to optionally add internal padding to cells
- `fill`[3.10.10/5.8] to give cells a background
- `stroke`[3.10.10/5.9] to optionally enable grid lines with a certain stroke

To meet different needs, there are various ways to set them.

If you need to override the above options for individual cells, you can use the `grid.cell`[3.10.10/6.1] element. Likewise, you can override individual grid lines with the `grid.hline`[3.10.10/6.2] and `grid.vline`[3.10.10/6.3] elements.

To configure an overall style for a grid, you may instead specify the option in any of the following fashions:

- As a single value that applies to all cells.
- As an array of values corresponding to each column. The array will be cycled if there are more columns than the array has items.
- As a function in the form of `(x, y) => value`. It receives the cell's column and row indices (both starting from zero) and should return the value to apply to that cell.

```
#grid(
  columns: 5,

  // By a single value
  align: center,
  // By a single but more complicated value
  inset: (x: 2pt, y: 3pt),
  // By an array of values (cycling)
  fill: (rgb("#239dad50"), none),
  // By a function that returns a value
  stroke: (x, y) => if calc.rem(x + y, 3) == 0 { 0.5pt },

  ..range(5 * 3).map(n => numbering("A", n + 1))
)
```

A B C D E
F G H I J
K L M N O

On top of that, you may apply styling rules[3.2] to `grid`[3.10.10] and `grid.cell`[3.10.10/6.1]. Especially, the `x`[3.10.10/6.1.2] and `y`[3.10.10/6.1.3] fields of `grid.cell` can be used in a `where`[3.5.15/8.2] selector, making it possible to style cells at specific columns or rows, or individual positions.

/3.1 Stroke styling precedence

As explained above, there are three ways to set the stroke of a grid cell: through `grid.cell`'s `stroke` field[3.10.10/6.1.9], by using `grid.hline`[3.10.10/6.2] and `grid.vline`[3.10.10/6.3], or by setting the `grid`'s `stroke`

field[3.10.10/5.9]. When multiple of these settings are present and conflict, the `hline` and `vline` settings take the highest precedence, followed by the `cell` settings, and finally the `grid` settings.

Furthermore, strokes of a repeated grid header or footer will take precedence over regular cell strokes.

/4 Accessibility

Grids do not carry any special semantics. Assistive Technology (AT) does not offer the ability to navigate two-dimensionally by cell in grids. If you want to present tabular data, use the `table`[3.6.20] element instead.

AT will read the grid cells in their semantic order. Usually, this is the order in which you passed them to the grid. However, if you manually positioned them using `grid.cell`'s `x` and `y` arguments[3.10.10/6.1.2], cells will be read row by row, from left to right (in left-to-right documents). A cell will be read when its position is first reached.

/5 Parameters

```
grid(
  columns: auto int relative fraction array,
  rows: auto int relative fraction array,
  gutter: auto int relative fraction array,
  column-gutter: auto int relative fraction array,
  row-gutter: auto int relative fraction array,
  inset: relative array dictionary function,
  align: auto array function alignment,
  fill: none color gradient tiling array function,
  stroke: none length color gradient stroke tiling array dictionary function,
  ..content,
) → content
```

/5.1 `columns` `auto` or `int` or `relative` or `fraction` or `array` *Settable* Default: `()`

The column sizes.

Either specify a track size array or provide an integer to create a grid with that many `auto`-sized columns. Note that opposed to rows and gutters, providing a single track size will only ever create a single column.

See the track size section[3.10.10/1] above for more details.

/5.2 `rows` `auto` or `int` or `relative` or `fraction` or `array` *Settable* Default: `()`

The row sizes.

If there are more cells than fit the defined rows, the last row is repeated until there are no more cells.

See the track size section[3.10.10/1] above for more details.

/5.3 `gutter` `auto` or `int` or `relative` or `fraction` or `array` Default: `()`

The gaps between rows and columns. This is a shorthand to set `column-gutter`[3.10.10/5.4] and `row-gutter`[3.10.10/5.5] to the same value.

If there are more gutters than defined sizes, the last gutter is repeated.

See the track size section[3.10.10/1] above for more details.

/5.4 `column-gutter` `auto` or `int` or `relative` or `fraction` or `array` *Settable* Default: `()`

The gaps between columns.

/5.5 `row-gutter` `auto` or `int` or `relative` or `fraction` or `array` *Settable* Default: `()`

The gaps between rows.

/5.6 `inset` `relative` or `array` or `dictionary` or `function` *Settable* Default: `(:)`

How much to pad the cells' content.

To specify a uniform inset for all cells, you can use a single length for all sides, or a dictionary of lengths for individual sides. See the box's documentation[3.10.5/2.7] for more details.

To specify varying inset for different cells, you can:
– use a single inset for all cells
– use an array of insets corresponding to each column
– use a function that maps a cell's position to its inset

See the styling section[3.10.10/3] above for more details.

In addition, you can find an example at the `table.inset`[3.6.20/3.6] parameter.

/5.7 **align** `auto` or `array` or `function` or `alignment` *Settable* Default: `auto`

How to align the cells' content.

If set to `auto`, the outer alignment is used.

You can specify the alignment in any of the following fashions:
– use a single alignment for all cells
– use an array of alignments corresponding to each column
– use a function that maps a cell's position to its alignment

See the styling section[3.10.10/3] above for details.

In addition, you can find an example at the `table.align`[3.6.20/3.7] parameter.

/5.8 **fill** `none` or `color` or `gradient` or `tiling` or `array` or `function` *Settable* Default: `none`

How to fill the cells.

This can be:
– a single color for all cells
– an array of colors corresponding to each column
– a function that maps a cell's position to its color

Most notably, arrays and functions are useful for creating striped grids. See the styling section[3.10.10/3] above for more details.

```
#grid(
  fill: (x, y) =>
    if calc.even(x + y) { luma(230) }
    else { white },
  align: center + horizon,
  columns: 4,
  inset: 2pt,
  [X], [O], [X], [O],
  [O], [X], [O], [X],
  [X], [O], [X], [O],
  [O], [X], [O], [X],
)
```



/5.9 **stroke** `none` or `length` or `color` or `gradient` or `stroke` or `tiling` or `array` or `dictionary` or `function` *Settable* Default: `(:)`

How to stroke[3.11.12] the cells.

Grids have no strokes by default, which can be changed by setting this option to the desired stroke.

If it is necessary to place lines which can cross spacing between cells produced by the `gutter`[3.10.10/5.3] option, or to override the stroke between multiple specific cells, consider specifying one or more of `grid.hline`[3.10.10/6.2] and `grid.vline`[3.10.10/6.3] alongside your grid cells.

To specify the same stroke for all cells, you can use a single stroke[3.11.12] for all sides, or a dictionary of strokes[3.11.12] for individual sides. See the rectangle's documentation[3.11.10/2.4] for more details.

To specify varying strokes for different cells, you can:
– use a single stroke for all cells
– use an array of strokes corresponding to each column
– use a function that maps a cell's position to its stroke

See the styling section[3.10.10/3] above for more details.

Example · Passing a function to set a stroke based on position

```
#set page(width: 420pt)
#set text(number-type: "old-style")
#show grid.cell.where(y: 0): set text(size: 1.3em)

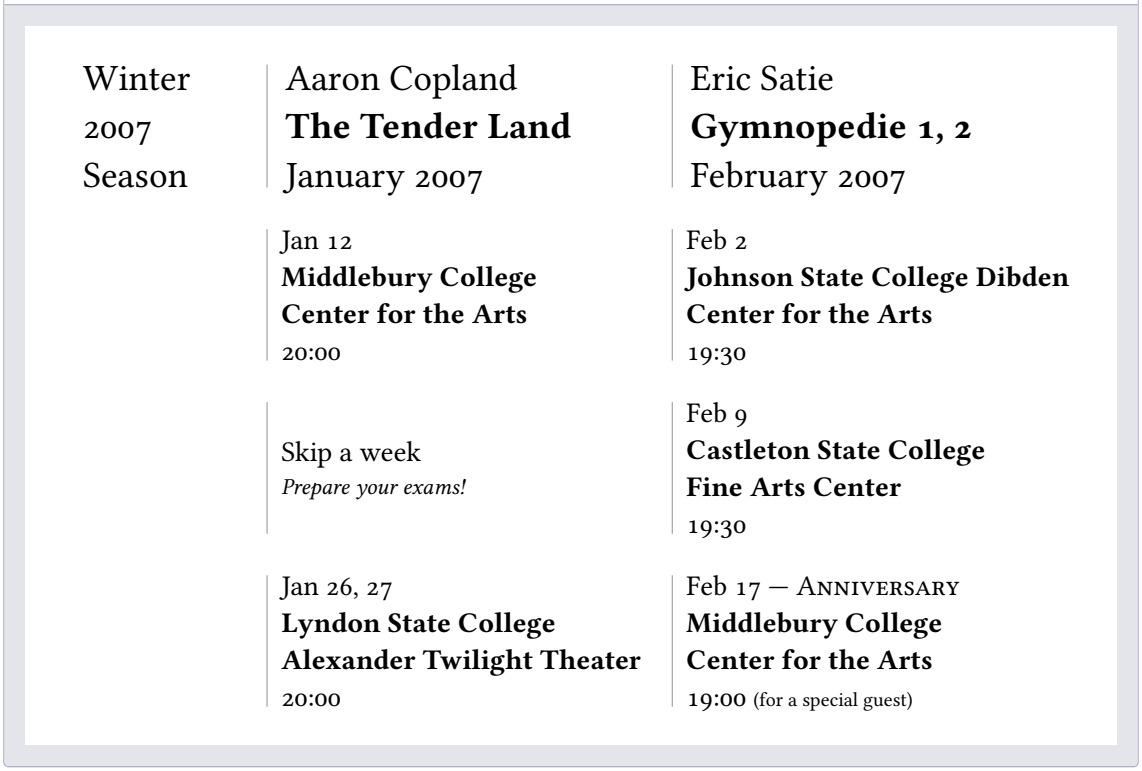
#grid(
  columns: (1fr, 2fr, 2fr),
  row-gutter: 1.5em,
  inset: (left: 0.5em),
  stroke: (x, y) => if x > 0 { (left: 0.5pt + gray) },
  align: horizon,

  [Winter \ 2007 \ Season],
  [Aaron Copland \ *The Tender Land* \ January 2007],
  [Eric Satie \ *Gymnopedie 1, 2* \ February 2007],

  [],
  [Jan 12 \ *Middlebury College \ Center for the Arts* \ 20:00],
  [Feb 2 \ *Johnson State College Dibden Center for the Arts* \ 19:30],

  [],
  [Skip a week \ #text(0.8em)[_Prepare your exams!_]],
  [Feb 9 \ *Castleton State College \ Fine Arts Center* \ 19:30],

  [],
  [Jan 26, 27 \ *Lyndon State College Alexander Twilight Theater* \ 20:00],
  [
    Feb 17 --- #smallcaps[Anniversary] \
    *Middlebury College \ Center for the Arts* \
    19:00 #text(0.7em)[(for a special guest)]
  ],
)
```



Example · Folding the stroke dictionary

```
#set page(height: 13em, width: 26em)

#let cv(..jobs) = grid(
  columns: 2,
  inset: 5pt,
  stroke: (x, y) => if x == 0 and y > 0 {
    (right: (
      paint: luma(180),
      thickness: 1.5pt,
      dash: "dotted",
    ))
  },
  grid.header(grid.cell(colspan: 2)[
    *Professional Experience*
    #box(width: 1fr, line(length: 100%, stroke: luma(180)))
  ]),
  ..{
    let last = none
    for job in jobs.pos() {
      (
        if job.year != last [*#job.year*],
        [
          *#job.company* - #job.role _(#job.timeframe)_ \
          #job.details
        ]
      )
      last = job.year
    }
  }
)

#cv(
  (
    year: 2012,
    company: [Pear Seed & Co.],
    role: [Lead Engineer],
    timeframe: [Jul - Dec],
    details: [
      - Raised engineers from 3x to 10x
      - Did a great job
    ],
  ),
  (
    year: 2012,
    company: [Mega Corp.],
    role: [VP of Sales],
    timeframe: [Mar - Jun],
    details: [- Closed tons of customers],
  ),
  (
    year: 2013,
    company: [Tiny Co.],
    role: [CEO],
    timeframe: [Jan - Dec],
    details: [- Delivered 4x more shareholder value],
  ),
  (
    year: 2014,
    company: [Glorbocorp Ltd],
    role: [CTO],
    timeframe: [Jan - Mar],
    details: [- Drove containerization forward],
```

```
  ),
)
```



/5.10 **children** `content` *Required* *Positional* *Variadic*

The contents of the grid cells, plus any extra grid lines specified with the `grid.hline`[3.10.10/6.2] and `grid.vline`[3.10.10/6.3] elements.

The cells are populated in row-major order.

/6 Definitions

/6.1 **cell** Element

A cell in the grid. You can use this function in the argument list of a grid to override grid style properties for an individual cell or manually positioning it within the grid. You can also use this function in show rules to apply certain styles to multiple cells at once.

For example, you can override the position and stroke for a single cell:

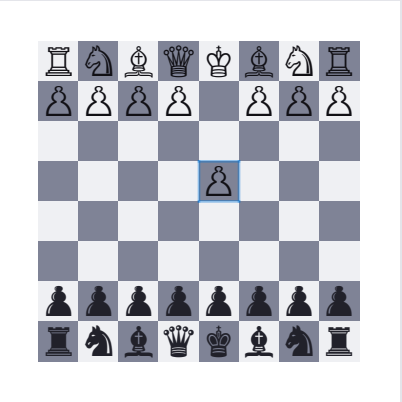
```
#set text(15pt, font: "Noto Sans Symbols 2")
#show regex("[♚-♟]"): set text(fill: rgb("21212A"))
#show regex("[♔-♙]"): set text(fill: rgb("111015"))

#grid(
  fill: (x, y) => rgb(
    if calc.odd(x + y) { "7F8396" }
    else { "EFF0F3" }
  ),
  columns: (1em,) * 8,
  rows: 1em,
  align: center + horizon,

  [♖], [♘], [♗], [♕], [♔], [♗], [♘], [♖],
  [♙], [♙], [♙], [♙], [],  [♙], [♙], [♙],
```

```
  grid.cell(
    x: 4, y: 3,
    stroke: blue.transparentize(60%)
  )[♙],

  ..(grid.cell(y: 6)[♟],) * 8,
  ..([♜], [♞], [♝], [♛], [♚], [♝], [♞], [♜])
    .map(grid.cell.with(y: 7)),
)
```



You may also apply a show rule on `grid.cell` to style all cells at once, which allows you, for example, to apply styles based on a cell's position. Refer to the examples of the `table.cell`[3.6.20/4.1] element to learn more about this.

```
grid.cell(
  content,
  x: auto int,
  y: auto int,
  colspan: int,
  rowspan: int,
  inset: auto relative dictionary,
  align: auto alignment,
  fill: none auto color gradient tiling,
  stroke: none length color gradient stroke tiling dictionary,
  breakable: auto bool,
) → content
```

/6.1.1 **body** `content` *Required* *Positional*

The cell's body.

/6.1.2 **x** `auto` or `int` *Settable* Default: `auto`

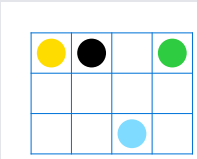
The cell's column (zero-indexed). This field may be used in show rules to style a cell depending on its column.

You may override this field to pick in which column the cell must be placed. If no row (y) is chosen, the cell will be placed in the first row (starting at row 0) with that column available (or a new row if none). If both x and y are chosen, however, the cell will be placed in that exact position. An error is raised if that position is not available (thus, it is usually wise to specify cells with a custom position before cells with automatic positions).

```typ
#let circ(c) = circle(
  fill: c,
  width: 5mm,
)

#grid(
  columns: 4,
  rows: 7mm,
  stroke: .5pt + blue,
  align: center + horizon,
  inset: 1mm,

  grid.cell(x: 2, y: 2, circ(aqua)),
  circ(yellow),
  grid.cell(x: 3, circ(green)),
  circ(black),
)
```



/6.1.3 **y** `auto` or `int` *Settable* Default: `auto`

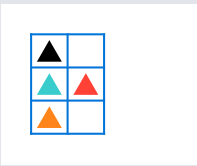
The cell's row (zero-indexed). This field may be used in show rules to style a cell depending on its row.

You may override this field to pick in which row the cell must be placed. If no column (x) is chosen, the cell will be placed in the first column (starting at column 0) available in the chosen row. If all columns in the chosen row are already occupied, an error is raised.

```typ
#let tri(c) = polygon.regular(
  fill: c,
  size: 5mm,
  vertices: 3,
)

#grid(
  columns: 2,
  stroke: blue,
  inset: 1mm,

  tri(black),
  grid.cell(y: 1, tri(teal)),
  grid.cell(y: 1, tri(red)),
  grid.cell(y: 2, tri(orange))
)
```



/6.1.4 **colspan** `int` *Settable* Default: 1

The amount of columns spanned by this cell.

/6.1.5 **rowspan** `int` *Settable* Default: 1

The amount of rows spanned by this cell.

/6.1.6 **inset** `auto` or `relative` or `dictionary` *Settable* Default: `auto`

The cell's inset[3.10.10/5.6] override.

/6.1.7 **align** `auto` or `alignment` *Settable* Default: `auto`

The cell's alignment[3.10.10/5.7] override.

/6.1.8 **fill** `none` or `auto` or `color` or `gradient` or `tiling` *Settable* Default: `auto`

The cell's fill[3.10.10/5.8] override.

/6.1.9 **stroke** `none` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable*
Default: `(:)`

The cell's stroke[3.10.10/5.9] override.

/6.1.10 **breakable** `auto` or `bool` *Settable* Default: `auto`

Whether rows spanned by this cell can be placed in different pages. When equal to `auto`, a cell spanning only fixed-size rows is unbreakable, while a cell spanning at least one `auto`-sized row is breakable.

/6.2 hline Element

A horizontal line in the grid.

Overrides any per-cell stroke, including stroke specified through the grid's `stroke` field. Can cross spacing between cells created through the grid's `column-gutter` option.

An example for this function can be found at the `table.hline`[3.6.20/4.2] element.

```
grid.hline(
  y: auto int,
  start: int,
  end: none int,
  stroke: none length color gradient stroke tiling dictionary,
  position: alignment,
) → content
```

/6.2.1 **y** `auto` or `int` *Settable* Default: `auto`

The row above which the horizontal line is placed (zero-indexed). If the `position` field is set to `bottom`, the line is placed below the row with the given index instead (see `grid.hline.position`[3.10.10/6.2.5] for details).

Specifying `auto` causes the line to be placed at the row below the last automatically positioned cell (that is, cell without coordinate overrides) before the line among the grid's children. If there is no such cell before the line, it is placed at the top of the grid (row 0). Note that specifying for this option exactly the total amount of rows in the grid causes this horizontal line to override the bottom border of the grid, while a value of 0 overrides the top border.

/6.2.2 **start** `int` *Settable* Default: `0`

The column at which the horizontal line starts (zero-indexed, inclusive).

/6.2.3 **end** `none` or `int` *Settable* Default: `none`

The column before which the horizontal line ends (zero-indexed, exclusive). Therefore, the horizontal line will be drawn up to and across column `end - 1`.

A value equal to `none` or to the amount of columns causes it to extend all the way towards the end of the grid.

/6.2.4 **stroke** `none` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable* Default: `1pt + black`

The line's stroke.

Specifying `none` removes any lines previously placed across this line's range, including hlines or per-cell stroke below it.

/6.2.5 **position** `alignment` *Settable* Default: `top`

The position at which the line is placed, given its row (`y`) - either `top` to draw above it or `bottom` to draw below it.

This setting is only relevant when row gutter is enabled (and shouldn't be used otherwise - prefer just increasing the `y` field by one instead), since then the position below a row becomes different from the position above the next row due to the spacing between both.

/6.3 vline Element

A vertical line in the grid.

Overrides any per-cell stroke, including stroke specified through the grid's `stroke` field. Can cross spacing between cells created through the grid's `row-gutter` option.

```
grid.vline(
  x: auto int,
  start: int,
  end: none int,
  stroke: none length color gradient stroke tiling dictionary,
  position: alignment,
) → content
```

/6.3.1 **x** auto or int *Settable* Default: auto

The column before which the vertical line is placed (zero-indexed). If the position field is set to end, the line is placed after the column with the given index instead (see grid.vline.position[3.10.10/6.3.5] for details).

Specifying auto causes the line to be placed at the column after the last automatically positioned cell (that is, cell without coordinate overrides) before the line among the grid's children. If there is no such cell before the line, it is placed before the grid's first column (column 0). Note that specifying for this option exactly the total amount of columns in the grid causes this vertical line to override the end border of the grid (right in LTR, left in RTL), while a value of 0 overrides the start border (left in LTR, right in RTL).

/6.3.2 **start** int *Settable* Default: 0

The row at which the vertical line starts (zero-indexed, inclusive).

/6.3.3 **end** none or int *Settable* Default: none

The row on top of which the vertical line ends (zero-indexed, exclusive). Therefore, the vertical line will be drawn up to and across row end − 1.

A value equal to none or to the amount of rows causes it to extend all the way towards the bottom of the grid.

/6.3.4 **stroke** none or length or color or gradient or stroke or tiling or dictionary *Settable* Default: 1pt + black

The line's stroke.

Specifying none removes any lines previously placed across this line's range, including vlines or per-cell stroke below it.

/6.3.5 **position** alignment *Settable* Default: start

The position at which the line is placed, given its column (x) – either start to draw before it or end to draw after it.

The values left and right are also accepted, but discouraged as they cause your grid to be inconsistent between left-to-right and right-to-left documents.

This setting is only relevant when column gutter is enabled (and shouldn't be used otherwise – prefer just increasing the x field by one instead), since then the position after a column becomes different from the position before the next column due to the spacing between both.

/6.4 header Element

A repeatable grid header.

If repeat is set to true, the header will be repeated across pages. For an example, refer to the table.header[3.6.20/4.4] element and the grid.stroke[3.10.10/5.9] parameter.

```
grid.header(
  repeat: bool,
  level: int,
  ..content,
) → content
```

/6.4.1 **repeat** `bool` *Settable* Default: `true`

Whether this header should be repeated across pages.

/6.4.2 **level** `int` *Settable* Default: `1`

The level of the header. Must not be zero.

This allows repeating multiple headers at once. Headers with different levels can repeat together, as long as they have ascending levels.

Notably, when a header with a lower level starts repeating, all higher or equal level headers stop repeating (they are "replaced" by the new header).

/6.4.3 **children** `content` *Required* *Positional* *Variadic*

The cells and lines within the header.

/6.5 footer Element

A repeatable grid footer.

Just like the `grid.header`[3.10.10/6.4] element, the footer can repeat itself on every page of the grid.

No other grid cells may be placed after the footer.

```
grid.footer(
  repeat: bool,
  ..content,
) → content
```

/6.5.1 **repeat** `bool` *Settable* Default: `true`

Whether this footer should be repeated across pages.

/6.5.2 **children** `content` *Required* *Positional* *Variadic*

The cells and lines within the footer.

3.10.11 `hide` Element

Hides content without affecting layout.

The `hide` function allows you to hide content while the layout still "sees" it. This is useful for creating blank space that is exactly as large as some content.

/1 Example

```
Hello Jane \
#hide[Hello] Joe
```

Hello Jane
Joe

/2 Redaction

This function may also be useful for redacting content as its arguments are neither present visually nor accessible to Assistive Technology. That said, there can be *some* traces of the hidden content (such as a bookmarked heading in the PDF's Document Outline).

Note that, depending on the circumstances, it may be possible for content to be reverse engineered based on its size in the layout. We thus do not recommend using this function to hide highly sensitive information.

/3 Parameters

```
hide(content) → content
```

/3.1 **body** `content` *Required* *Positional*

The content to hide.

3.10.12 layout

Provides access to the current outer container's (or page's, if none) dimensions (width and height).

Accepts a function that receives a single parameter, which is a dictionary with keys `width` and `height`, both of type `length`[3.10.13]. The function is provided `context`[3.4], meaning you don't need to use it in combination with the `context` keyword. This is why `measure`[3.10.14] can be called in the example below.

```
#let text = lorem(30)
#layout(size => [
  #let (height,) = measure(
    width: size.width,
    text,
  )
  This text is #height high with
  the current page width: \
  #text
])
```

This text is 64.79pt high with the current page width:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri.

Note that the `layout` function forces its contents into a `block`[3.10.4]-level container, so placement relative to the page or pagebreaks are not possible within it.

If the `layout` call is placed inside a box with a width of `800pt` and a height of `400pt`, then the specified function will be given the argument `(width: 800pt, height: 400pt)`. If it is placed directly into the page, it receives the page's dimensions minus its margins. This is mostly useful in combination with `measurement`[3.10.14].

To retrieve the *remaining* height of the page rather than its full size, you can wrap your `layout` call in a `block(height: 1fr)`. This works because the block automatically grows to fill the remaining space (see the `fraction`[3.10.9] documentation for more details).

```
#set page(height: 150pt)

#lorem(20)

#block(height: 1fr, layout(size => [
  Remaining height: #size.height
]))
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Remaining height: 56.4pt

You can also use this function to resolve a `ratio`[3.10.20] to a fixed length. This might come in handy if you're building your own layout abstractions.

```
#layout(size => {
  let half = 50% * size.width
  [Half a page is #half wide.]
})
```

Half a page is 105pt wide.

Note that the width or height provided by `layout` will be infinite if the corresponding page dimension is set to `auto`.

/1 Parameters

```
layout(function) → content
```

/1.1 func `function` *Required* *Positional*

A function to call with the outer container's size. Its return value is displayed in the document.

The container's size is given as a dictionary[3.5.11] with the keys `width` and `height`, both of type `length`[3.10.13].

This function is called once for each time the content returned by `layout` appears in the document. This makes it possible to generate content that depends on the dimensions of its container.

3.10.13 `length`

A size or distance, possibly expressed with contextual units.

Typst supports the following length units:

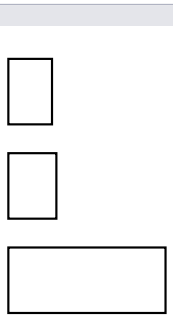
- Points: `72pt`
- Millimeters: `254mm`
- Centimeters: `2.54cm`
- Inches: `1in`
- Relative to font size: `2.5em`

You can multiply lengths with and divide them by integers and floats.

/1 Example

```
#rect(width: 20pt)
#rect(width: 2em)
#rect(width: 1in)

#(3em + 5pt).em \
#(20pt).em \
#(40em + 2pt).abs \
#(5em).abs
```



3
0
2pt
0pt

/2 Fields

- `abs`: A length with just the absolute component of the current length (that is, excluding the `em` component).
- `em`: The amount of `em` units in this length, as a float[3.5.14].

/3 Definitions

/3.1 `pt`

Converts this length to points.

Fails with an error if this length has non-zero `em` units (such as `5em + 2pt` instead of just `2pt`). Use the `abs` field (such as in `(5em + 2pt).abs.pt()`) to ignore the `em` component of the length (thus converting only its absolute component).

```
self.pt() → float
```

/3.2 `mm`

Converts this length to millimeters.

Fails with an error if this length has non-zero `em` units. See the pt[3.10.13/3.1] method for more details.

```
self.mm() → float
```

/3.3 `cm`

Converts this length to centimeters.

Fails with an error if this length has non-zero `em` units. See the pt[3.10.13/3.1] method for more details.

```
self.cm() → float
```

/3.4 inches

Converts this length to inches.

Fails with an error if this length has non-zero `em` units. See the pt[3.10.13/3.1] method for more details.

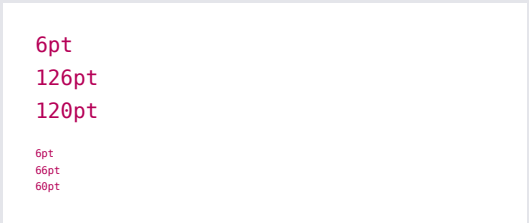
```
self.inches() → float
```

/3.5 to-absolute

Resolve this length to an absolute length.

```
#set text(size: 12pt)
#context [
  #(6pt).to-absolute() \
  #(6pt + 10em).to-absolute() \
  #(10em).to-absolute()
]

#set text(size: 6pt)
#context [
  #(6pt).to-absolute() \
  #(6pt + 10em).to-absolute() \
  #(10em).to-absolute()
]
```




```
self.to-absolute() → length
```

3.10.14 `measure` Contextual

Measures the layouted size of content.

The `measure` function lets you determine the layouted size of content. By default an infinite space is assumed, so the measured dimensions may not necessarily match the final dimensions of the content. If you want to measure in the current layout dimensions, you can combine `measure` and `layout`[3.10.12].

/1 Example

The same content can have a different size depending on the context[3.4] that it is placed into. In the example below, the `#content` is of course bigger when we increase the font size.

```
#let content = [Hello!]
#content
#set text(14pt)
#content
```

Hello! Hello!

For this reason, you can only measure when context is available.

```
#let thing(body) = context {
  let size = measure(body)
  [Width of "#body" is #size.width]
}

#thing[Hey] \
#thing[Welcome]
```

Width of “Hey” is 18.53pt
Width of “Welcome” is 41.26pt

The measure function returns a dictionary with the entries `width` and `height`, both of type `length`[3.10.13].

/2 Parameters

```
measure(
  width: auto length,
  height: auto length,
  content,
) → dictionary
```

/2.1 `width` `auto` or `length` — Default: `auto`

The width available to layout the content.

Setting this to `auto` indicates infinite available width.

Note that using the `width` and `height` parameters of this function is different from measuring a sized `block`[3.10.4] containing the content. In the following example, the former will get the dimensions of the inner content instead of the dimensions of the block.

```
#context measure(lorem(100), width: 400pt)

#context measure(block(lorem(100), width: 400pt))
```

```
(width: 395.89pt, height: 122.34pt)
(width: 400pt, height: 122.34pt)
```

/2.2 `height` `auto` or `length` — Default: `auto`

The height available to layout the content.

Setting this to `auto` indicates infinite available height.

/2.3 **content** `content` *Required* *Positional*

The content whose size to measure.

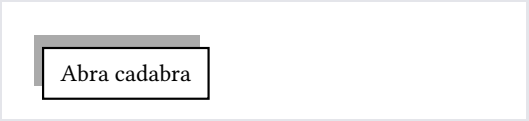
3.10.15 `move` Element

Moves content without affecting layout.

The `move` function allows you to move content while the layout still 'sees' it at the original positions. Containers will still be sized as if the content was not moved.

/1 Example

```
#rect(inset: 0pt, fill: gray, move(
  dx: 4pt, dy: 6pt,
  rect(
    inset: 8pt,
    fill: white,
    stroke: black,
    [Abra cadabra]
  )
))
```



/2 Accessibility

Moving is transparent to Assistive Technology (AT). Your content will be read in the order it appears in the source, regardless of any visual movement. If you need to hide content from AT altogether in PDF export, consider using `pdf.artifact`[3.14.4].

/3 Parameters

```
move(
  dx: relative,
  dy: relative,
  content,
) → content
```

/3.1 **dx** `relative` *Settable* Default: `0% + 0pt`

The horizontal displacement of the content.

/3.2 **dy** `relative` *Settable* Default: `0% + 0pt`

The vertical displacement of the content.

/3.3 **body** `content` *Required* *Positional*

The content to move.

3.10.16 pad Element

Adds spacing around content.

The spacing can be specified for each side individually, or for all sides at once by specifying a positional argument.

/1 Example

```
#set align(center)

#pad(x: 16pt, image("typing.jpg"))
_Typing speeds can be
 measured in words per minute._
```



Typing speeds can be measured in words per minute.

/2 Parameters

```
pad(
  left: relative,
  top: relative,
  right: relative,
  bottom: relative,
  x: relative,
  y: relative,
  rest: relative,
  content,
) → content
```

/2.1 **left** `relative` *Settable* Default: `0% + 0pt`

The padding at the left side.

/2.2 **top** `relative` *Settable* Default: `0% + 0pt`

The padding at the top side.

/2.3 **right** `relative` *Settable* Default: `0% + 0pt`

The padding at the right side.

/2.4 **bottom** `relative` *Settable* Default: `0% + 0pt`

The padding at the bottom side.

/2.5 **x** `relative` Default: `0% + 0pt`

A shorthand to set `left` and `right` to the same value.

/2.6 **y** `relative` Default: `0% + 0pt`

A shorthand to set `top` and `bottom` to the same value.

/2.7 **rest** `relative` Default: `0% + 0pt`

A shorthand to set all four sides to the same value.

/2.8 **body** `content` *Required* *Positional*

The content to pad at the sides.

3.10.17 page Element

Layouts its child onto one or multiple pages.

Although this function is primarily used in set rules to affect page properties, it can also be used to explicitly render its argument onto a set of pages of its own.

Pages can be set to use `auto` as their width or height. In this case, the pages will grow to fit their content on the respective axis.

The Guide for Page Setup[4.2] explains how to use this and related functions to set up a document with many examples.

/1 Example

```
#set page("us-letter")

There you go, US friends!
```


/2 Accessibility

The contents of the page's header, footer, foreground, and background are invisible to Assistive Technology (AT) like screen readers. Only the body of the page is read by AT. Do not include vital information not included elsewhere in the document in these areas.

/3 Styling

Note that the page[3.10.17] element cannot be targeted by show rules; writing `show page: ..` has no effect. To repeat content on every page, you can instead configure the header[3.10.17/4.13], footer[3.10.17/4.15], background[3.10.17/4.17], and foreground[3.10.17/4.18] properties with a set rule.

/4 Parameters

```
page(
  paper: str,
  width: auto length,
  height: auto length,
  flipped: bool,
  margin: auto relative dictionary,
  bleed: relative dictionary,
```

```
  binding: auto alignment,
  columns: int,
  fill: none auto color gradient tiling,
  numbering: none str function,
  supplement: none auto content,
  number-align: alignment,
  header: none auto content,
  header-ascent: relative,
  footer: none auto content,
  footer-descent: relative,
  background: none content,
  foreground: none content,
  body: content,
) → content
```

/4.1 **paper** `str` Default: `"a4"`

A standard paper size to set width and height.

This is just a shorthand for setting `width` and `height` and, as such, cannot be retrieved in a context expression.

Variant	Details
`"a0"`	A paper that is 841.0 × 1189.0 millimeters in size.
`"a1"`	A paper that is 594.0 × 841.0 millimeters in size.
`"a2"`	A paper that is 420.0 × 594.0 millimeters in size.
`"a3"`	A paper that is 297.0 × 420.0 millimeters in size.
`"a4"`	A paper that is 210.0 × 297.0 millimeters in size.
`"a5"`	A paper that is 148.0 × 210.0 millimeters in size.
`"a6"`	A paper that is 105.0 × 148.0 millimeters in size.
`"a7"`	A paper that is 74.0 × 105.0 millimeters in size.
`"a8"`	A paper that is 52.0 × 74.0 millimeters in size.
`"a9"`	A paper that is 37.0 × 52.0 millimeters in size.
`"a10"`	A paper that is 26.0 × 37.0 millimeters in size.
`"a11"`	A paper that is 18.0 × 26.0 millimeters in size.
`"iso-b1"`	A paper that is 707.0 × 1000.0 millimeters in size.
`"iso-b2"`	A paper that is 500.0 × 707.0 millimeters in size.
`"iso-b3"`	A paper that is 353.0 × 500.0 millimeters in size.
`"iso-b4"`	A paper that is 250.0 × 353.0 millimeters in size.
`"iso-b5"`	A paper that is 176.0 × 250.0 millimeters in size.
`"iso-b6"`	A paper that is 125.0 × 176.0 millimeters in size.
`"iso-b7"`	A paper that is 88.0 × 125.0 millimeters in size.
`"iso-b8"`	A paper that is 62.0 × 88.0 millimeters in size.
`"iso-c3"`	A paper that is 324.0 × 458.0 millimeters in size.
`"iso-c4"`	A paper that is 229.0 × 324.0 millimeters in size.
`"iso-c5"`	A paper that is 162.0 × 229.0 millimeters in size.
`"iso-c6"`	A paper that is 114.0 × 162.0 millimeters in size.
`"iso-c7"`	A paper that is 81.0 × 114.0 millimeters in size.
`"iso-c8"`	A paper that is 57.0 × 81.0 millimeters in size.
`"din-d3"`	A paper that is 272.0 × 385.0 millimeters in size.
`"din-d4"`	A paper that is 192.0 × 272.0 millimeters in size.
`"din-d5"`	A paper that is 136.0 × 192.0 millimeters in size.

Variant	Details
`"din-d6"`	A paper that is 96.0 × 136.0 millimeters in size.
`"din-d7"`	A paper that is 68.0 × 96.0 millimeters in size.
`"din-d8"`	A paper that is 48.0 × 68.0 millimeters in size.
`"sis-g5"`	A paper that is 169.0 × 239.0 millimeters in size.
`"sis-e5"`	A paper that is 115.0 × 220.0 millimeters in size.
`"ansi-a"`	A paper that is 216.0 × 279.0 millimeters in size.
`"ansi-b"`	A paper that is 279.0 × 432.0 millimeters in size.
`"ansi-c"`	A paper that is 432.0 × 559.0 millimeters in size.
`"ansi-d"`	A paper that is 559.0 × 864.0 millimeters in size.
`"ansi-e"`	A paper that is 864.0 × 1118.0 millimeters in size.
`"arch-a"`	A paper that is 229.0 × 305.0 millimeters in size.
`"arch-b"`	A paper that is 305.0 × 457.0 millimeters in size.
`"arch-c"`	A paper that is 457.0 × 610.0 millimeters in size.
`"arch-d"`	A paper that is 610.0 × 914.0 millimeters in size.
`"arch-e1"`	A paper that is 762.0 × 1067.0 millimeters in size.
`"arch-e"`	A paper that is 914.0 × 1219.0 millimeters in size.
`"jis-b0"`	A paper that is 1030.0 × 1456.0 millimeters in size.
`"jis-b1"`	A paper that is 728.0 × 1030.0 millimeters in size.
`"jis-b2"`	A paper that is 515.0 × 728.0 millimeters in size.
`"jis-b3"`	A paper that is 364.0 × 515.0 millimeters in size.
`"jis-b4"`	A paper that is 257.0 × 364.0 millimeters in size.
`"jis-b5"`	A paper that is 182.0 × 257.0 millimeters in size.
`"jis-b6"`	A paper that is 128.0 × 182.0 millimeters in size.
`"jis-b7"`	A paper that is 91.0 × 128.0 millimeters in size.
`"jis-b8"`	A paper that is 64.0 × 91.0 millimeters in size.
`"jis-b9"`	A paper that is 45.0 × 64.0 millimeters in size.
`"jis-b10"`	A paper that is 32.0 × 45.0 millimeters in size.
`"jis-b11"`	A paper that is 22.0 × 32.0 millimeters in size.
`"sac-d0"`	A paper that is 764.0 × 1064.0 millimeters in size.
`"sac-d1"`	A paper that is 532.0 × 760.0 millimeters in size.
`"sac-d2"`	A paper that is 380.0 × 528.0 millimeters in size.
`"sac-d3"`	A paper that is 264.0 × 376.0 millimeters in size.
`"sac-d4"`	A paper that is 188.0 × 260.0 millimeters in size.
`"sac-d5"`	A paper that is 130.0 × 184.0 millimeters in size.
`"sac-d6"`	A paper that is 92.0 × 126.0 millimeters in size.
`"iso-id-1"`	A paper that is 85.6 × 53.98 millimeters in size.
`"iso-id-2"`	A paper that is 74.0 × 105.0 millimeters in size.
`"iso-id-3"`	A paper that is 88.0 × 125.0 millimeters in size.
`"asia-f4"`	A paper that is 210.0 × 330.0 millimeters in size.
`"jp-shiroku-ban-4"`	A paper that is 264.0 × 379.0 millimeters in size.
`"jp-shiroku-ban-5"`	A paper that is 189.0 × 262.0 millimeters in size.
`"jp-shiroku-ban-6"`	A paper that is 127.0 × 188.0 millimeters in size.
`"jp-kiku-4"`	A paper that is 227.0 × 306.0 millimeters in size.
`"jp-kiku-5"`	A paper that is 151.0 × 227.0 millimeters in size.
`"jp-business-card"`	A paper that is 91.0 × 55.0 millimeters in size.
`"cn-business-card"`	A paper that is 90.0 × 54.0 millimeters in size.
`"eu-business-card"`	A paper that is 85.0 × 55.0 millimeters in size.

Variant	Details
`"fr-tellière"`	A paper that is 340.0 × 440.0 millimeters in size.
`"fr-couronne-écriture"`	A paper that is 360.0 × 460.0 millimeters in size.
`"fr-couronne-édition"`	A paper that is 370.0 × 470.0 millimeters in size.
`"fr-raisin"`	A paper that is 500.0 × 650.0 millimeters in size.
`"fr-carré"`	A paper that is 450.0 × 560.0 millimeters in size.
`"fr-jésus"`	A paper that is 560.0 × 760.0 millimeters in size.
`"uk-brief"`	A paper that is 406.4 × 342.9 millimeters in size.
`"uk-draft"`	A paper that is 254.0 × 406.4 millimeters in size.
`"uk-foolscap"`	A paper that is 203.2 × 330.2 millimeters in size.
`"uk-quarto"`	A paper that is 203.2 × 254.0 millimeters in size.
`"uk-crown"`	A paper that is 508.0 × 381.0 millimeters in size.
`"uk-book-a"`	A paper that is 111.0 × 178.0 millimeters in size.
`"uk-book-b"`	A paper that is 129.0 × 198.0 millimeters in size.
`"us-letter"`	A paper that is 215.9 × 279.4 millimeters in size.
`"us-legal"`	A paper that is 215.9 × 355.6 millimeters in size.
`"us-tabloid"`	A paper that is 279.4 × 431.8 millimeters in size.
`"us-executive"`	A paper that is 184.15 × 266.7 millimeters in size.
`"us-foolscap-folio"`	A paper that is 215.9 × 342.9 millimeters in size.
`"us-statement"`	A paper that is 139.7 × 215.9 millimeters in size.
`"us-ledger"`	A paper that is 431.8 × 279.4 millimeters in size.
`"us-oficio"`	A paper that is 215.9 × 340.36 millimeters in size.
`"us-gov-letter"`	A paper that is 203.2 × 266.7 millimeters in size.
`"us-gov-legal"`	A paper that is 215.9 × 330.2 millimeters in size.
`"us-business-card"`	A paper that is 88.9 × 50.8 millimeters in size.
`"us-digest"`	A paper that is 139.7 × 215.9 millimeters in size.
`"us-trade"`	A paper that is 152.4 × 228.6 millimeters in size.
`"newspaper-compact"`	A paper that is 280.0 × 430.0 millimeters in size.
`"newspaper-berliner"`	A paper that is 315.0 × 470.0 millimeters in size.
`"newspaper-broadsheet"`	A paper that is 381.0 × 578.0 millimeters in size.
`"presentation-16-9"`	A paper that is 297.0 × 167.0625 millimeters in size.
`"presentation-4-3"`	A paper that is 280.0 × 210.0 millimeters in size.

/4.2 **`width`** `auto` or `length` *Settable* Default: `595.28pt`

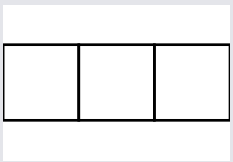
The width of the final page, after any trims have been applied.

In professional printing setups, this may be smaller than the sheet size fed into the printer.

See the `bleed`[3.10.17/4.6] parameter for details on how to set a trim or bleed area.

```
#set page(
  width: 3cm,
  margin: (x: 0cm),
)

#for i in range(3) {
  box(square(width: 1cm))
}
```



/4.3 **`height`** `auto` or `length` *Settable* Default: `841.89pt`

The height of the final page area, after any trims have been applied.

If this is set to `auto`, page breaks can only be triggered manually by inserting a page break[3.10.18] or by adding another non-empty page set rule. Most examples throughout this documentation use `auto` for the height of the page to dynamically grow and shrink to fit their content.

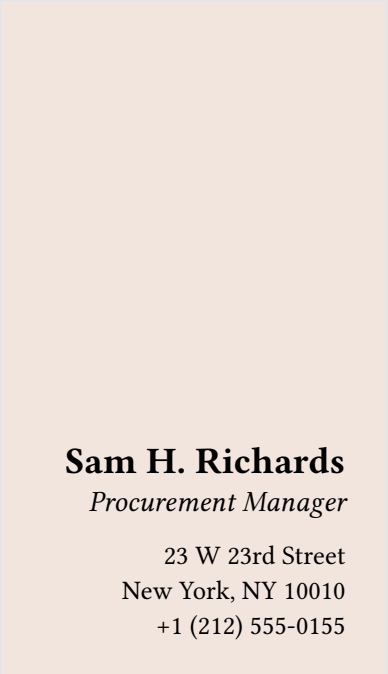
/4.4 flipped `bool` *Settable* Default: `false`

Whether the page is flipped into landscape orientation.

```
#set page(
  "us-business-card",
  flipped: true,
  fill: rgb("f2e5dd"),
)

#set align(bottom + end)
#text(14pt)[*Sam H. Richards*] \
_Procurement Manager_

#set text(10pt)
23 W 23rd Street \
New York, NY 10010 \
+1 (212) 555-0155
```



/4.5 margin `auto` or `relative` or `dictionary` *Settable* Default: `auto`

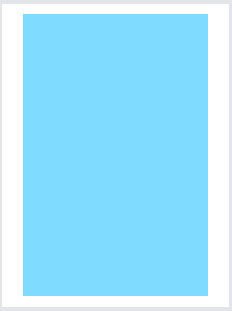
The page's margins.

- `auto`: The margins are set automatically to 2.5/21 times the smaller dimension of the page. This results in 2.5 cm margins for an A4 page.
- A single length: The same margin on all sides.
- A dictionary: With a dictionary, the margins can be set individually. The dictionary can contain the following keys in order of precedence:
  - `top`: The top margin.
  - `right`: The right margin.
  - `bottom`: The bottom margin.
  - `left`: The left margin.
  - `inside`: The margin at the inner side of the page (where the binding[3.10.17/4.7] is).
  - `outside`: The margin at the outer side of the page (opposite to the binding[3.10.17/4.7]).
  - `x`: The horizontal margins.
  - `y`: The vertical margins.
  - `rest`: The margins on all sides except those for which the dictionary explicitly sets a size.

All keys are optional; omitted keys will use their previously set value, or the default margin if never set. In addition, the values for `left` and `right` are mutually exclusive with the values for `inside` and `outside`. The values should be relative lengths or `auto`.

```
#set page(
 width: 3cm,
 height: 4cm,
 margin: (x: 8pt, y: 4pt),
)

#rect(
  width: 100%,
  height: 100%,
  fill: aqua,
)
```



/4.6 **bleed** `relative` or `dictionary` *Settable* Default: `(:)`

The page's bleed margin.

The bleed is the area of content that extends beyond the final trimmed size of the page. It ensures that no unprinted edges appear in the final product, even if minor trimming misalignments occur.

Accepted values:

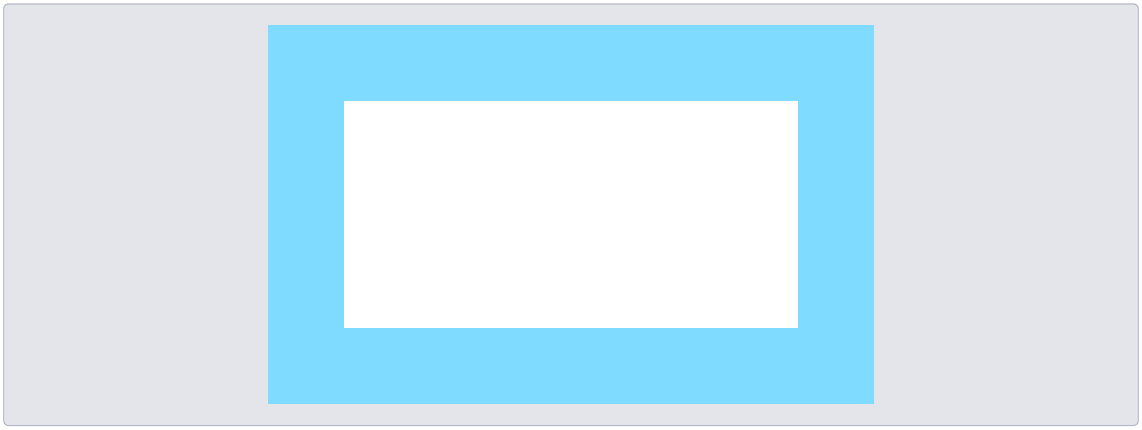
- A single length: The same bleed on all sides.
- A dictionary: With a dictionary, the bleed margins can be set individually. The dictionary may include the following keys, listed in order of precedence:
  - `top`: The top bleed margin.
  - `right`: The right bleed margin.
  - `bottom`: The bottom bleed margin.
  - `left`: The left bleed margin.
  - `inside`: The bleed margin at the inner side of the page (where the binding[3.10.17/4.7] is).
  - `outside`: The bleed margin at the outer side of the page (opposite to the binding[3.10.17/4.7]).
  - `x`: The horizontal bleed margins.
  - `y`: The vertical bleed margins.
  - `rest`: The bleed margins on all sides except those for which the dictionary explicitly sets a size.

All keys are optional; omitted keys will use their previously set value, or `0pt` if never set. In addition, the values for `left` and `right` are mutually exclusive with the values for `inside` and `outside`. The values should be relative lengths.

In PDF export, if the bleed is non-zero, a `TrimBox` is defined for the page.

```
#set page(
  width: 8cm,
  height: 5cm,
  margin: 1cm,
  // The bleed is not visible in the preview;
  // it exceeds beyond the page.
  bleed: 0.5cm,
  // Fills the entire bleed area, so there will
  // be no white strips after printing and trimming.
  background: rect(width: 100%, height: 100%, fill: aqua),
)

#rect(width: 100%, height: 100%, fill: white)
```



/4.7 **binding** `auto` or `alignment` *Settable* Default: `auto`

On which side the pages will be bound.

- `auto`: Equivalent to `left` if the text direction[3.7.12/2.19] is left-to-right and `right` if it is right-to-left.
- `left`: Bound on the left side.
- `right`: Bound on the right side.

This affects the meaning of the `inside` and `outside` options for margins.

/4.8 **columns** `int` *Settable* Default: `1`

How many columns the page has.

If you need to insert columns into a page or other container, you can also use the `columns` function[3.10.7].

```
#set page(columns: 2, height: 4.8cm)
Climate change is one of the most
pressing issues of our time, with
the potential to devastate
communities, ecosystems, and
economies around the world. It's
clear that we need to take urgent
action to reduce our carbon
emissions and mitigate the impacts
of a rapidly changing climate.
```

Climate change is one of the most pressing issues of our time, with the potential to devastate communities, ecosystems, and economies around the world. It's clear that we need to take urgent action to reduce our carbon emissions and mitigate the impacts

/4.9 **fill** `none` or `auto` or `color` or `gradient` or `tiling` *Settable* Default: `auto`

The page's background fill.

Setting this to something non-transparent instructs the printer to color the complete page. If you are considering larger production runs, it may be more environmentally friendly and cost-effective to source pre-dyed pages and not set this property.

When set to `none`, the background becomes transparent. Note that PDF pages will still appear with a (usually white) background in viewers, but they are actually transparent. (If you print them, no color is used for the background.)

The default of `auto` results in `none` for PDF output, and `white` for PNG and SVG.

```
#set page(fill: rgb("444352"))
#set text(fill: rgb("fdfdfd"))
*Dark mode enabled.*
```

Dark mode enabled.

/4.10 **numbering** `none` or `str` or `function` *Settable* Default: `none`

How to number the pages. You can refer to the Page Setup Guide for customizing page numbers[4.2.3].

Accepts a numbering pattern or function[3.6.13] taking one or two numbers:

1. The first number is the current page number.

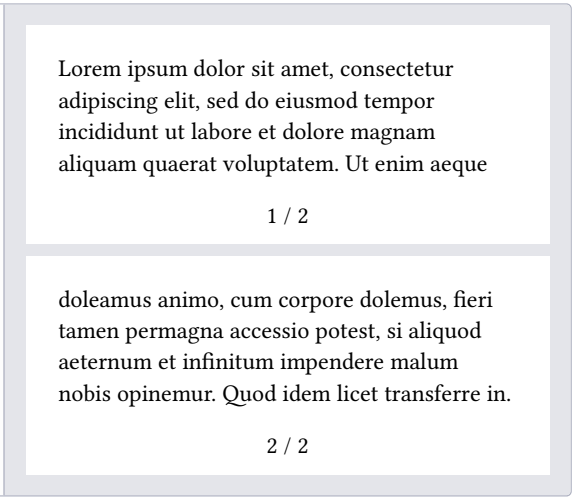
2. The second number is the total number of pages. In a numbering pattern, the second number can be omitted. If a function is passed, it will receive one argument in the context of links or references, and two arguments when producing the visible page numbers.

These are logical numbers controlled by the page counter, and may thus not match the physical numbers. Specifically, they are the current[3.12.1/11.1] and the final[3.12.1/11.4] value of `counter(page)`. See the `counter`[3.12.1/5] documentation for more details.

If an explicit `footer`[3.10.17/4.15] (or `header`[3.10.17/4.13] for top-aligned[3.10.17/4.12] numbering) is given, the numbering is ignored.

```
#set page(
  height: 100pt,
  margin: (top: 16pt, bottom: 24pt),
  numbering: "1 / 1",
)

#lorem(48)
```



/4.11 `supplement` `none` or `auto` or `content` *Settable* Default: `auto`

A supplement for the pages.

For page references, this is added before the page number.

```
#set page(numbering: "1.", supplement: [p.])

= Introduction <intro>
We are on #ref(<intro>, form: "page")!
```



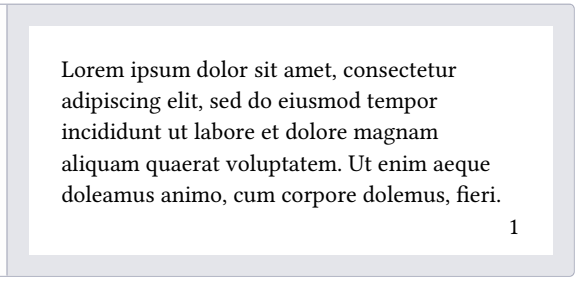
/4.12 `number-align` `alignment` *Settable* Default: `center + bottom`

The alignment of the page numbering.

If the vertical component is `top`, the numbering is placed into the header and if it is `bottom`, it is placed in the footer. Horizon alignment is forbidden. If an explicit matching `header` or `footer` is given, the numbering is ignored.

```
#set page(
  margin: (top: 16pt, bottom: 24pt),
  numbering: "1",
  number-align: right,
)

#lorem(30)
```



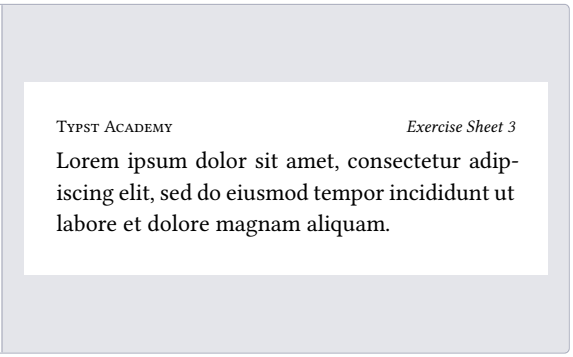
/4.13 **header** `none` or `auto` or `content` *Settable* Default: `auto`

The page's header. Fills the top margin of each page.

– Content: Shows the content as the header.
– `auto`: Shows the page number if a `numbering`[3.10.17/4.10] is set and `number-align`[3.10.17/4.12] is `top`.
– `none`: Suppresses the header.

```
#set par(justify: true)
#set page(
  margin: (top: 32pt, bottom: 20pt),
  header: [
    #set text(8pt)
    #smallcaps[Typst Academy]
    #h(1fr) _Exercise Sheet 3_
  ],
)

#lorem(19)
```



/4.14 **header-ascent** `relative` *Settable* Default: `30% + 0pt`

The amount the header is raised into the top margin. Ratios are relative to the height of the top margin.

/4.15 **footer** `none` or `auto` or `content` *Settable* Default: `auto`

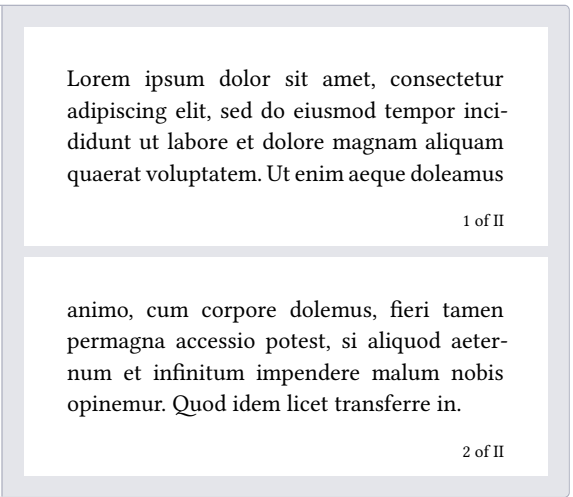
The page's footer. Fills the bottom margin of each page.

– Content: Shows the content as the footer.
– `auto`: Shows the page number if a `numbering`[3.10.17/4.10] is set and `number-align`[3.10.17/4.12] is `bottom`.
– `none`: Suppresses the footer.

For just a page number, the `numbering` property typically suffices. If you want to create a custom footer but still display the page number, you can directly access the page counter[3.12.1].

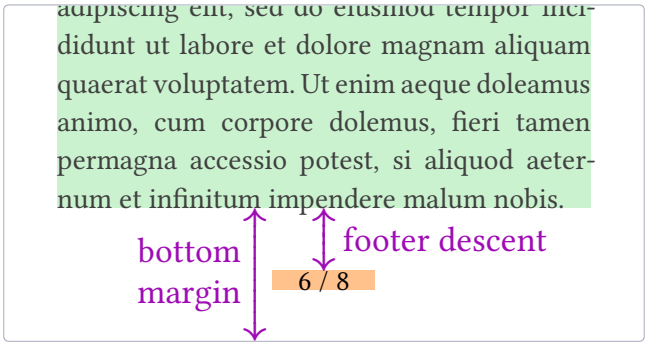
```
#set par(justify: true)
#set page(
  height: 100pt,
  margin: 20pt,
  footer: context [
    #set align(right)
    #set text(8pt)
    #counter(page).display(
      "1 of I",
      both: true,
    )
  ]
)

#lorem(48)
```



/4.16 **footer-descent** `relative` *Settable* Default: `30% + 0pt`

The amount the footer is lowered into the bottom margin. Ratios are relative to the height of the bottom margin.



/4.17 **background** `none` or `content` *Settable* Default: `none`

Content in the page's background.

This content will be placed behind the page's body. It can be used to place a background image or a watermark.

For convenience, relative lengths[3.10.21] are resolved against the page size including the page `bleed`[3.10.17/4.6] when used in background content. For example, on a page that is `100mm` wide with a `5mm` bleed, a width of `100%` is computed as `5mm + 100mm + 5mm`.

```
#set page(background: rotate(24deg,
  text(18pt, fill: rgb("FFCBC4"))[
    *CONFIDENTIAL*
  ]
))

= Typst's secret plans
In the year 2023, we plan to take
over the world (of typesetting).
```



/4.18 **foreground** `none` or `content` *Settable* Default: `none`

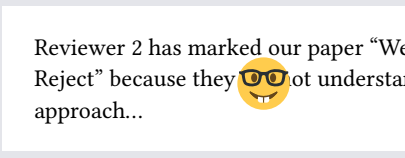
Content in the page's foreground.

This content will overlay the page's body.

Relative lengths are resolved against the page size including `bleed`[3.10.17/4.6], following the same behavior as `background`[3.10.17/4.17].

```
#set page(foreground: text(24pt)[🤓])

Reviewer 2 has marked our paper
"Weak Reject" because they did
not understand our approach...
```



/4.19 **body** `content` Default: `[]`

The contents of the page(s).

Multiple pages will be created if the content does not fit on a single page. A new page with the page properties prior to the function invocation will be created after the body has been typeset.

3.10.18 pagebreak Element

A manual page break.

Must not be used inside any containers.

/1 Example

```
The next page contains
more details on compound theory.
#pagebreak()

== Compound Theory
In 1984, the first ...
```

The next page contains more details on compound theory.

Compound Theory

In 1984, the first ...

Even without manual page breaks, content will be automatically paginated based on the configured page size. You can set the page height[3.10.17/4.3] to `auto` to let the page grow dynamically until a manual page break occurs.

Pagination tries to avoid single lines of text at the top or bottom of a page (these are called *widows* and *orphans*). You can adjust the `text.costs`[3.7.12/2.21] parameter to disable this behavior.

/2 Parameters

```
pagebreak(
  weak: bool,
  to: none str,
) → content
```

/2.1 weak `bool` *Settable* Default: `false`

If `true`, the page break is skipped if the current page is already empty.

/2.2 to `none` or `str` *Settable* Default: `none`

If given, ensures that the next page will be an even/odd page, with an empty page in between if necessary.

```
#set page(height: 30pt)

First.
#pagebreak(to: "odd")
Third.
```

First.

Third.

Variant	Details
`"even"`	Next page will be an even page.
`"odd"`	Next page will be an odd page.

3.10.19 `place` Element

Places content relatively to its parent container.

Placed content can be either overlaid (the default) or floating. Overlaid content is aligned with the parent container according to the given `alignment`[3.10.19/4.1], and shown over any other content added so far in the container. Floating content is placed at the top or bottom of the container, displacing other content down or up respectively. In both cases, the content position can be adjusted with `dx`[3.10.19/4.5] and `dy`[3.10.19/4.6] offsets without affecting the layout.

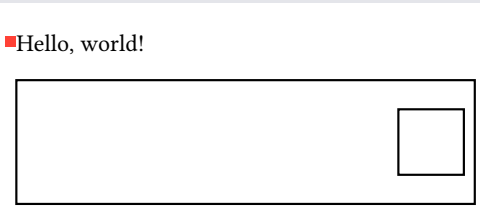
The parent can be any container such as a `block`[3.10.4], `box`[3.10.5], `rect`[3.11.10], etc. A top level `place` call will place content directly in the text area of the current page. This can be used for absolute positioning on the page: with a `top + left` `alignment`[3.10.19/4.1], the offsets `dx` and `dy` will set the position of the element's top left corner relatively to the top left corner of the text area. For absolute positioning on the full page including margins, you can use `place` in `page.foreground`[3.10.17/4.18] or `page.background`[3.10.17/4.17].

/1 Examples

```
#set page(height: 120pt)
Hello, world!

#rect(
  width: 100%,
  height: 2cm,
  place(horizon + right, square()),
)

#place(
  top + left,
  dx: -5pt,
  square(size: 5pt, fill: red),
)
```



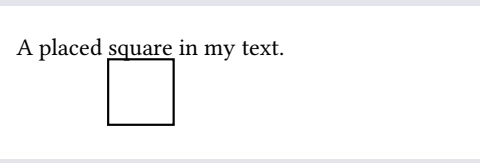
/2 Effect on the position of other elements

Overlaid elements don't take space in the flow of content, but a `place` call inserts an invisible block-level element in the flow. This can affect the layout by breaking the current paragraph. To avoid this, you can wrap the `place` call in a `box`[3.10.5] when the call is made in the middle of a paragraph. The alignment and offsets will then be relative to this zero-size box. To make sure it doesn't interfere with spacing, the box should be attached to a word using a word joiner.

For example, the following defines a function for attaching an annotation to the following word:

```
#let annotate(..args) = {
  box(place(..args))
  sym.wj
  h(0pt, weak: true)
}

A placed #annotate(square(), dy: 2pt)
square in my text.
```



The zero-width weak spacing serves to discard spaces between the function call and the next word.

/3 Accessibility

Assistive Technology (AT) will always read the placed element at the point where it logically appears in the document, regardless of where this function physically moved it. Put its markup where it would make the most sense in the reading order.

/4 Parameters

```
place(
  auto alignment,
  scope: str,
  float: bool,
  clearance: length,
  dx: relative,
  dy: relative,
  content,
) → content
```

/4.1 alignment — auto or alignment — *Positional* *Settable* — Default: start

Relative to which position in the parent container to place the content.

– If `float` is `false`, then this can be any alignment other than `auto`.
– If `float` is `true`, then this must be `auto`, `top`, or `bottom`.

When `float` is `false` and no vertical alignment is specified, the content is placed at the current position on the vertical axis.

/4.2 scope — str — *Settable* — Default: "column"

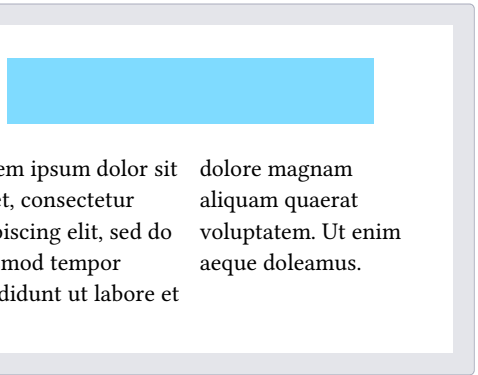
Relative to which containing scope something is placed.

The parent scope is primarily used with figures and, for this reason, the figure function has a mirrored `scope` parameter[3.6.8/5.4]. Nonetheless, it can also be more generally useful to break out of the columns. A typical example would be to create a single-column title section[4.2.4] in a two-column document.

Note that parent-scoped placement is currently only supported if `float` is `true`. This may change in the future.

```
#set page(height: 150pt, columns: 2)
#place(
  top + center,
  scope: "parent",
  float: true,
  rect(width: 80%, fill: aqua),
)

#lorem(25)
```



Variant	Details
"column"	Place into the current column.
"parent"	Place relative to the parent, letting the content span over all columns.

/4.3 float — bool — *Settable* — Default: false

Whether the placed element has floating layout.

Floating elements are positioned at the top or bottom of the parent container, displacing in-flow content. They are always placed in the in-flow order relative to each other, as well as before any content following a later `place.flush`[3.10.19/5.1] element.

```
#set page(height: 150pt)
#let note(where, body) = place(
  center + where,
  float: true,
  clearance: 6pt,
  rect(body),
)

#lorem(10)
#note(bottom)[Bottom 1]
#note(bottom)[Bottom 2]
#lorem(40)
#note(top)[Top]
#lorem(10)
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam

Bottom 1

Bottom 2

Top

aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

/4.4 **clearance** `length` *Settable* Default: `1.5em`

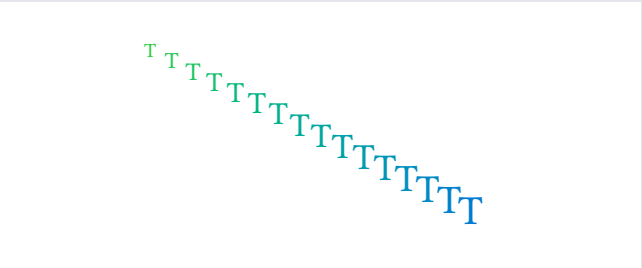
The spacing between the placed element and other elements in a floating layout.

Has no effect if `float` is `false`.

/4.5 **dx** `relative` *Settable* Default: `0% + 0pt`

The horizontal displacement of the placed content.

```
#set page(height: 100pt)
#for x in range(-8, 8) {
  place(center + horizon,
    dx: x * 8pt, dy: x * 4pt,
    text(
      size: calc.root(x + 10, 3) * 6pt,
      fill: color.mix((green, 8 - x), (blue, 8 + x)),
    )[T]
  )
}
```



This does not affect the layout of in-flow content. In other words, the placed content is treated as if it were wrapped in a `move`[3.10.15] element.

/4.6 **dy** `relative` *Settable* Default: `0% + 0pt`

The vertical displacement of the placed content.

This does not affect the layout of in-flow content. In other words, the placed content is treated as if it were wrapped in a `move`[3.10.15] element.

/4.7 **body** `content` *Required* *Positional*

The content to place.

/5 Definitions

/5.1 `flush` Element

Asks the layout algorithm to place pending floating elements before continuing with the content.

This is useful for preventing floating figures from spilling into the next section.

```
#lorem(15)

#figure(
  rect(width: 100%, height: 50pt),
  placement: auto,
  caption: [A rectangle],
)

#place.flush()

This text appears after the figure.
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.



Figure 1: A rectangle

This text appears after the figure.

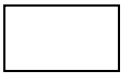
`place.flush() → content`

3.10.20 # `ratio`

A ratio of a whole.

A ratio is written as a number, followed by a percent sign. Ratios most often appear as part of a relative length[3.10.21], to specify the size of some layout element relative to the page or some container.

```
#rect(width: 25%)
```



However, they can also describe any other property that is relative to some base, e.g. an amount of horizontal scaling[3.10.24/2.2] or the height of parentheses[3.8.17/2.1.1] relative to the height of the content they enclose.

/1 ## Scripting

Within your own code, you can use ratios as you like. You can multiply them with various other types as shown below:

Multiply by	Example	Result
`ratio`[3.10.20]	`27% * 10%`	`2.7%`
`length`[3.10.13]	`27% * 100pt`	`27pt`
`relative`[3.10.21]	`27% * (10% + 100pt)`	`2.7% + 27pt`
`angle`[3.10.3]	`27% * 100deg`	`27deg`
`int`[3.5.16]	`27% * 2`	`54%`
`float`[3.5.14]	`27% * 0.37037`	`10%`
`fraction`[3.10.9]	`27% * 3fr`	`0.81fr`

When ratios are displayed[3.5.24] in the document, they are rounded to two significant digits for readability.

3.10.21 `relative`

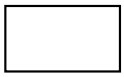
A length in relation to some known length.

This type is a combination of a length[3.10.13] with a ratio[3.10.20]. It results from addition and subtraction of a length and a ratio. Wherever a relative length is expected, you can also use a bare length or ratio.

/1 Relative to the page

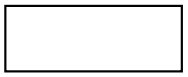
A common use case is setting the width or height of a layout element (e.g., block[3.10.4], rect[3.11.10], etc.) as a certain percentage of the width of the page. Here, the rectangle's width is set to `25%`, so it takes up one fourth of the page's *inner* width (the width minus margins).

```
#rect(width: 25%)
```



Bare lengths or ratios are always valid where relative lengths are expected, but the two can also be freely mixed:

```
#rect(width: 25% + 1cm)
```



For contents in the page `background`[3.10.17/4.17] and `foreground`[3.10.17/4.18], relative lengths are resolved against the page size including `bleed`[3.10.17/4.6]. This choice is made for convenience, as creating a proper bleed-aware background inherently requires extending it into the bleed area.

If you're trying to size an element so that it takes up the page's *full* width, you have a few options (this highly depends on your exact use case):

1. Set page margins to `0pt` (`#set page(margin: 0pt)`)
2. Multiply the ratio by the known full page width (`21cm * 69%`)
3. Use padding which will negate the margins (`#pad(x: -2.5cm, ..)`)
4. Use the page `background`[3.10.17/4.17] or `foreground`[3.10.17/4.18] field as those don't take margins into account (note that it will render the content outside of the document flow, see `place`[3.10.19] to control the content position)

/2 Relative to a container

When a layout element (e.g. a rect[3.11.10]) is nested in another layout container (e.g. a block[3.10.4]) instead of being a direct descendant of the page, relative widths become relative to the container:

```
#block(
  width: 100pt,
  fill: aqua,
  rect(width: 50%),
)
```



/3 Scripting

You can multiply relative lengths by ratios[3.10.20], integers[3.5.16], and floats[3.5.14].

A relative length has the following fields:
– `length`: Its length[3.10.13] component.
– `ratio`: Its ratio[3.10.20] component.

```
#(100% - 50pt).length \
#(100% - 50pt).ratio
```

```
-50pt
100%
```

3.10.22 `repeat` Element

Repeats content to the available space.

This can be useful when implementing a custom index, reference, or outline.

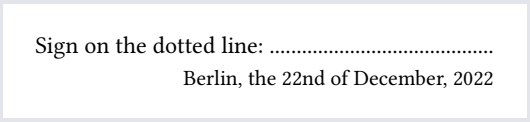
Space may be inserted between the instances of the body parameter, so be sure to adjust the `justify`[3.10.22/3.3] parameter accordingly.

Errors if there are no bounds on the available space, as it would create infinite content.

/1 Example

```
Sign on the dotted line:
#box(width: 1fr, repeat[.])

#set text(10pt)
#v(8pt, weak: true)
#align(right)[
  Berlin, the 22nd of December, 2022
]
```



/2 Accessibility

Repeated content is automatically marked as an artifact[3.14.4] and hidden from Assistive Technology (AT). Do not use this function to create content that contributes to the meaning of your document.

/3 Parameters

```
repeat(
  content,
  gap: length,
  justify: bool,
) → content
```

/3.1 **body** `content` *Required* *Positional*

The content to repeat.

/3.2 **gap** `length` *Settable* Default: `0pt`

The gap between each instance of the body.

/3.3 **justify** `bool` *Settable* Default: `true`

Whether to increase the gap between instances to completely fill the available space.

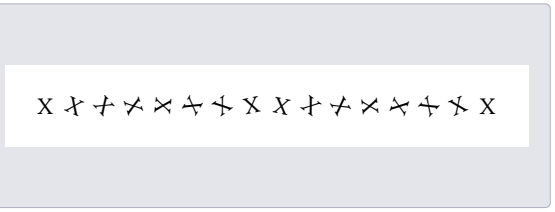
3.10.23 `rotate` Element

Rotates content without affecting layout.

Rotates an element by a given angle. The layout will act as if the element was not rotated unless you specify `reflow: true`.

/1 Example

```
#stack(
  dir: ltr,
  spacing: 1fr,
  ..range(16)
    .map(i => rotate(24deg * i)[X]),
)
```



/2 Parameters

```
rotate(
  angle,
  origin: alignment,
  reflow: bool,
  content,
) → content
```

/2.1 `angle` `angle` *Positional* *Settable* Default: `0deg`

The amount of rotation.

```
#rotate(-1.571rad)[Space!]
```



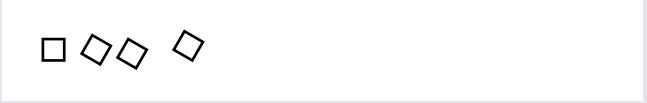
/2.2 `origin` `alignment` *Settable* Default: `center + horizon`

The origin of the rotation.

If, for instance, you wanted the bottom left corner of the rotated element to stay aligned with the baseline, you would set it to `bottom + left` instead.

```
#set text(spacing: 8pt)
#let square = square.with(width: 8pt)

#box(square())
#box(rotate(30deg, origin: center, square()))
#box(rotate(30deg, origin: top + left, square()))
#box(rotate(30deg, origin: bottom + right, square()))
```

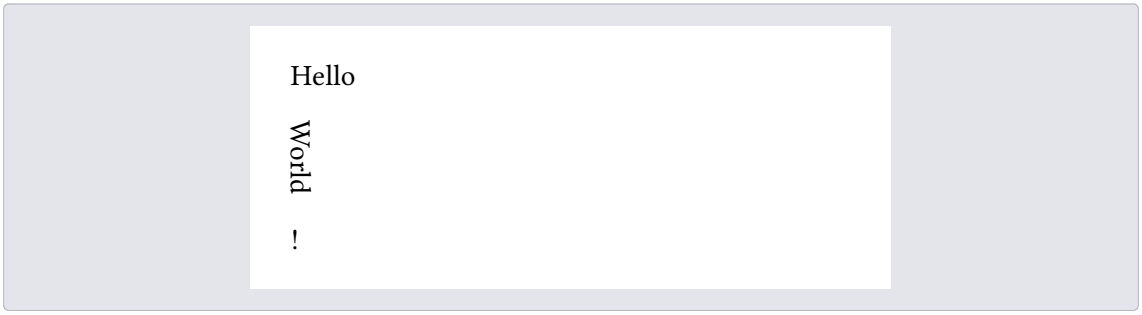


/2.3 `reflow` `bool` *Settable* Default: `false`

Whether the rotation impacts the layout.

If set to `false`, the rotated content will retain the bounding box of the original content. If set to `true`, the bounding box will take the rotation of the content into account and adjust the layout accordingly.

```
Hello #rotate(90deg, reflow: true)[World]!
```



/2.4 **body** `content` *Required* *Positional*

The content to rotate.

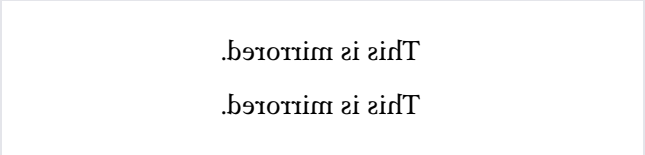
3.10.24 `scale` Element

Scales content without affecting layout.

Lets you mirror content by specifying a negative scale on a single axis.

/1 Example

```
#set align(center)
#scale(x: -100%)[This is mirrored.]
#scale(x: -100%, reflow: true)[This is mirrored.]
```



/2 Parameters

```
scale(
  factor: auto length ratio,
  x: auto length ratio,
  y: auto length ratio,
  origin: alignment,
  reflow: bool,
  content,
) → content
```

/2.1 `factor` `auto` or `length` or `ratio` Default: `100%`

The scaling factor for both axes, as a positional argument. This is just an optional shorthand notation for setting x and y to the same value.

/2.2 `x` `auto` or `length` or `ratio` *Settable* Default: `100%`

The horizontal scaling factor.

The body will be mirrored horizontally if the parameter is negative.

/2.3 `y` `auto` or `length` or `ratio` *Settable* Default: `100%`

The vertical scaling factor.

The body will be mirrored vertically if the parameter is negative.

/2.4 `origin` `alignment` *Settable* Default: `center + horizon`

The origin of the transformation.

```
A#box(scale(75%)[A])A \
B#box(scale(75%, origin: bottom + left)[B])B
```



/2.5 `reflow` `bool` *Settable* Default: `false`

Whether the scaling impacts the layout.

If set to `false`, the scaled content will be allowed to overlap other content. If set to `true`, it will compute the new size of the scaled content and adjust the layout accordingly.

```
Hello #scale(x: 20%, y: 40%, reflow: true)[World]!
```



/2.6 **body** `content` *Required* *Positional*

The content to scale.

3.10.25 `skew` Element

Skews content.

Skews an element in horizontal and/or vertical direction. The layout will act as if the element was not skewed unless you specify `reflow: true`.

/1 Example

```
#skew(ax: -12deg)[
  This is some fake italic text.
]
```

This is some fake italic text.

/2 Parameters

```
skew(
  ax: angle,
  ay: angle,
  origin: alignment,
  reflow: bool,
  content,
) → content
```

/2.1 `ax` `angle` *Settable*

Default: `0deg`

The horizontal skewing angle.

```
#skew(ax: 30deg)[Skewed]
```

Skewed

/2.2 `ay` `angle` *Settable*

Default: `0deg`

The vertical skewing angle.

```
#skew(ay: 30deg)[Skewed]
```

Skewed

/2.3 `origin` `alignment` *Settable*

Default: `center + horizon`

The origin of the skew transformation.

The origin will stay fixed during the operation.

```
X #box(skew(ax: -30deg, origin: center + horizon)[X]) X \
X #box(skew(ax: -30deg, origin: bottom + left)[X]) X \
X #box(skew(ax: -30deg, origin: top + right)[X]) X
```

X X X
X X X
X X X

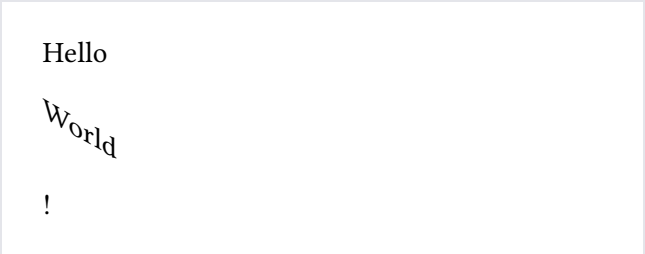
/2.4 `reflow` `bool` *Settable*

Default: `false`

Whether the skew transformation impacts the layout.

If set to `false`, the skewed content will retain the bounding box of the original content. If set to `true`, the bounding box will take the transformation of the content into account and adjust the layout accordingly.

```
Hello #skew(ay: 30deg, reflow: true, "World")!
```



/2.5 **body** `content` *Required* *Positional*

The content to skew.

3.10.26 h Element

Inserts horizontal spacing into a paragraph.

The spacing can be absolute, relative, or fractional. In the last case, the remaining space on the line is distributed among all fractional spacings according to their relative fractions.

/1 Example

```
First #h(1cm) Second \
First #h(30%) Second
```

First Second
First Second

/2 Fractional spacing

With fractional spacing, you can align things within a line without forcing a paragraph break (like `align`[3.10.1] would). Each fractionally sized element gets space based on the ratio of its fraction to the sum of all fractions.

```
First #h(1fr) Second \
First #h(1fr) Second #h(1fr) Third \
First #h(2fr) Second #h(1fr) Third
```

First Second
First Second Third
First Second Third

/3 Mathematical Spacing

In mathematical formulas[3.8], you can additionally use these constants to add spacing between elements: `thin` (1/6 em), `med` (2/9 em), `thick` (5/18 em), `quad` (1 em), `wide` (2 em).

/4 Parameters

```
h(
  relative fraction,
  weak: bool,
) → content
```

/4.1 amount `relative` or `fraction` *Required* *Positional*

How much spacing to insert.

/4.2 weak `bool` *Settable* Default: `false`

If `true`, the spacing collapses at the start or end of a paragraph. Moreover, from multiple adjacent weak spacings all but the largest one collapse.

Weak spacing in markup also causes all adjacent markup spaces to be removed, regardless of the amount of spacing inserted. To force a space next to weak spacing, you can explicitly write `#" "` (for a normal space) or `~` (for a non-breaking space). The latter can be useful to create a construct that always attaches to the preceding word with one non-breaking space, independently of whether a markup space existed in front or not.

```
#h(1cm, weak: true)
We identified a group of _weak_
specimens that fail to manifest
in most cases. However, when
#h(8pt, weak: true) supported
#h(8pt, weak: true) on both sides,
they do show up.

Further #h(0pt, weak: true) more,
even the smallest of them swallow
adjacent markup spaces.
```

We identified a group of *weak* specimens that fail to manifest in most cases. However, when supported on both sides, they do show up.

Furthermore, even the smallest of them swallow adjacent markup spaces.

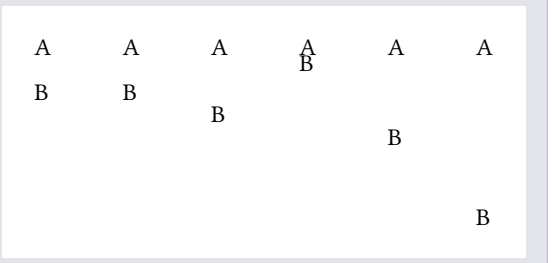
3.10.27 V Element

Inserts vertical spacing into a flow of blocks.

The spacing can be absolute, relative, or fractional. In the last case, the remaining space on the page is distributed among all fractional spacings according to their relative fractions.

/1 Example

```
#grid(
  rows: 3cm,
  columns: 6,
  gutter: 1fr,
  [A #parbreak() B],
  [A #v(0pt) B],
  [A #v(10pt) B],
  [A #v(0pt, weak: true) B],
  [A #v(40%, weak: true) B],
  [A #v(1fr) B],
)
```



/2 Parameters

```
v(
  relative fraction,
  weak: bool,
) → content
```

/2.1 **amount** `relative` or `fraction` *Required* *Positional*

How much spacing to insert.

/2.2 **weak** `bool` *Settable* Default: `false`

If `true`, the spacing collapses at the start or end of a flow. Moreover, from multiple adjacent weak spacings all but the largest one collapse. Weak spacings will always collapse adjacent paragraph spacing, even if the paragraph spacing is larger.

```
The following theorem is
foundational to the field:
#v(4pt, weak: true)
$ x^2 + y^2 = r^2 $
#v(4pt, weak: true)
The proof is simple:
```

The following theorem is foundational to the field:

$$x^2 + y^2 = r^2$$

The proof is simple:

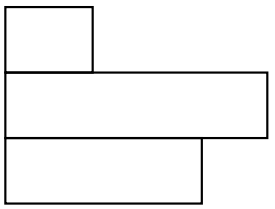
3.10.28 `stack` Element

Arranges content and spacing horizontally or vertically.

The stack places a list of items along an axis, with optional spacing between each item.

/1 Example

```
#stack(
  dir: ttb,
  rect(width: 40pt),
  rect(width: 120pt),
  rect(width: 90pt),
)
```



/2 Accessibility

Stacks do not carry any special semantics. The contents of the stack are read by Assistive Technology (AT) in the order in which they have been passed to this function.

/3 Parameters

```
stack(
  dir: direction,
  spacing: none relative fraction,
  ..relative fraction content,
) → content
```

/3.1 `dir` `direction` *Settable* Default: `ttb`

The direction along which the items are stacked. Possible values are:

- `ltr`: Left to right.
- `rtl`: Right to left.
- `ttb`: Top to bottom.
- `btt`: Bottom to top.

You can use the `start` and `end` methods to obtain the initial and final points (respectively) of a direction, as `alignment`. You can also use the `axis` method to determine whether a direction is `"horizontal"` or `"vertical"`. The `inv` method returns a direction's inverse direction.

For example, `ttb.start()` is `top`, `ttb.end()` is `bottom`, `ttb.axis()` is `"vertical"` and `ttb.inv()` is equal to `btt`.

/3.2 `spacing` `none` or `relative` or `fraction` *Settable* Default: `none`

Spacing to insert between items where no explicit spacing was provided.

/3.3 `children` `relative` or `fraction` or `content` *Required* *Positional* *Variadic*

The children to stack along the axis.

3.11 Visualize

Drawing and data visualization.

If you want to create more advanced drawings or plots, also have a look at the CeTZ[A] package as well as more specialized packages[B] for your use case.

3.11.1 Accessibility

All shapes and paths drawn by Typst are automatically marked as artifacts[3.14.4] to make them invisible to Assistive Technology (AT) during PDF export. However, their contents (if any) remain accessible.

If you are using the functions in this category to create an illustration with semantic meaning, make it accessible by wrapping it in a `figure`[3.6.8] function call. Use its `alt` parameter[3.6.8/5.2] to provide an alternative description[4.4.7].

Definitions

3.11.2 Circle A circle with optional content. . . . 401
3.11.3 Color A color in a specific color space. . . . 403
3.11.4 Curve A curve consisting of movements, lines, and Bézier segments. . . . 417
3.11.5 Ellipse An ellipse with optional content. . . . 424
3.11.6 Gradient A color gradient. . . . 425
3.11.7 Image A raster or vector graphic. . . . 434
3.11.8 Line A line from one point to another. . . . 438
3.11.9 Polygon A closed polygon. . . . 440
3.11.10 Rectangle A rectangle with optional content. . . . 442
3.11.11 Square A square with optional content. . . . 444
3.11.12 Stroke Defines how to draw a line. . . . 446
3.11.13 Tiling A repeating tiling fill. . . . 450

[A] https://github.com/johannes-wolf/cetz
[B] https://typst.app/universe

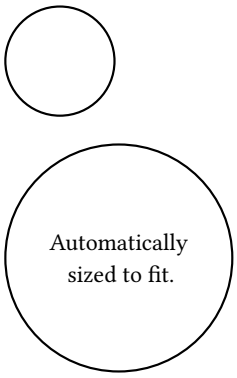
3.11.2 `circle` Element

A circle with optional content.

/1 Example

```
// Without content.
#circle(radius: 25pt)

// With content.
#circle[
  #set align(center + horizon)
  Automatically \
  sized to fit.
]
```



/2 Parameters

```
circle(
  radius: length,
  width: auto relative,
  height: auto relative fraction,
  fill: none color gradient tiling,
  stroke: none auto length color gradient stroke tiling dictionary,
  inset: relative dictionary,
  outset: relative dictionary,
  none content,
) → content
```

/2.1 `radius` `length`

Default: `0pt`

The circle's radius. This is mutually exclusive with `width` and `height`.

/2.2 `width` `auto` or `relative` *Settable*

Default: `auto`

The circle's width. This is mutually exclusive with `radius` and `height`.

In contrast to `radius`, this can be relative to the parent container's width.

/2.3 `height` `auto` or `relative` or `fraction` *Settable*

Default: `auto`

The circle's height. This is mutually exclusive with `radius` and `width`.

In contrast to `radius`, this can be relative to the parent container's height.

/2.4 `fill` `none` or `color` or `gradient` or `tiling` *Settable*

Default: `none`

How to fill the circle. See the rectangle's documentation[3.11.10/2.3] for more details.

/2.5 `stroke` `none` or `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable*

Default: `auto`

How to stroke the circle. See the rectangle's documentation[3.11.10/2.4] for more details.

/2.6 `inset` `relative` or `dictionary` *Settable*

Default: `0% + 5pt`

How much to pad the circle's content. See the box's documentation[3.10.5/2.7] for more details.

/2.7 **outset** `relative` or `dictionary` *Settable* Default: `(:)`

How much to expand the circle's size without affecting the layout. See the box's documentation[3.10.5/2.8] for more details.

/2.8 **body** `none` or `content` *Positional* *Settable* Default: `none`

The content to place into the circle. The circle expands to fit this content, keeping the 1-1 aspect ratio.

3.11.3 color

A color in a specific color space.

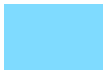
Typst supports:
- sRGB through the `rgb` function[3.11.3/4.5]
- Device CMYK through the `cmyk` function[3.11.3/4.6]
- D65 Gray through the `luma` function[3.11.3/4.1]
- Oklab through the `oklab` function[3.11.3/4.2]
- Oklch through the `oklch` function[3.11.3/4.3]
- Linear RGB through the `color.linear-rgb` function[3.11.3/4.4]
- HSL through the `color.hsl` function[3.11.3/4.7]
- HSV through the `color.hsv` function[3.11.3/4.8]
- Color spaces described by spot colorants through the `color.spot` type[3.11.3/4.21]

All color spaces except for CMYK and spot colorants have alpha channels.

Throughout the documentation, we use the term *process color* for colors that can be blended with each other (currently all colors other than spot colors). In this term, *process* signifies that the shade is being created throughout the printing process instead of ahead of time.

/1 Example

```
#rect(fill: aqua)
```

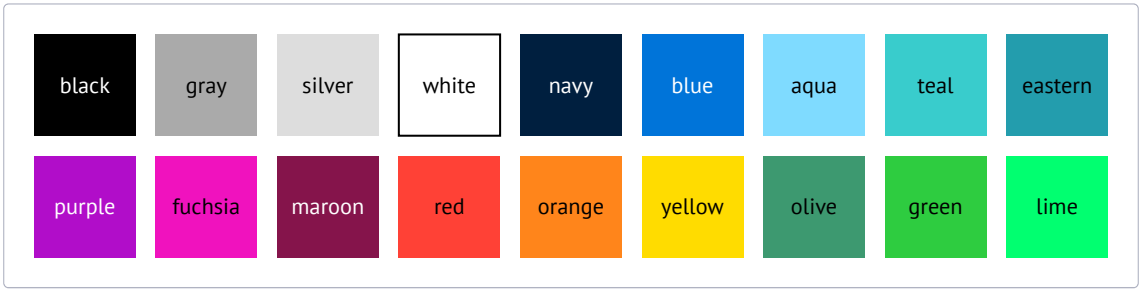


/2 Predefined colors

Typst defines the following built-in colors:

Color	Definition
black	`luma(0)`
gray	`luma(170)`
silver	`luma(221)`
white	`luma(255)`
navy	`rgb("#001f3f")`
blue	`rgb("#0074d9")`
aqua	`rgb("#7fdbff")`
teal	`rgb("#39cccc")`
eastern	`rgb("#239dad")`
purple	`rgb("#b10dc9")`
fuchsia	`rgb("#f012be")`
maroon	`rgb("#85144b")`
red	`rgb("#ff4136")`
orange	`rgb("#ff851b")`
yellow	`rgb("#ffdc00")`
olive	`rgb("#3d9970")`
green	`rgb("#2ecc40")`
lime	`rgb("#01ff70")`

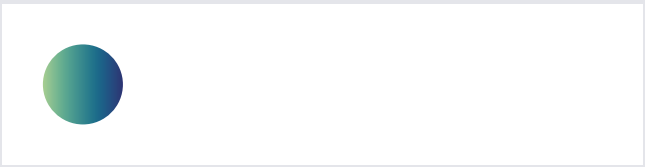
The predefined colors and the most important color constructors are available globally and also in the color type's scope, so you can write either `color.red` or just `red`.



/3 Predefined color maps

Typst also includes a number of preset color maps that can be used for gradients[3.11.6/2]. These are simply arrays of colors defined in the module `color.map`.

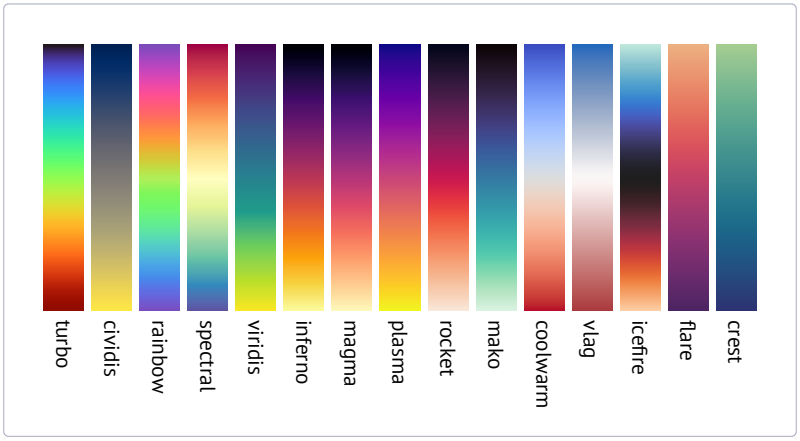
```
#circle(fill: gradient.linear(..color.map.crest))
```



Map	Details
`turbo`	A perceptually uniform rainbow-like color map. Read this blog post[A] for more details.
`cividis`	A blue to gray to yellow color map. See this blog post[B] for more details.
`rainbow`	Cycles through the full color spectrum. This color map is best used by setting the interpolation color space to HSL[3.11.3/4.7]. The rainbow gradient is **not suitable** for data visualization because it is not perceptually uniform, so the differences between values become unclear to your readers. It should only be used for decorative purposes.
`spectral`	Red to yellow to blue color map.
`viridis`	A purple to teal to yellow color map.
`inferno`	A black to red to yellow color map.
`magma`	A black to purple to yellow color map.
`plasma`	A purple to pink to yellow color map.
`rocket`	A black to red to white color map.
`mako`	A black to teal to white color map.
`coolwarm`	A blue to white to red color map with smooth transitions.
`vlag`	A light blue to white to red color map.
`icefire`	A light teal to black to orange color map.
`flare`	A orange to purple color map that is perceptually uniform.
`crest`	A light green to blue color map.

Some popular presets are not included because they are not available under a free licence. Others, like Jet[C], are not included because they are not color blind friendly. Feel free to use or create a package with other presets that are useful to you!

[A] https://ai.googleblog.com/2019/08/turbo-improved-rainbow-colormap-for.html
[B] https://bids.github.io/colormap/
[C] https://jakevdp.github.io/blog/2014/10/16/how-bad-is-your-colormap/



/4 Definitions

/4.1 luma

Create a grayscale color.

A grayscale color is represented internally by a single `lightness` component.

These components are also available using the `components`[3.11.3/4.9] method.

```
#for x in range(250, step: 50) {
  box(square(fill: luma(x)))
}
```



```
color.luma(
  int ratio,
  ratio,
  color,
) → color
```

/4.1.1 lightness int or ratio *Required* *Positional*

The lightness component.

/4.1.2 alpha ratio *Required* *Positional*

The alpha component.

/4.1.3 color color *Required* *Positional*

Alternatively: The color to convert to grayscale.

If this is given, the `lightness` should not be given.

/4.2 oklab

Create an Oklab[A] color.

This color space is well suited for the following use cases:
- Color manipulation such as saturating while keeping perceived hue
- Creating grayscale images with uniform perceived lightness
- Creating smooth and uniform color transition and gradients

A linear Oklab color is represented internally by an array of four components:
- lightness (`ratio`[3.10.20])
- a (`float`[3.5.14] or `ratio`[3.10.20]. Ratios are relative to `0.4`; meaning `50%` is equal to `0.2`)
- b (`float`[3.5.14] or `ratio`[3.10.20]. Ratios are relative to `0.4`; meaning `50%` is equal to `0.2`)

[A] https://bottosson.github.io/posts/oklab/

– alpha (ratio[3.10.20])

These components are also available using the components[3.11.3/4.9] method.

```
#square(
  fill: oklab(27%, 20%, -3%, 50%)
)
```



```
color.oklab(
  ratio,
  float ratio,
  float ratio,
  ratio,
  color,
) → color
```

/4.2.1 **lightness** ratio *Required* *Positional*

The lightness component.

/4.2.2 **a** float or ratio *Required* *Positional*

The a ("green/red") component.

/4.2.3 **b** float or ratio *Required* *Positional*

The b ("blue/yellow") component.

/4.2.4 **alpha** ratio *Required* *Positional*

The alpha component.

/4.2.5 **color** color *Required* *Positional*

Alternatively: The color to convert to Oklab.

If this is given, the individual components should not be given.

/4.3 oklch

Create an Oklch[A] color.

This color space is well suited for the following use cases:
– Color manipulation involving lightness, chroma, and hue
– Creating grayscale images with uniform perceived lightness
– Creating smooth and uniform color transition and gradients

A linear Oklch color is represented internally by an array of four components:
– lightness (ratio[3.10.20])
– chroma (float[3.5.14] or ratio[3.10.20]. Ratios are relative to 0.4; meaning 50% is equal to 0.2)
– hue (angle[3.10.3])
– alpha (ratio[3.10.20])

These components are also available using the components[3.11.3/4.9] method.

```
#square(
  fill: oklch(40%, 0.2, 160deg, 50%)
)
```



[A] https://bottosson.github.io/posts/oklab/

```
color.oklch(
  ratio,
  float ratio,
  angle,
  ratio,
  color,
) → color
```

/4.3.1 **lightness** ratio *Required* *Positional*

The lightness component.

/4.3.2 **chroma** float or ratio *Required* *Positional*

The chroma component.

/4.3.3 **hue** angle *Required* *Positional*

The hue component.

/4.3.4 **alpha** ratio *Required* *Positional*

The alpha component.

/4.3.5 **color** color *Required* *Positional*

Alternatively: The color to convert to Oklch.

If this is given, the individual components should not be given.

/4.4 linear-rgb

Create an RGB(A) color with linear luma.

This color space is similar to sRGB, but with the distinction that the color component are not gamma corrected. This makes it easier to perform color operations such as blending and interpolation. Although, you should prefer to use the oklab function[3.11.3/4.2] for these.

A linear RGB(A) color is represented internally by an array of four components:
- red (ratio[3.10.20])
- green (ratio[3.10.20])
- blue (ratio[3.10.20])
- alpha (ratio[3.10.20])

These components are also available using the components[3.11.3/4.9] method.

```
#square(fill: color.linear-rgb(
  30%, 50%, 10%,
))
```



```
color.linear-rgb(
  int ratio,
  int ratio,
  int ratio,
  int ratio,
  color,
) → color
```

/4.4.1 **red** int or ratio *Required* *Positional*

The red component.

/4.4.2 **green** int or ratio *Required* *Positional*

The green component.

/4.4.3 **blue** int or ratio *Required* *Positional*

The blue component.

/4.4.4 **alpha** int or ratio *Required* *Positional*

The alpha component.

/4.4.5 **color** color *Required* *Positional*

Alternatively: The color to convert to linear RGB(A).

If this is given, the individual components should not be given.

/4.5 rgb

Create an RGB(A) color.

The color is specified in the sRGB color space.

An RGB(A) color is represented internally by an array of four components:
- red (ratio[3.10.20])
- green (ratio[3.10.20])
- blue (ratio[3.10.20])
- alpha (ratio[3.10.20])

These components are also available using the components[3.11.3/4.9] method.

```
#square(fill: rgb("#b1f2eb"))
#square(fill: rgb(87, 127, 230))
#square(fill: rgb(25%, 13%, 65%))
```



```
color.rgb(
  int ratio,
  int ratio,
  int ratio,
  int ratio,
  str,
  color,
) → color
```

/4.5.1 **red** int or ratio *Required* *Positional*

The red component.

/4.5.2 **green** int or ratio *Required* *Positional*

The green component.

/4.5.3 **blue** int or ratio *Required* *Positional*

The blue component.

/4.5.4 **alpha** int or ratio *Required* *Positional*

The alpha component.

/4.5.5 **hex** str *Required* *Positional*

Alternatively: The color in hexadecimal notation.

Accepts three, four, six or eight hexadecimal digits and optionally a leading hash.

If this is given, the individual components should not be given.

```
#text(16pt, rgb("#239dad"))[
  *Typst*
]
```

Typst

/4.5.6 **color** color *Required* *Positional*

Alternatively: The color to convert to RGB(a).

If this is given, the individual components should not be given.

/4.6 cmyk

Create a CMYK color.

This is useful if you want to target a specific printer. The conversion to RGB for display preview might differ from how your printer reproduces the color.

A CMYK color is represented internally by an array of four components:
- cyan (ratio[3.10.20])
- magenta (ratio[3.10.20])
- yellow (ratio[3.10.20])
- key (ratio[3.10.20])

These components are also available using the components[3.11.3/4.9] method.

Note that CMYK colors are not currently supported when PDF/A output is enabled.

```
#square(
  fill: cmyk(27%, 0%, 3%, 5%)
)
```



```
color.cmyk(
  ratio,
  ratio,
  ratio,
  ratio,
  color,
) → color
```

/4.6.1 **cyan** ratio *Required* *Positional*

The cyan component.

/4.6.2 **magenta** ratio *Required* *Positional*

The magenta component.

/4.6.3 **yellow** ratio *Required* *Positional*

The yellow component.

/4.6.4 **key** ratio *Required* *Positional*

The key component.

/4.6.5 **color** color *Required* *Positional*

Alternatively: The color to convert to CMYK.

If this is given, the individual components should not be given.

/4.7 hsl

Create an HSL color.

This color space is useful for specifying colors by hue, saturation and lightness. It is also useful for color manipulation, such as saturating while keeping perceived hue.

An HSL color is represented internally by an array of four components:
– hue (angle[3.10.3])
– saturation (ratio[3.10.20])
– lightness (ratio[3.10.20])
– alpha (ratio[3.10.20])

These components are also available using the components[3.11.3/4.9] method.

```
#square(
  fill: color.hsl(30deg, 50%, 60%)
)
```



```
color.hsl(
  angle,
  int ratio,
  int ratio,
  int ratio,
  color,
) → color
```

/4.7.1 **hue** angle *Required* *Positional*

The hue angle.

/4.7.2 **saturation** int or ratio *Required* *Positional*

The saturation component.

/4.7.3 **lightness** int or ratio *Required* *Positional*

The lightness component.

/4.7.4 **alpha** int or ratio *Required* *Positional*

The alpha component.

/4.7.5 **color** color *Required* *Positional*

Alternatively: The color to convert to HSL.

If this is given, the individual components should not be given.

/4.8 hsv

Create an HSV color.

This color space is useful for specifying colors by hue, saturation and value. It is also useful for color manipulation, such as saturating while keeping perceived hue.

An HSV color is represented internally by an array of four components:
– hue (angle[3.10.3])
– saturation (ratio[3.10.20])
– value (ratio[3.10.20])
– alpha (ratio[3.10.20])

These components are also available using the components[3.11.3/4.9] method.

```
#square(
  fill: color.hsv(30deg, 50%, 60%)
)
```



```
color.hsv(
  angle,
  int ratio,
  int ratio,
  int ratio,
  color,
) → color
```

/4.8.1 **hue** angle *Required* *Positional*

The hue angle.

/4.8.2 **saturation** int or ratio *Required* *Positional*

The saturation component.

/4.8.3 **value** int or ratio *Required* *Positional*

The value component.

/4.8.4 **alpha** int or ratio *Required* *Positional*

The alpha component.

/4.8.5 **color** color *Required* *Positional*

Alternatively: The color to convert to HSL.

If this is given, the individual components should not be given.

/4.9 components

Extracts the components of this color.

The size and values of this array depends on the color space. You can obtain the color space using space[3.11.3/4.10]. Below is a table of the color spaces and their components:

Color space	C1	C2	C3	C4
luma[3.11.3/4.1]	Lightness			
oklab[3.11.3/4.2]	Lightness	a	b	Alpha
oklch[3.11.3/4.3]	Lightness	Chroma	Hue	Alpha
linear-rgb[3.11.3/4.4]	Red	Green	Blue	Alpha
rgb[3.11.3/4.5]	Red	Green	Blue	Alpha
cmyk[3.11.3/4.6]	Cyan	Magenta	Yellow	Key
hsl[3.11.3/4.7]	Hue	Saturation	Lightness	Alpha
hsv[3.11.3/4.8]	Hue	Saturation	Value	Alpha
spot[3.11.3/4.21]	Tint			

For the meaning and type of each individual value, see the documentation of the corresponding color space. The alpha component is optional and only included if the `alpha` argument is `true`. The length of the returned array depends on the number of components and whether the alpha component is included.

```
// note that the alpha component is included by default
#rgb(40%, 60%, 80%).components()
```

```
(40%, 60%, 80%, 100%)
```

```
self.components(alpha: bool) → array
```

/4.9.1 **alpha** `bool` Default: `true`

Whether to include the alpha component.

/4.10 space

Returns the constructor function for this color's space.

Returns one of:
- luma[3.11.3/4.1]
- oklab[3.11.3/4.2]
- oklch[3.11.3/4.3]
- linear-rgb[3.11.3/4.4]
- rgb[3.11.3/4.5]
- cmyk[3.11.3/4.6]
- hsl[3.11.3/4.7]
- hsv[3.11.3/4.8]

```
#let color = cmyk(1%, 2%, 3%, 4%)
#(color.space() == cmyk)
```

```
true
```

```
self.space() → spot any
```

/4.11 to-hex

Returns the color's RGB(A) hex representation (such as `#ffaa32` or `#020304fe`). The alpha component (last two digits in `#020304fe`) is omitted if it is equal to `ff` (255 / 100%).

```
self.to-hex() → str
```

/4.12 lighten

Lightens a color by a given factor.

```
self.lighten(ratio) → color
```

/4.12.1 **factor** `ratio` *Required* *Positional*

The factor to lighten the color by.

/4.13 darken

Darkens a color by a given factor.

```
self.darken(ratio) → color
```

/4.13.1 **factor** `ratio` *Required* *Positional*

The factor to darken the color by.

/4.14 saturate

Increases the saturation of a color by a given factor.

Only process colors can be saturated. If you want to saturate a spot color, convert it into a process color first.

```
self.saturate(ratio) → color
```

/4.14.1 **factor** `ratio` *Required* *Positional*

The factor to saturate the color by.

/4.15 desaturate

Decreases the saturation of a color by a given factor.

Only process colors can be desaturated. If you want to desaturate a spot color, convert it into a process color first.

```
self.desaturate( ratio ) → color
```

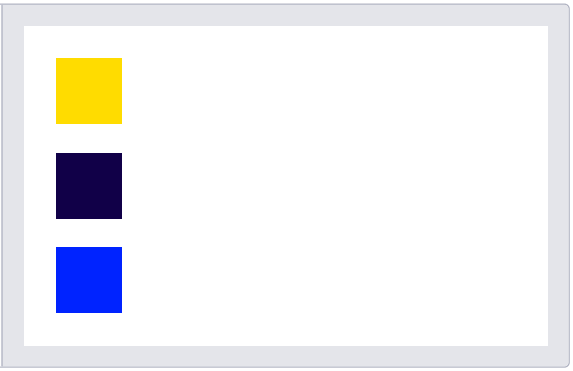
/4.15.1 factor ratio *Required* *Positional*

The factor to desaturate the color by.

/4.16 negate

Produces the complementary color using a provided color space. You can think of it as the opposite side on a color wheel.

```
#square(fill: yellow)
#square(fill: yellow.negate())
#square(fill: yellow.negate(space: rgb))
```



```
self.negate(space: auto spot any ) → color
```

/4.16.1 space auto or spot or any Default: auto

The color space used for the transformation. By default, a perceptual color space is used.

/4.17 rotate

Rotates the hue of the color by a given angle.

This function only works on color models with a well-defined hue component, i.e. Oklch, HSL, and HSV.

```
self.rotate(
  angle ,
  space: any ,
) → color
```

/4.17.1 angle angle *Required* *Positional*

The angle to rotate the hue by.

/4.17.2 space any Default: oklch

The color space used to rotate. By default, this happens in a perceptual color space (oklch[3.11.3/4.3]).

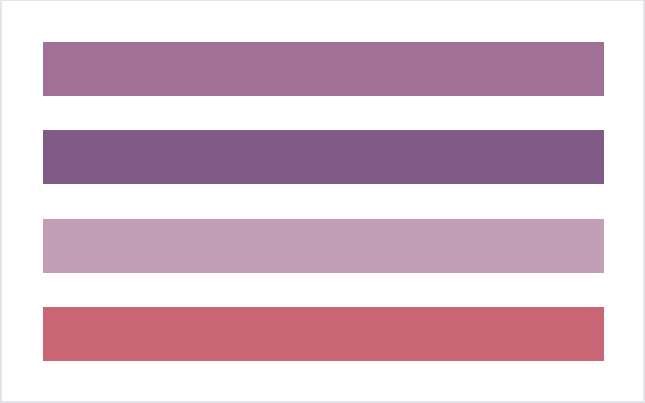
/4.18 mix

Create a color by mixing two or more colors.

In color spaces with a hue component (HSL, HSV, Oklch), only two colors can be mixed at once. Mixing more than two colors in such a space will result in an error!

```
#set block(height: 20pt, width: 100%)
#block(fill: red.mix(blue))
#block(fill: red.mix(blue, space: rgb))
```

```
#block(fill: color.mix(red, blue, white))
#block(fill: color.mix((red, 70%), (blue, 30%)))
```



```
color.mix(
  ..color array,
  space: auto spot any,
) → color
```

/4.18.1 **colors** color or array *Required* *Positional* *Variadic*

The colors, optionally with weights, specified as a pair (array of length two) of color and weight (float or ratio).

The weights do not need to add to 100%, they are relative to the sum of all weights.

/4.18.2 **space** auto or spot or any Default: auto

The color space to mix in. By default, this happens in a perceptual color space (oklab[3.11.3/4.2]) or, if all colors use the same spot colorant, using that colorant.

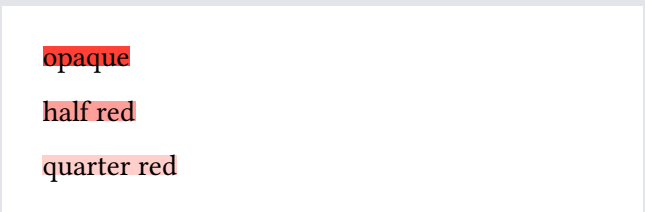
All colors will be converted into this color space.

/4.19 transparentize

Makes a color more transparent by a given factor.

This method is relative to the existing alpha value. If the scale is positive, calculates `alpha - alpha * scale`. Negative scales behave like `color.opacify(-scale)`.

```
#block(fill: red)[opaque]
#block(fill: red.transparentize(50%))[half red]
#block(fill: red.transparentize(75%))[quarter red]
```




```
self.transparentize(ratio) → color
```

/4.19.1 **scale** ratio *Required* *Positional*

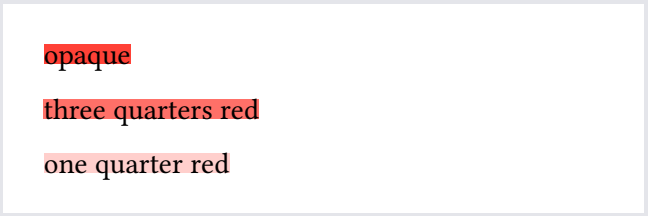
The factor to change the alpha value by.

/4.20 opacify

Makes a color more opaque by a given scale.

This method is relative to the existing alpha value. If the scale is positive, calculates `alpha + scale - alpha * scale`. Negative scales behave like `color.transparentize(-scale)`.

```
#let half-red = red.transparentize(50%)
#block(fill: half-red.opacify(100%))[opaque]
#block(fill: half-red.opacify(50%))[three quarters red]
#block(fill: half-red.opacify(-50%))[one quarter red]
```




```
self.opacify( ratio ) → color
```

/4.20.1 scale `ratio` *Required* *Positional*

The scale to change the alpha value by.

/4.21 spot

A spot colorant from which spot colors can be created.

Use spot colors to request a precise pigment in a professional print environment. Once you have created a spot colorant, you can create colors using its `tint` method[3.11.3/4.21.2.1].

/4.21.1 Constructor

Create a new spot colorant.

```
color.spot(
  none str,
  color,
) → spot
```

/4.21.1.1 name `none` or `str` *Required* *Positional*

Name of the spot colorant to use.

In production, this name will be manually checked and matched to a colorant, so this value needs to be unambiguous. It's best to reference a color from a registry like PANTONE, HKS, RAL, Toyo & DIC, etc.

Values in here may be treated case-sensitively during production: `"PANTONE 2221 C"` and `"PANTONE 2221 c"` may be treated as separate colors. Ensure that you are using a consistent naming convention, either referencing a registry or through coordination with your production printing experts.

If this value is `"all"` and your print will involve multiple color plates[A], use of this colorant will result in the specified tint being applied equally to all plates. If you choose `none`, no colorant will be applied when using this color. This special value is often used to indicate cuts or varnishes. Be sure to discuss this with your production printer!

We do not recommend using the names `"Cyan"`, `"Magenta"`, `"Yellow"`, `"Key"`, `"Black"`, or their translations to your local language. Depending on your printer, they may or may not be interpreted as CMYK process colors.

[A] https://en.wikipedia.org/wiki/Offset_printing#Plates

Variant	Details
`"all"`	Use all available color plates instead of only a specific colorant.

/4.21.1.2 **fallback** color *Required* *Positional*

How to render this color if the specified colorant is not available.

Many mediums, like on-screen preview and household printers, will not have this specific spot colorant available. To display an approximation of the intended print, another, available color is used instead.

/4.21.2 **Definitions on spot**

/4.21.2.1 **tint**

Create a spot color at a specific tint of this colorant.

The tint represents what percentage of the colorant is applied. A tint of `100%` means the colorant is applied at full strength, while `0%` means no colorant is applied.

```
#let pantone = color.spot(
  "PANTONE 2221 C",
  rgb("#239dad")
)

#square(fill: pantone.tint(100%))
#square(fill: pantone.tint(70%))
#square(fill: pantone.tint(40%))
```



```
self.tint( ratio ) → color
```

/4.21.2.1.1 **value** ratio *Required* *Positional*

The tint percentage, between `0%` and `100%`.

3.11.4 `curve` Element

A curve consisting of movements, lines, and Bézier segments.

At any point in time, there is a conceptual pen or cursor.
- Move elements move the cursor without drawing.
- Line/Quadratic/Cubic elements draw a segment from the cursor to a new position, potentially with control point for a Bézier curve.
- Close elements draw a straight or smooth line back to the start of the curve or the latest preceding move segment.

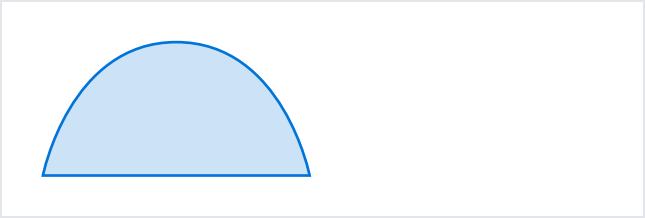
For layout purposes, the bounding box of the curve is a tight rectangle containing all segments as well as the point `(0pt, 0pt)`.

Positions may be specified absolutely (i.e. relatively to `(0pt, 0pt)`), or relative to the current pen/cursor position, that is, the position where the previous segment ended.

Bézier curve control points can be skipped by passing `none` or automatically mirrored from the preceding segment by passing `auto`.

/1 Example

```
#curve(
  fill: blue.lighten(80%),
  stroke: blue,
  curve.move((0pt, 50pt)),
  curve.line((100pt, 50pt)),
  curve.cubic(none, (90pt, 0pt), (50pt, 0pt)),
  curve.close(),
)
```



/2 Parameters

```
curve(
  fill: none color gradient tiling,
  fill-rule: str,
  stroke: none auto length color gradient stroke tiling dictionary,
  ..content,
) → content
```

/2.1 **`fill`** `none` or `color` or `gradient` or `tiling` *Settable* Default: `none`

How to fill the curve.

When setting a fill, the default stroke disappears. To create a curve with both fill and stroke, you have to configure both.

/2.2 **`fill-rule`** `str` *Settable* Default: `"non-zero"`

The drawing rule used to fill the curve.

```typ
// We use `.with` to get a new
// function that has the common
// arguments pre-applied.
#let star = curve.with(
  fill: red,
  curve.move((25pt, 0pt)),
  curve.line((10pt, 50pt)),
  curve.line((50pt, 20pt)),
  curve.line((0pt, 20pt)),
  curve.line((40pt, 50pt)),
  curve.close(),
)

#star(fill-rule: "non-zero")
#star(fill-rule: "even-odd")
```



Variant	Details
`"non-zero"`	Specifies that "inside" is computed by a non-zero sum of signed edge crossings.
`"even-odd"`	Specifies that "inside" is computed by an odd number of edge crossings.

/2.3 **stroke** `none` or `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable* Default: `auto`

How to stroke[3.11.12] the curve.

Can be set to `none` to disable the stroke or to `auto` for a stroke of `1pt` black if and only if no fill is given.

```typ
#let down = curve.line((40pt, 40pt), relative: true)
#let up = curve.line((40pt, -40pt), relative: true)

#curve(
  stroke: 4pt + gradient.linear(red, blue),
  down, up, down, up, down,
)
```



/2.4 **components** `content` *Required* *Positional* *Variadic*

The components of the curve, in the form of moves, line and Bézier segment, and closes.

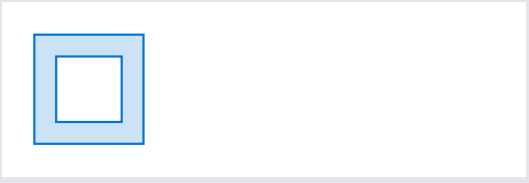
/3 Definitions

/3.1 move Element

Starts a new curve component.

If no `curve.move` element is passed, the curve will start at `(0pt, 0pt)`.

```typ
#curve(
  fill: blue.lighten(80%),
  fill-rule: "even-odd",
  stroke: blue,
  curve.line((50pt, 0pt)),
  curve.line((50pt, 50pt)),
  curve.line((0pt, 50pt)),
```



```
  curve.close(),
  curve.move((10pt, 10pt)),
  curve.line((40pt, 10pt)),
  curve.line((40pt, 40pt)),
  curve.line((10pt, 40pt)),
  curve.close(),
)
```

```
curve.move(
  array,
  relative: bool,
) → content
```

/3.1.1 **start** `array` *Required* *Positional*

The starting point for the new component.

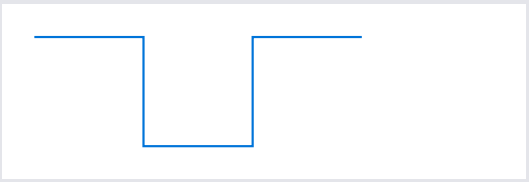
/3.1.2 **relative** `bool` *Settable* Default: `false`

Whether the coordinates are relative to the previous point.

/3.2 line Element

Adds a straight line from the current point to a following one.

```
#curve(
  stroke: blue,
  curve.line((50pt, 0pt)),
  curve.line((50pt, 50pt)),
  curve.line((100pt, 50pt)),
  curve.line((100pt, 0pt)),
  curve.line((150pt, 0pt)),
)
```



```
curve.line(
  array,
  relative: bool,
) → content
```

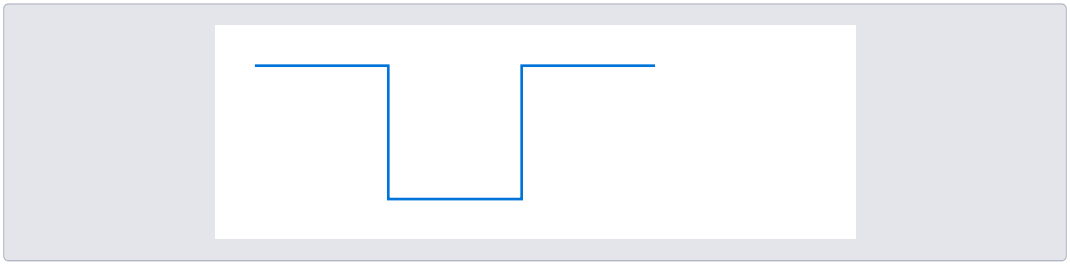
/3.2.1 **end** `array` *Required* *Positional*

The point at which the line shall end.

/3.2.2 **relative** `bool` *Settable* Default: `false`

Whether the coordinates are relative to the previous point.

```
#curve(
  stroke: blue,
  curve.line((50pt, 0pt), relative: true),
  curve.line((0pt, 50pt), relative: true),
  curve.line((50pt, 0pt), relative: true),
  curve.line((0pt, -50pt), relative: true),
  curve.line((50pt, 0pt), relative: true),
)
```



/3.3 quad Element

Adds a quadratic Bézier curve segment from the last point to `end`, using `control` as the control point.

```
// Function to illustrate where the control point is.
#let mark((x, y)) = place(
  dx: x - 1pt, dy: y - 1pt,
  circle(fill: aqua, radius: 2pt),
)

#mark((20pt, 20pt))

#curve(
  stroke: blue,
  curve.move((0pt, 100pt)),
  curve.quad((20pt, 20pt), (100pt, 0pt)),
)
```



```
curve.quad(
  none auto array,
  array,
  relative: bool,
) → content
```

/3.3.1 control `none` or `auto` or `array` *Required* *Positional*

The control point of the quadratic Bézier curve.

- If `auto` and this segment follows another quadratic Bézier curve, the previous control point will be mirrored.
- If `none`, the control point defaults to `end`, and the curve will be a straight line.

```
#curve(
  stroke: 2pt,
  curve.quad((20pt, 40pt), (40pt, 40pt), relative: true),
  curve.quad(auto, (40pt, -40pt), relative: true),
)
```



/3.3.2 **end** `array` *Required* *Positional*

The point at which the segment shall end.

/3.3.3 **relative** `bool` *Settable* Default: `false`

Whether the `control` and `end` coordinates are relative to the previous point.

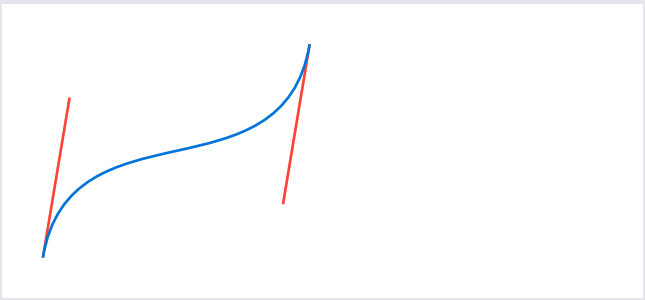
/3.4 cubic Element

Adds a cubic Bézier curve segment from the last point to `end`, using `control-start` and `control-end` as the control points.

```
// Function to illustrate where the control points are.
#let handle(start, end) = place(
  line(stroke: red, start: start, end: end)
)

#handle((0pt, 80pt), (10pt, 20pt))
#handle((90pt, 60pt), (100pt, 0pt))

#curve(
  stroke: blue,
  curve.move((0pt, 80pt)),
  curve.cubic((10pt, 20pt), (90pt, 60pt), (100pt, 0pt)),
)
```



```
curve.cubic(
  none auto array ,
  none array ,
  array ,
  relative: bool ,
) → content
```

/3.4.1 **control-start** `none` or `auto` or `array` *Required* *Positional*

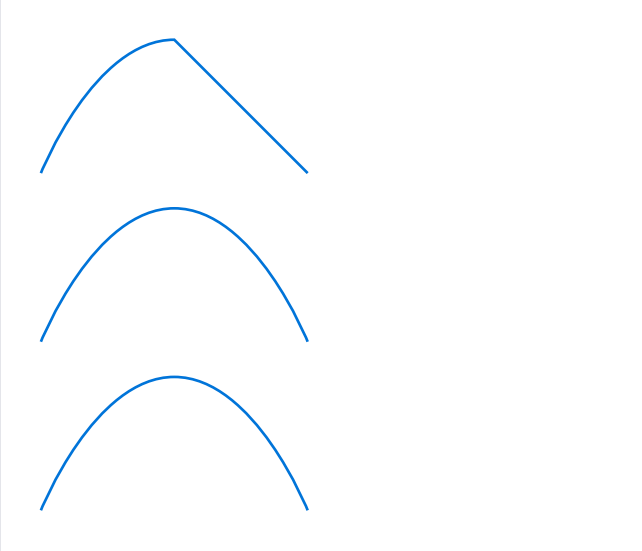
The control point going out from the start of the curve segment.

- If `auto` and this element follows another `curve.cubic` element, the last control point will be mirrored. In SVG terms, this makes `curve.cubic` behave like the `S` operator instead of the `C` operator.
- If `none`, the curve has no first control point, or equivalently, the control point defaults to the curve's starting point.

```
#curve(
  stroke: blue,
  curve.move((0pt, 50pt)),
  // - No start control point
  // - End control point at `(20pt, 0pt)`
  // - End point at `(50pt, 0pt)`
  curve.cubic(none, (20pt, 0pt), (50pt, 0pt)),
  // - No start control point
  // - No end control point
  // - End point at `(50pt, 0pt)`
  curve.cubic(none, none, (100pt, 50pt)),
)

#curve(
  stroke: blue,
  curve.move((0pt, 50pt)),
  curve.cubic(none, (20pt, 0pt), (50pt, 0pt)),
  // Passing `auto` instead of `none` means the start control point
  // mirrors the end control point of the previous curve. Mirror of
  // `(20pt, 0pt)` w.r.t `(50pt, 0pt)` is `(80pt, 0pt)`.
  curve.cubic(auto, none, (100pt, 50pt)),
)

#curve(
  stroke: blue,
  curve.move((0pt, 50pt)),
  curve.cubic(none, (20pt, 0pt), (50pt, 0pt)),
  // `(80pt, 0pt)` is the same as `auto` in this case.
  curve.cubic((80pt, 0pt), none, (100pt, 50pt)),
)
```



/3.4.2 **control-end** `none` or `array` *Required* *Positional*

The control point going into the end point of the curve segment.

If set to `none`, the curve has no end control point, or equivalently, the control point defaults to the curve's end point.

/3.4.3 **end** `array` *Required* *Positional*

The point at which the curve segment shall end.

/3.4.4 **relative** `bool` *Settable* Default: `false`

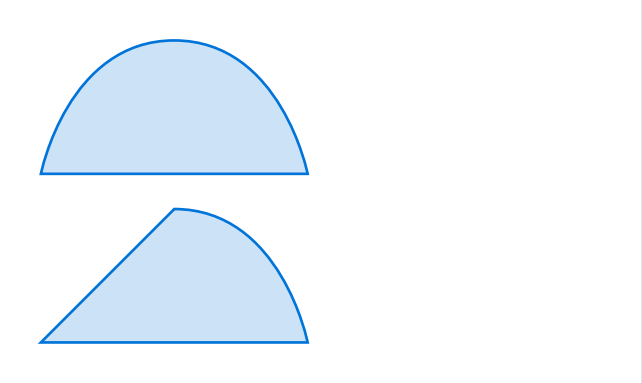
Whether the `control-start`, `control-end`, and `end` coordinates are relative to the previous point.

/3.5 **close** Element

Closes the curve by adding a segment from the last point to the start of the curve (or the last preceding `curve.move` point).

```
// We define a function to show the same shape with
// both closing modes.
#let shape(mode: "smooth") = curve(
  fill: blue.lighten(80%),
  stroke: blue,
  curve.move((0pt, 50pt)),
  curve.line((100pt, 50pt)),
  curve.cubic(auto, (90pt, 0pt), (50pt, 0pt)),
  curve.close(mode: mode),
)

#shape(mode: "smooth")
#shape(mode: "straight")
```



```
curve.close(mode: str) → content
```

/3.5.1 **mode** `str` *Settable*

Default: `"smooth"`

How to close the curve.

Variant	Details
`"smooth"`	Closes the curve with a smooth segment that takes into account the control point opposite the start point.
`"straight"`	Closes the curve with a straight line.

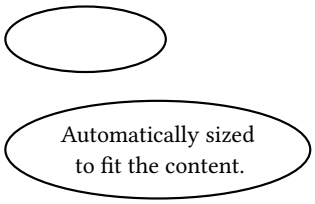
3.11.5 `ellipse` Element

An ellipse with optional content.

/1 Example

```
// Without content.
#ellipse(width: 35%, height: 30pt)

// With content.
#ellipse[
  #set align(center)
  Automatically sized \
  to fit the content.
]
```



/2 Parameters

```
ellipse(
  width: auto relative,
  height: auto relative fraction,
  fill: none color gradient tiling,
  stroke: none auto length color gradient stroke tiling dictionary,
  inset: relative dictionary,
  outset: relative dictionary,
  none content,
) → content
```

/2.1 **`width`** `auto` or `relative` *Settable* Default: `auto`

The ellipse's width, relative to its parent container.

/2.2 **`height`** `auto` or `relative` or `fraction` *Settable* Default: `auto`

The ellipse's height, relative to its parent container.

/2.3 **`fill`** `none` or `color` or `gradient` or `tiling` *Settable* Default: `none`

How to fill the ellipse. See the rectangle's documentation[3.11.10/2.3] for more details.

/2.4 **`stroke`** `none` or `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable* Default: `auto`

How to stroke the ellipse. See the rectangle's documentation[3.11.10/2.4] for more details.

/2.5 **`inset`** `relative` or `dictionary` *Settable* Default: `0% + 5pt`

How much to pad the ellipse's content. See the box's documentation[3.10.5/2.7] for more details.

/2.6 **`outset`** `relative` or `dictionary` *Settable* Default: `(:)`

How much to expand the ellipse's size without affecting the layout. See the box's documentation[3.10.5/2.8] for more details.

/2.7 **`body`** `none` or `content` *Positional* *Settable* Default: `none`

The content to place into the ellipse.

When this is omitted, the ellipse takes on a default size of at most `45pt` by `30pt`.

3.11.6 gradient

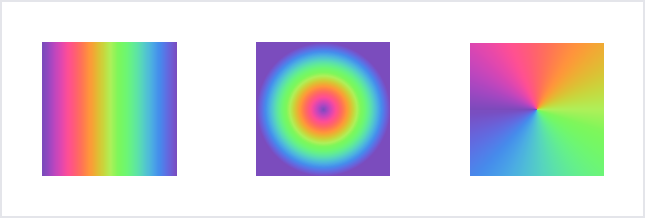
A color gradient.

Typst supports linear gradients through the gradient.linear function[3.11.6/7.1], radial gradients through the gradient.radial function[3.11.6/7.2], and conic gradients through the gradient.conic function[3.11.6/7.3].

A gradient can be used for the following purposes:
- As a fill to paint the interior of a shape: `rect(fill: gradient.linear(..))`
- As a stroke to paint the outline of a shape: `rect(stroke: 1pt + gradient.linear(..))`
- As the fill of text: `set text(fill: gradient.linear(..))`
- As a color map you can sample[3.11.6/7.15] from: `gradient.linear(..).sample(50%)`

/1 Examples

```
#stack(
  dir: ltr,
  spacing: 1fr,
  square(fill: gradient.linear(..color.map.rainbow)),
  square(fill: gradient.radial(..color.map.rainbow)),
  square(fill: gradient.conic(..color.map.rainbow)),
)
```



Gradients are also supported on text, but only when setting the relativeness[3.11.6/7.9] to either `auto` (the default value) or `"parent"`. To create word-by-word or glyph-by-glyph gradients, you can wrap the words or characters of your text in boxes[3.10.5] manually or through a show rule[3.2.2].

```
#set text(fill: gradient.linear(red, blue))
#let rainbow(content) = {
  set text(fill: gradient.linear(..color.map.rainbow))
  box(content)
}

This is a gradient on text, but with a #rainbow[twist]!
```



/2 Stops

A gradient is composed of a series of stops. Each of these stops has a color and an offset. The offset is a ratio[3.10.20] between `0%` and `100%` or an angle between `0deg` and `360deg`. The offset is a relative position that determines how far along the gradient the stop is located. The stop's color is the color of the gradient at that position. You can choose to omit the offsets when defining a gradient. In this case, Typst will space all stops evenly.

Typst predefines color maps that you can use as stops. See the color[3.11.3/3] documentation for more details.

/3 Relativeness

The location of the `0%` and `100%` stops depends on the dimensions of a container. This container can either be the shape that it is being painted on, or the closest surrounding container. This is controlled by the

`relative` argument of a gradient constructor. By default, gradients are relative to the shape they are being painted on, unless the gradient is applied on text, in which case they are relative to the closest ancestor container.

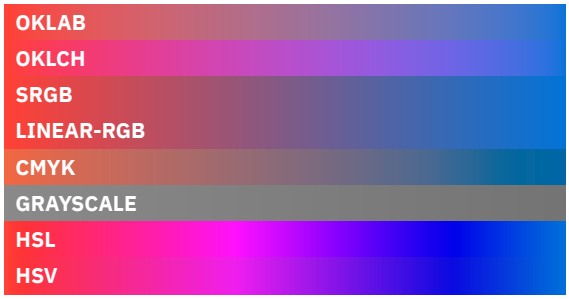
Typst determines the ancestor container as follows:
– For shapes that are placed at the root/top level of the document, the closest ancestor is the page itself.
– For other shapes, the ancestor is the innermost `block`[3.10.4] or `box`[3.10.5] that contains the shape.

/4 Color spaces and interpolation

Gradients can be interpolated in any color space. By default, gradients are interpolated in the Oklab[3.11.3/4.2] color space, which is a perceptually uniform[A] color space. This means that the gradient will be perceived as having a smooth progression of colors. This is particularly useful for data visualization.

However, you can choose to interpolate the gradient in any supported color space you want, but beware that some color spaces are not suitable for perceptually interpolating between colors. Consult the table below when choosing an interpolation space.

Color space	Perceptually uniform?
Oklab[3.11.3/4.2]	*Yes*
Oklch[3.11.3/4.3]	*Yes*
sRGB[3.11.3/4.5]	*No*
linear-RGB[3.11.3/4.4]	*Yes*
CMYK[3.11.3/4.6]	*No*
Grayscale[3.11.3/4.1]	*Yes*
HSL[3.11.3/4.7]	*No*
HSV[3.11.3/4.8]	*No*

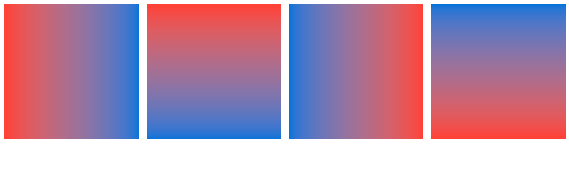


/5 Direction

Some gradients are sensitive to direction. For example, a linear gradient has an angle that determines its direction. Typst uses a clockwise angle, with 0° being from left to right, 90° from top to bottom, 180° from right to left, and 270° from bottom to top.

```
#stack(
  dir: ltr,
  spacing: 1fr,
  square(fill: gradient.linear(red, blue, angle: 0deg)),
  square(fill: gradient.linear(red, blue, angle: 90deg)),
  square(fill: gradient.linear(red, blue, angle: 180deg)),
  square(fill: gradient.linear(red, blue, angle: 270deg)),
)
```

[A] https://programmingdesignsystems.com/color/perceptually-uniform-color-spaces/index.html



/6 Note on file sizes

Gradients can be quite large, especially if they have many stops. This is because gradients are stored as a list of colors and offsets, which can take up a lot of space. In SVG export, gradients are stored as a list of `color.rgb`[3.11.3/4.5] colors replicating the original color space with an optimized number of extra stops in between. In PDF export, the same applies to gradients in the `color.oklab`[3.11.3/4.2], `color.oklch`[3.11.3/4.3], `color.hsv`[3.11.3/4.8], `color.hsl`[3.11.3/4.7], and `color.linear-rgb`[3.11.3/4.4] color spaces. This avoids needing to encode these color spaces in your PDF file, but it does add extra stops to your gradient, which can increase the file size.

/7 Definitions

/7.1 `linear`

Creates a new linear gradient, in which colors transition along a straight line.

```
#rect(
  width: 100%,
  height: 20pt,
  fill: gradient.linear(
    ..color.map.viridis,
  ),
)
```



```
gradient.linear(
  ..color array ,
  space: auto spot any ,
  relative: auto str ,
  direction ,
  angle ,
) → gradient
```

/7.1.1 `stops` `color` or `array` *Required* *Positional* *Variadic*

The color stops[3.11.6/2] of the gradient.

/7.1.2 `space` `auto` or `spot` or `any` Default: `auto`

The color space in which to interpolate the gradient.

Defaults to a perceptually uniform color space called Oklab[3.11.3/4.2].

/7.1.3 `relative` `auto` or `str` Default: `auto`

The relative placement[3.11.6/3] of the gradient.

The parent of an element is the innermost `box`[3.10.5] or `block`[3.10.4] that contains the element, or, if there is none, the page itself.

Variant	Details
`"self"`	Relative to itself (its own bounding box).
`"parent"`	Relative to its parent (the parent's bounding box).

/7.1.4 **dir** direction *Positional* Default: ltr

The direction of the gradient.

/7.1.5 **angle** angle *Required* *Positional*

The angle of the gradient.

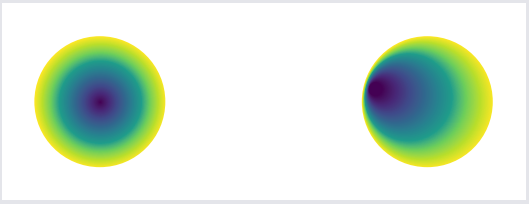
/7.2 radial

Creates a new radial gradient, in which colors radiate away from an origin.

The gradient is defined by two circles: the focal circle and the end circle. The focal circle is a circle with center `focal-center` and radius `focal-radius`, that defines the points at which the gradient starts and has the color of the first stop. The end circle is a circle with center `center` and radius `radius`, that defines the points at which the gradient ends and has the color of the last stop. The gradient is then interpolated between these two circles.

Using these four values, also called the focal point for the starting circle and the center and radius for the end circle, we can define a gradient with more interesting properties than a basic radial gradient.

```
#stack(
  dir: ltr,
  spacing: 1fr,
  circle(fill: gradient.radial(
    ..color.map.viridis,
  )),
  circle(fill: gradient.radial(
    ..color.map.viridis,
    focal-center: (10%, 40%),
    focal-radius: 5%,
  )),
)
```



```
gradient.radial(
  ..color array,
  space: auto spot any,
  relative: auto str,
  center: array,
  radius: ratio,
  focal-center: auto array,
  focal-radius: ratio,
) → gradient
```

/7.2.1 **stops** color or array *Required* *Positional* *Variadic*

The color stops[3.11.6/2] of the gradient.

/7.2.2 **space** auto or spot or any Default: auto

The color space in which to interpolate the gradient.

Defaults to a perceptually uniform color space called Oklab[3.11.3/4.2].

/7.2.3 **relative** auto or str Default: auto

The relative placement[3.11.6/3] of the gradient.

The parent of an element is the innermost box[3.10.5] or block[3.10.4] that contains the element, or, if there is none, the page itself.

Variant	Details
"self"	Relative to itself (its own bounding box).
"parent"	Relative to its parent (the parent's bounding box).

/7.2.4 **center** array — Default: (50%, 50%)

The center of the end circle of the gradient.

A value of (50%, 50%) means that the end circle is centered inside of its container.

/7.2.5 **radius** ratio — Default: 50%

The radius of the end circle of the gradient.

By default, it is set to 50%. The ending radius must be bigger than the focal radius.

/7.2.6 **focal-center** auto or array — Default: auto

The center of the focal circle of the gradient.

The focal center must be inside of the end circle.

A value of (50%, 50%) means that the focal circle is centered inside of its container.

By default it is set to the same as the center of the last circle.

/7.2.7 **focal-radius** ratio — Default: 0%

The radius of the focal circle of the gradient.

The focal center must be inside of the end circle.

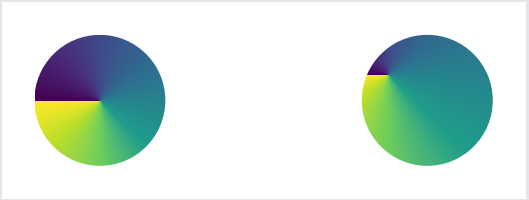
By default, it is set to 0%. The focal radius must be smaller than the ending radius.

/7.3 conic

Creates a new conic gradient, in which colors change radially around a center point.

You can control the center point of the gradient by using the `center` argument. By default, the center point is the center of the shape.

```
#stack(
  dir: ltr,
  spacing: 1fr,
  circle(fill: gradient.conic(
    ..color.map.viridis,
  )),
  circle(fill: gradient.conic(
    ..color.map.viridis,
    center: (20%, 30%),
  )),
)
```



```
gradient.conic(
  ..color array ,
  angle: angle ,
  space: auto spot any ,
  relative: auto str ,
  center: array ,
) → gradient
```

/7.3.1 **stops** color or array *Required* *Positional* *Variadic*

The color stops[3.11.6/2] of the gradient.

/7.3.2 **angle** angle — Default: 0deg

The angle of the gradient.

/7.3.3 **space** auto or spot or any — Default: auto

The color space in which to interpolate the gradient.

Defaults to a perceptually uniform color space called Oklab[3.11.3/4.2].

/7.3.4 **relative** auto or str Default: auto

The relative placement[3.11.6/3] of the gradient.

The parent of an element is the innermost box[3.10.5] or block[3.10.4] that contains the element, or, if there is none, the page itself.

Variant	Details
"self"	Relative to itself (its own bounding box).
"parent"	Relative to its parent (the parent's bounding box).

/7.3.5 **center** array Default: (50%, 50%)

The center of the circle of the gradient.

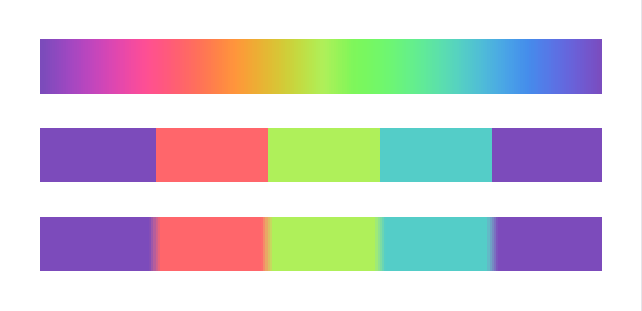
A value of (50%, 50%) means that the circle is centered inside of its container.

/7.4 sharp

Creates a sharp version of this gradient.

Sharp gradients have discrete jumps between colors, instead of a smooth transition. They are particularly useful for creating color lists for a preset gradient.

```
#set rect(width: 100%, height: 20pt)
#let grad = gradient.linear(..color.map.rainbow)
#rect(fill: grad)
#rect(fill: grad.sharp(5))
#rect(fill: grad.sharp(5, smoothness: 20%))
```



```
self.sharp(
  int,
  smoothness: ratio,
) → gradient
```

/7.4.1 **steps** int *Required* *Positional*

The number of stops in the gradient.

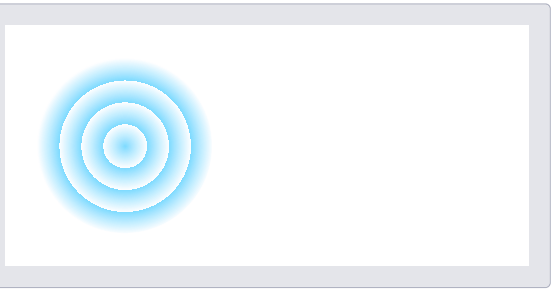
/7.4.2 **smoothness** ratio Default: 0%

How much to smooth the gradient.

/7.5 repeat

Repeats this gradient a given number of times, optionally mirroring it at every second repetition.

```
#circle(
  radius: 40pt,
  fill: gradient
    .radial(aqua, white)
    .repeat(4),
)
```



```
self.repeat(
  int,
  mirror: bool,
) → gradient
```

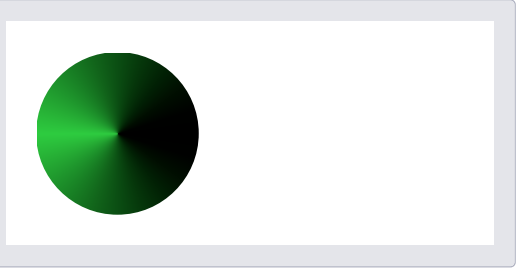
/7.5.1 **repetitions** `int` *Required* *Positional*

The number of times to repeat the gradient.

/7.5.2 **mirror** `bool` Default: `false`

Whether to mirror the gradient at every second repetition, i.e., the first instance (and all odd ones) stays unchanged.

```
#circle(
  radius: 40pt,
  fill: gradient
    .conic(green, black)
    .repeat(2, mirror: true)
)
```



/7.6 **kind**

Returns the kind of this gradient.

```
self.kind() → function
```

/7.7 **stops**

Returns the stops of this gradient.

```
self.stops() → array
```

/7.8 **space**

Returns the mixing space of this gradient.

```
self.space() → spot any
```

/7.9 **relative**

Returns the relative placement of this gradient.

```
self.relative() → auto str
```

/7.10 angle

Returns the angle of this gradient.

Returns none if the gradient is neither linear nor conic.

```
self.angle() → none angle
```

/7.11 center

Returns the center of this gradient.

Returns none if the gradient is neither radial nor conic.

```
self.center() → none array
```

/7.12 radius

Returns the radius of this gradient.

Returns none if the gradient is not radial.

```
self.radius() → none ratio
```

/7.13 focal-center

Returns the focal-center of this gradient.

Returns none if the gradient is not radial.

```
self.focal-center() → none array
```

/7.14 focal-radius

Returns the focal-radius of this gradient.

Returns none if the gradient is not radial.

```
self.focal-radius() → none ratio
```

/7.15 sample

Sample the gradient at a given position.

The position is either a position along the gradient (a ratio[3.10.20] between 0% and 100%) or an angle[3.10.3]. Any value outside of this range will be clamped.

```
self.sample( angle ratio ) → color
```

/7.15.1 t angle or ratio *Required* *Positional*

The position at which to sample the gradient.

/7.16 samples

Samples the gradient at multiple positions at once and returns the results as an array.

```
self.samples(.. angle ratio ) → array
```

/7.16.1 **ts** `angle` or `ratio` *Required* *Positional* *Variadic*

The positions at which to sample the gradient.

3.11.7 `image` Element

A raster or vector graphic.

You can wrap the image in a `figure`[3.6.8] to give it a number and caption.

Like most elements, images are *block-level* by default and thus do not integrate themselves into adjacent paragraphs. To force an image to become inline, put it into a `box`[3.10.5].

/1 Example

```
#figure(
  image("molecular.jpg", width: 80%),
  caption: [
    A step in the molecular testing
    pipeline of our lab.
  ],
)
```

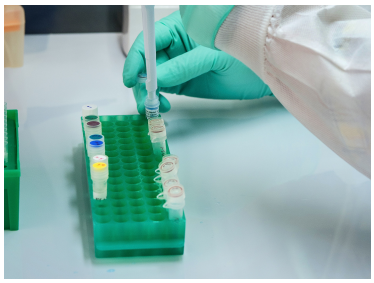


Figure 1: A step in the molecular testing pipeline of our lab.

/2 Parameters

```
image(
  str path bytes,
  format: auto str dictionary,
  width: auto relative,
  height: auto relative fraction,
  alt: none str,
  page: int,
  fit: str,
  scaling: auto str,
  icc: auto str path bytes,
) → content
```

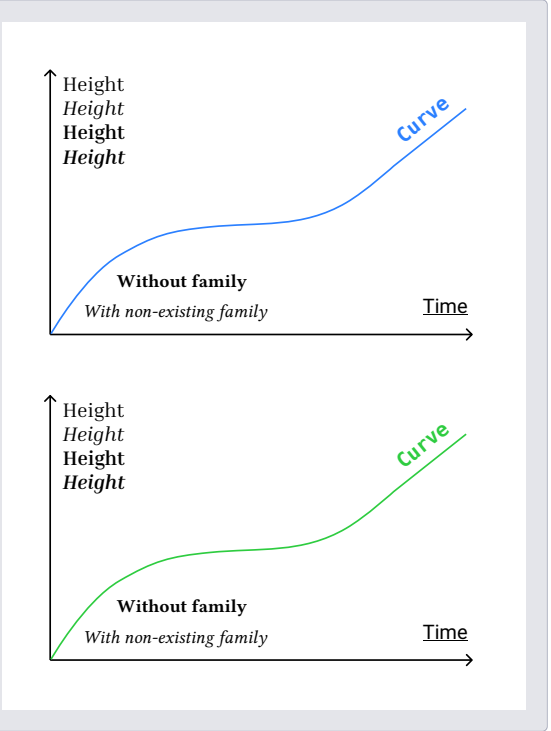
/2.1 `source` `str` or `path` or `bytes` *Required* *Positional*

A path to an image file or raw bytes making up an image in one of the supported formats[3.11.7/2.2].

Bytes can be used to specify raw pixel data in a row-major, left-to-right, top-to-bottom format.

```
#let original = read("diagram.svg")
#let changed = original.replace(
  "#2B80FF", // blue
  green.to-hex(),
)

#image(bytes(original))
#image(bytes(changed))
```



/2.2 **format** `auto` or `str` or `dictionary` *Settable* Default: `auto`

The image's format.

By default, the format is detected automatically. Typically, you thus only need to specify this when providing raw bytes as the `source`[3.11.7/2.1] (even then, Typst will try to figure out the format automatically, but that's not always possible).

Supported formats are `"png"`, `"jpg"`, `"gif"`, `"svg"`, `"pdf"`, `"webp"` as well as raw pixel data.

Note that several restrictions apply when using PDF files as images:

- When exporting to PDF, any PDF image file used must have a version equal to or lower than the export target PDF version[3.14.2/1].
- PDF files as images are currently not supported when exporting with a specific PDF standard, like PDF/A-3 or PDF/UA-1. In these cases, you can instead use SVGs to embed vector images.
- The image file must not be password-protected.
- Tags in your PDF image will not be preserved. Instead, you must provide an alternative description[3.11.7/2.5] to make the image accessible.

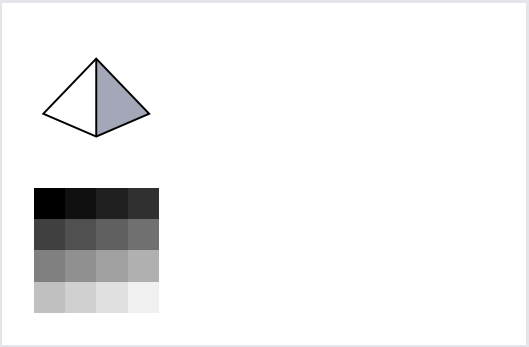
When providing raw pixel data as the `source`, you must specify a dictionary with the following keys as the `format`:

- `encoding` (str[3.5.27]): The encoding of the pixel data. One of:
  - `"rgb8"` (three 8-bit channels: red, green, blue)
  - `"rgba8"` (four 8-bit channels: red, green, blue, alpha)
  - `"luma8"` (one 8-bit channel)
  - `"lumaa8"` (two 8-bit channels: luma and alpha)
- `width` (int[3.5.16]): The pixel width of the image.
- `height` (int[3.5.16]): The pixel height of the image.

The pixel width multiplied by the height multiplied by the channel count for the specified encoding must then match the `source` data.

```
#image(
  read(
    "tetrahedron.svg",
    encoding: none,
  ),
  format: "svg",
  width: 2cm,
)

#image(
  bytes(range(16).map(x => x * 16)),
  format: (
    encoding: "luma8",
    width: 4,
    height: 4,
  ),
  width: 2cm,
)
```



Variant	Details
`"png"`	Raster format for illustrations and transparent graphics.
`"jpg"`	Lossy raster format suitable for photos.
`"gif"`	Raster format that is typically used for short animated clips. Typst can load GIFs, but they will become static.
`"webp"`	Raster format that supports both lossy and lossless compression.
`"svg"`	The vector graphics format of the web.
`"pdf"`	High-fidelity document and graphics format, with focus on exact reproduction in print.

/2.3 **`width`** `auto` or `relative` *Settable* Default: `auto`

The width of the image.

/2.4 **`height`** `auto` or `relative` or `fraction` *Settable* Default: `auto`

The height of the image.

/2.5 **`alt`** `none` or `str` *Settable* Default: `none`

An alternative description of the image.

This text is used by Assistive Technology (AT) like screen readers to describe the image to users with visual impairments.

When the image is wrapped in a `figure`[3.6.8], use this parameter rather than the figure's `alt` parameter[3.6.8/5.2] to describe the image. The only exception to this rule is when the image and the other contents in the figure form a single semantic unit. In this case, use the figure's `alt` parameter to describe the entire composition and do not use this parameter.

You can learn how to write good alternative descriptions in the Accessibility Guide[4.4.7].

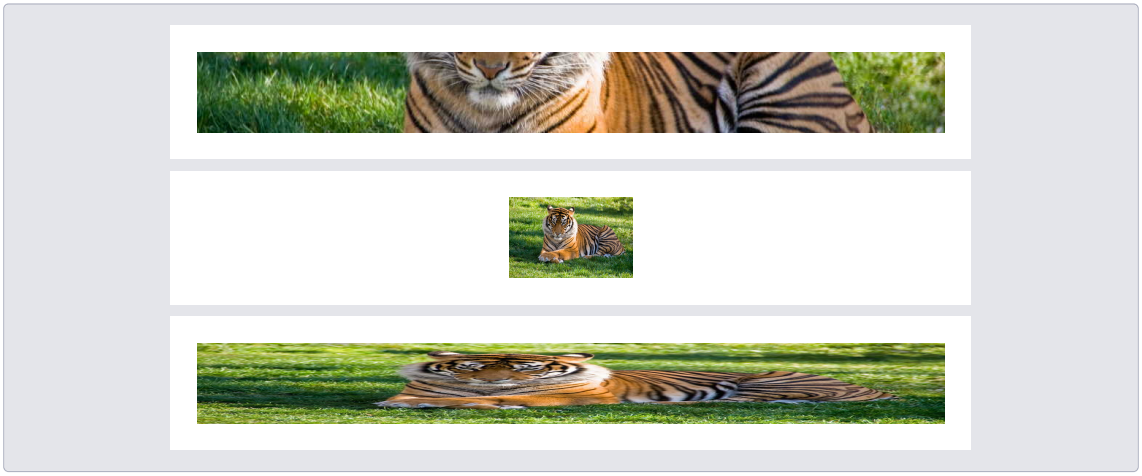
/2.6 **`page`** `int` *Settable* Default: `1`

The page number that should be embedded as an image. This attribute only has an effect for PDF files.

/2.7 **`fit`** `str` *Settable* Default: `"cover"`

How the image should adjust itself to a given area (the area is defined by the `width` and `height` fields). Note that `fit` doesn't visually change anything if the area's aspect ratio is the same as the image's one.

```
#set page(width: 300pt, height: 50pt, margin: 10pt)
#image("tiger.jpg", width: 100%, fit: "cover")
#image("tiger.jpg", width: 100%, fit: "contain")
#image("tiger.jpg", width: 100%, fit: "stretch")
```



Variant	Details
`"cover"`	The image should completely cover the area (preserves aspect ratio by cropping the image only horizontally or vertically). This is the default.
`"contain"`	The image should be fully contained in the area (preserves aspect ratio; doesn't crop the image; one dimension can be narrower than specified).
`"stretch"`	The image should be stretched so that it exactly fills the area, even if this means that the image will be distorted (doesn't preserve aspect ratio and doesn't crop the image).

/2.8 **`scaling`** `auto` or `str` *Settable* Default: `auto`

A hint to viewers how they should scale the image.

When set to `auto`, the default is left up to the viewer. For PNG export, Typst will default to smooth scaling, like most PDF and SVG viewers.

Note: The exact look may differ across PDF viewers.

Variant	Details
`"smooth"`	Scale with a smoothing algorithm such as bilinear interpolation.
`"pixelated"`	Scale with nearest neighbor or a similar algorithm to preserve the pixelated look of the image.

/2.9 **`icc`** `auto` or `str` or `path` or `bytes` *Settable* Default: `auto`

An ICC profile for the image.

ICC profiles define how to interpret the colors in an image. When set to `auto`, Typst will try to extract an ICC profile from the image.

3.11.8 `line` Element

A line from one point to another.

/1 Example

```
#set page(height: 100pt)

#line(length: 100%)
#line(end: (50%, 50%))
#line(
  length: 4cm,
  stroke: 2pt + maroon,
)
```



/2 Parameters

```
line(
  start: array,
  end: none array,
  length: relative,
  angle: angle,
  stroke: length color gradient stroke tiling dictionary,
) → content
```

/2.1 **start** `array` *Settable* Default: `(0% + 0pt, 0% + 0pt)`

The start point of the line.

Must be an array of exactly two relative lengths.

/2.2 **end** `none` or `array` *Settable* Default: `none`

The point where the line ends.

/2.3 **length** `relative` *Settable* Default: `0% + 30pt`

The line's length. This is only respected if `end` is `none`.

/2.4 **angle** `angle` *Settable* Default: `0deg`

The angle at which the line points away from the origin. This is only respected if `end` is `none`.

/2.5 **stroke** `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable*

Default: `1pt + black`

How to stroke[3.11.12] the line.

```
#set line(length: 100%)
#stack(
  spacing: 1em,
  line(stroke: 2pt + red),
  line(stroke: (paint: blue, thickness: 4pt, cap: "round")),
  line(stroke: (paint: blue, thickness: 1pt, dash: "dashed")),
  line(stroke: (paint: blue, thickness: 1pt, dash: ("dot", 2pt, 4pt, 2pt))),
)
```



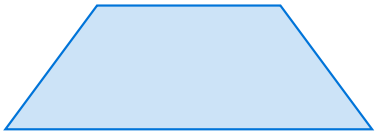
3.11.9 `polygon` Element

A closed polygon.

The polygon is defined by its corner points and is closed automatically.

/1 Example

```
#polygon(
  fill: blue.lighten(80%),
  stroke: blue,
  (20%, 0pt),
  (60%, 0pt),
  (80%, 2cm),
  (0%,  2cm),
)
```



/2 Parameters

```
polygon(
  fill: none color gradient tiling,
  fill-rule: str,
  stroke: none auto length color gradient stroke tiling dictionary,
  ..array,
) → content
```

/2.1 `fill` `none` or `color` or `gradient` or `tiling` *Settable* Default: `none`

How to fill the polygon.

When setting a fill, the default stroke disappears. To create a rectangle with both fill and stroke, you have to configure both.

/2.2 `fill-rule` `str` *Settable* Default: `"non-zero"`

The drawing rule used to fill the polygon.

See the curve documentation[3.11.4/2.2] for an example.

Variant	Details
`"non-zero"`	Specifies that "inside" is computed by a non-zero sum of signed edge crossings.
`"even-odd"`	Specifies that "inside" is computed by an odd number of edge crossings.

/2.3 `stroke` `none` or `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable* Default: `auto`

How to stroke[3.11.12] the polygon.

Can be set to `none` to disable the stroke or to `auto` for a stroke of `1pt` black if and only if no fill is given.

/2.4 `vertices` `array` *Required* *Positional* *Variadic*

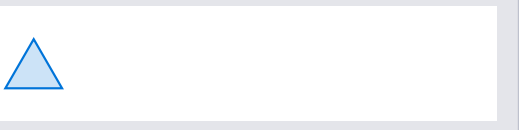
The vertices of the polygon. Each point is specified as an array of two relative lengths[3.10.21].

/3 Definitions

/3.1 `regular`

A regular polygon, defined by its size and number of vertices.

```
#polygon.regular(
  fill: blue.lighten(80%),
  stroke: blue,
  size: 30pt,
  vertices: 3,
)
```



```
polygon.regular(
  fill: none color gradient tiling,
  stroke: none auto length color gradient stroke tiling dictionary,
  size: length,
  vertices: int,
) → content
```

/3.1.1 **fill** `none` or `color` or `gradient` or `tiling`

How to fill the polygon. See the general polygon's documentation[3.11.9/2.1] for more details.

/3.1.2 **stroke** `none` or `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary`

How to stroke the polygon. See the general polygon's documentation[3.11.9/2.3] for more details.

/3.1.3 **size** `length` Default: `1em`

The diameter of the circumcircle[A] of the regular polygon.

/3.1.4 **vertices** `int` Default: `3`

The number of vertices in the polygon.

[A] https://en.wikipedia.org/wiki/Circumcircle

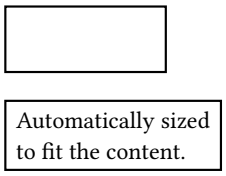
3.11.10 `rect` Element

A rectangle with optional content.

/1 Example

```
// Without content.
#rect(width: 35%, height: 30pt)

// With content.
#rect[
  Automatically sized \
  to fit the content.
]
```



Automatically sized
to fit the content.

/2 Parameters

```
rect(
  width: auto relative,
  height: auto relative fraction,
  fill: none color gradient tiling,
  stroke: none auto length color gradient stroke tiling dictionary,
  radius: relative dictionary,
  inset: relative dictionary,
  outset: relative dictionary,
  none content,
) → content
```

/2.1 `width` `auto` or `relative` *Settable* Default: `auto`

The rectangle's width, relative to its parent container.

/2.2 `height` `auto` or `relative` or `fraction` *Settable* Default: `auto`

The rectangle's height, relative to its parent container.

/2.3 `fill` `none` or `color` or `gradient` or `tiling` *Settable* Default: `none`

How to fill the rectangle.

When setting a fill, the default stroke disappears. To create a rectangle with both fill and stroke, you have to configure both.

```
#rect(fill: blue)
```



/2.4 `stroke` `none` or `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable* Default: `auto`

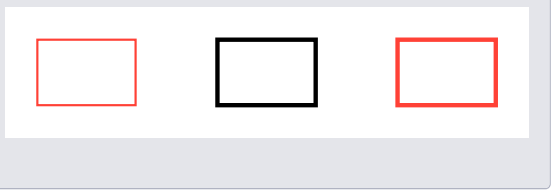
How to stroke the rectangle. This can be:

- `none` to disable stroking
- `auto` for a stroke of `1pt + black` if and only if no fill is given.
- Any kind of stroke[3.11.12]
- A dictionary describing the stroke for each side individually. The dictionary can contain the following keys in order of precedence:
  - `top`: The top stroke.

- `right`: The right stroke.
- `bottom`: The bottom stroke.
- `left`: The left stroke.
- `x`: The left and right stroke.
- `y`: The top and bottom stroke.
- `rest`: The stroke on all sides except those for which the dictionary explicitly sets a size.

All keys are optional; omitted keys will use their previously set value, or the default stroke if never set.

```
#stack(
  dir: ltr,
  spacing: 1fr,
  rect(stroke: red),
  rect(stroke: 2pt),
  rect(stroke: 2pt + red),
)
```

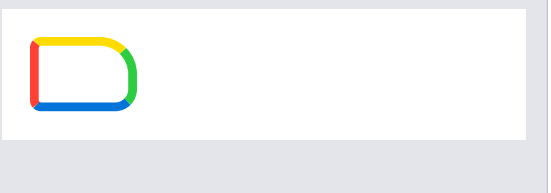


/2.5 **radius** `relative` or `dictionary` *Settable* Default: `(:)`

How much to round the rectangle's corners, relative to the minimum of the width and height divided by two. This can be:

- A relative length for a uniform corner radius.
- A dictionary: With a dictionary, the stroke for each side can be set individually. The dictionary can contain the following keys in order of precedence:
  - `top-left`: The top-left corner radius.
  - `top-right`: The top-right corner radius.
  - `bottom-right`: The bottom-right corner radius.
  - `bottom-left`: The bottom-left corner radius.
  - `left`: The top-left and bottom-left corner radii.
  - `top`: The top-left and top-right corner radii.
  - `right`: The top-right and bottom-right corner radii.
  - `bottom`: The bottom-left and bottom-right corner radii.
  - `rest`: The radii for all corners except those for which the dictionary explicitly sets a size.

```
#set rect(stroke: 4pt)
#rect(
  radius: (
    left: 5pt,
    top-right: 20pt,
    bottom-right: 10pt,
  ),
  stroke: (
    left: red,
    top: yellow,
    right: green,
    bottom: blue,
  ),
)
```



/2.6 **inset** `relative` or `dictionary` *Settable* Default: `0% + 5pt`

How much to pad the rectangle's content. See the box's documentation[3.10.5/2.7] for more details.

/2.7 **outset** `relative` or `dictionary` *Settable* Default: `(:)`

How much to expand the rectangle's size without affecting the layout. See the box's documentation[3.10.5/2.8] for more details.

/2.8 **body** `none` or `content` *Positional* *Settable* Default: `none`

The content to place into the rectangle.

When this is omitted, the rectangle takes on a default size of at most `45pt` by `30pt`.

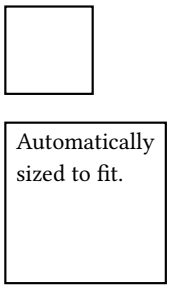
3.11.11 `square` Element

A square with optional content.

/1 Example

```
// Without content.
#square(size: 40pt)

// With content.
#square[
  Automatically \
  sized to fit.
]
```



/2 Parameters

```
square(
  size: auto length,
  width: auto relative,
  height: auto relative fraction,
  fill: none color gradient tiling,
  stroke: none auto length color gradient stroke tiling dictionary,
  radius: relative dictionary,
  inset: relative dictionary,
  outset: relative dictionary,
  none content,
) → content
```

/2.1 `size` `auto` or `length` Default: `auto`

The square's side length. This is mutually exclusive with `width` and `height`.

/2.2 `width` `auto` or `relative` *Settable* Default: `auto`

The square's width. This is mutually exclusive with `size` and `height`.

In contrast to `size`, this can be relative to the parent container's width.

/2.3 `height` `auto` or `relative` or `fraction` *Settable* Default: `auto`

The square's height. This is mutually exclusive with `size` and `width`.

In contrast to `size`, this can be relative to the parent container's height.

/2.4 `fill` `none` or `color` or `gradient` or `tiling` *Settable* Default: `none`

How to fill the square. See the rectangle's documentation[3.11.10/2.3] for more details.

/2.5 `stroke` `none` or `auto` or `length` or `color` or `gradient` or `stroke` or `tiling` or `dictionary` *Settable* Default: `auto`

How to stroke the square. See the rectangle's documentation[3.11.10/2.4] for more details.

/2.6 `radius` `relative` or `dictionary` *Settable* Default: `(:)`

How much to round the square's corners. See the rectangle's documentation[3.11.10/2.5] for more details.

/2.7 **inset** `relative` or `dictionary` *Settable* Default: `0% + 5pt`

How much to pad the square's content. See the box's documentation[3.10.5/2.7] for more details.

/2.8 **outset** `relative` or `dictionary` *Settable* Default: `(:)`

How much to expand the square's size without affecting the layout. See the box's documentation[3.10.5/2.8] for more details.

/2.9 **body** `none` or `content` *Positional* *Settable* Default: `none`

The content to place into the square. The square expands to fit this content, keeping the 1-1 aspect ratio.

When this is omitted, the square takes on a default size of at most `30pt`.

3.11.12 # stroke

Defines how to draw a line.

A stroke has a *paint* (a solid color or gradient), a *thickness,* a line *cap,* a line *join,* a *miter limit,* and a *dash* pattern. All of these values are optional and have sensible defaults.

/1 Example

```
#set line(length: 100%)
#stack(
  spacing: 1em,
  line(stroke: 2pt + red),
  line(stroke: (paint: blue, thickness: 4pt, cap: "round")),
  line(stroke: (paint: blue, thickness: 1pt, dash: "dashed")),
  line(stroke: 2pt + gradient.linear(..color.map.rainbow)),
)
```



/2 Simple strokes

You can create a simple solid stroke from a color, a thickness, or a combination of the two. Specifically, wherever a stroke is expected you can pass any of the following values:

- A length specifying the stroke's thickness. The color is inherited, defaulting to black.
- A color to use for the stroke. The thickness is inherited, defaulting to `1pt`.
- A stroke combined from color and thickness using the `+` operator as in `2pt + red`.

For full control, you can also provide a dictionary[3.5.11] or a `stroke` object to any function that expects a stroke. The dictionary's keys may include any of the parameters for the constructor function, shown below.

/3 Fields

On a stroke object, you can access any of the fields listed in the constructor function. For example, `(2pt + blue).thickness` is `2pt`. Meanwhile, `stroke(red).cap` is `auto` because it's unspecified. Fields set to `auto` are inherited.

/4 Constructor

Converts a value to a stroke or constructs a stroke with the given parameters.

Note that in most cases you do not need to convert values to strokes in order to use them, as they will be converted automatically. However, this constructor can be useful to ensure a value has all the fields of a stroke.

```
#let my-func(x) = {
    x = stroke(x) // Convert to a stroke
    [Stroke has thickness #x.thickness.]
}
#my-func(3pt) \
#my-func(red) \
#my-func(stroke(cap: "round", thickness: 1pt))
```

Stroke has thickness 3pt.
Stroke has thickness auto.
Stroke has thickness 1pt.

```
stroke(
  auto color gradient tiling,
  auto length,
  auto str,
  auto str,
  none auto str array dictionary,
  auto float,
) → stroke
```

/4.1 **paint** auto or color or gradient or tiling *Required* *Positional*

The color or gradient to use for the stroke.

If set to auto, the value is inherited, defaulting to black.

/4.2 **thickness** auto or length *Required* *Positional*

The stroke's thickness.

If set to auto, the value is inherited, defaulting to 1pt.

/4.3 **cap** auto or str *Required* *Positional*

How the ends of the stroke are rendered.

If set to auto, the value is inherited, defaulting to "butt".

Variant	Details
"butt"	Square stroke cap with the edge at the stroke's end point.
"round"	Circular stroke cap centered at the stroke's end point.
"square"	Square stroke cap centered at the stroke's end point.

/4.4 **join** auto or str *Required* *Positional*

How sharp turns are rendered.

If set to auto, the value is inherited, defaulting to "miter".

Variant	Details
"miter"	Segments are joined with sharp edges. Sharp bends exceeding the miter limit are bevelled instead.
"round"	Segments are joined with circular corners.
"bevel"	Segments are joined with a bevel (a straight edge connecting the butts of the joined segments).

/4.5 **dash** none or auto or str or array or dictionary *Required* *Positional*

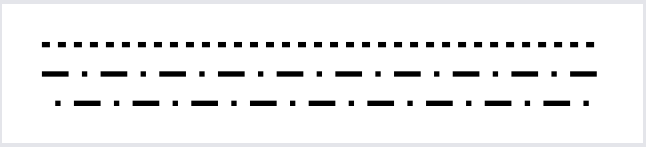
The dash pattern to use. This can be:

- One of the predefined patterns listed below.
- none, as an equivalent to the predefined pattern "solid".
- An array[3.5.2] with alternating lengths for dashes and gaps. You can also use the string "dot" for a length equal to the line thickness.
- A dictionary[3.5.11] with the keys array (same as the array above), and phase (of type length[3.10.13]), which defines where in the pattern to start drawing.

If set to auto, the value is inherited, defaulting to none.

Example · Usage

```
#set line(length: 100%, stroke: 2pt)
#stack(
  spacing: 1em,
  line(stroke: (dash: "dashed")),
  line(stroke: (dash: (10pt, 5pt, "dot", 5pt))),
  line(stroke: (dash: (array: (10pt, 5pt, "dot", 5pt), phase: 10pt))),
)
```



Example · Predefined patterns

```
#set page(width: auto, height: auto)

#let patterns = (
  ("dotted", "dashed", "dash-dotted").map(d => (
    "loosely-" + d,
    d,
    "densely-" + d,
  ))
).flatten()


#let display(dash) = {
  set par(spacing: 0.3em)

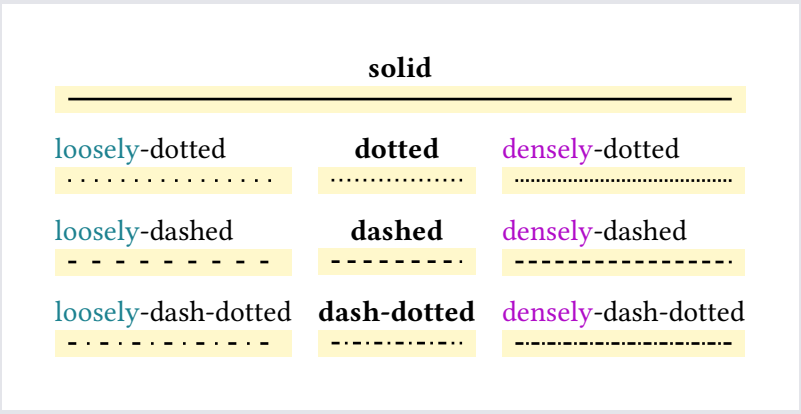
  show "loosely": set text(eastern.darken(17%))
  show "densely": set text(purple)

  let derived = dash.starts-with("loosely-") or dash.starts-with("densely-")
  if derived { dash } else { strong(dash) }

  rect(
    line(stroke: (dash: dash), length: 100%),
    stroke: none,
    fill: yellow.transparentize(80%),
  )
}

#table(
  columns: 3,
  stroke: none,
  align: (left, center, left),

  table.cell(colspan: 3, align: center, display("solid")),
  ..patterns.map(display),
)
```



Variant	Details
`"solid"`	`()`
`"dotted"`	`("dot", 2pt)`
`"densely-dotted"`	`("dot", 1pt)`
`"loosely-dotted"`	`("dot", 4pt)`
`"dashed"`	`(3pt, 3pt)`
`"densely-dashed"`	`(3pt, 2pt)`
`"loosely-dashed"`	`(3pt, 6pt)`
`"dash-dotted"`	`(3pt, 2pt, "dot", 2pt)`
`"densely-dash-dotted"`	`(3pt, 1pt, "dot", 1pt)`
`"loosely-dash-dotted"`	`(3pt, 4pt, "dot", 4pt)`

/4.6 `miter-limit` `auto` or `float` *Required* *Positional*

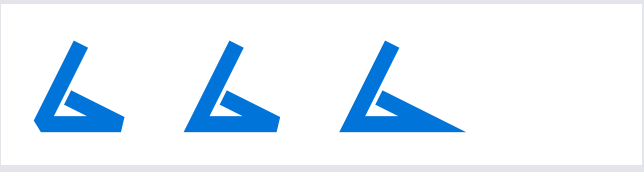
Number at which protruding sharp bends are rendered with a bevel instead or a miter join. The higher the number, the sharper an angle can be before it is bevelled. Only applicable if `join` is `"miter"`.

Specifically, the miter limit is the maximum ratio between the corner's protrusion length and the stroke's thickness.

If set to `auto`, the value is inherited, defaulting to `4.0`.

```
#let items = (
  curve.move((15pt, 0pt)),
  curve.line((0pt, 30pt)),
  curve.line((30pt, 30pt)),
  curve.line((10pt, 20pt)),
)

#set curve(stroke: 6pt + blue)
#stack(
  dir: ltr,
  spacing: 1cm,
  curve(stroke: (miter-limit: 1), ..items),
  curve(stroke: (miter-limit: 4), ..items),
  curve(stroke: (miter-limit: 5), ..items),
)
```



3.11.13 `tiling`

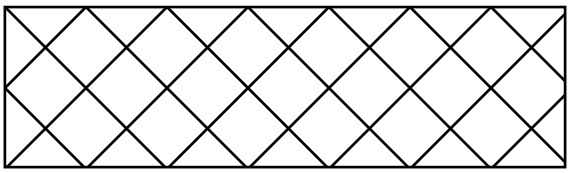
A repeating tiling fill.

Typst supports the most common type of tilings, where a pattern is repeated in a grid-like fashion, covering the entire area of an element that is filled or stroked. The pattern is defined by a tile `size`[3.11.13/3.1] and a body defining the content of each cell. You can also add horizontal or vertical `spacing`[3.11.13/3.2] between the cells of the tiling and `offset`[3.11.13/3.3] the starting position of the tiling.

/1 Example

```
#let pat = tiling(size: (30pt, 30pt), {
  place(line(start: (0%, 0%), end: (100%, 100%)))
  place(line(start: (0%, 100%), end: (100%, 0%)))
})

#rect(fill: pat, width: 100%, height: 60pt, stroke: 1pt)
```



/2 Tilings on text

Tilings are also supported on text, but only when setting `relative`[3.11.13/3.4] to either `auto` (the default value) or `"parent"`. To create word-by-word or glyph-by-glyph tilings, you can wrap the words or characters of your text in boxes[3.10.5] manually or through a show rule[3.2.2].

```
#let pat = tiling(
  size: (30pt, 30pt),
  relative: "parent",
  square(
    size: 30pt,
    fill: gradient
      .conic(..color.map.rainbow),
  )
)

#set text(fill: pat)
#lorem(10)
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

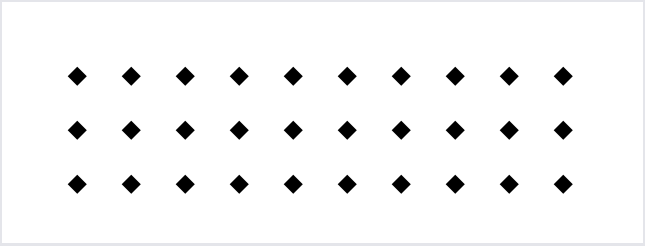
/3 Constructor

Construct a new tiling.

```
#let pat = tiling(
  size: (20pt, 20pt),
  relative: "parent",
  place(
    dx: 5pt,
    dy: 5pt,
    rotate(45deg, square(
      size: 5pt,
      fill: black,
    )),
```

```
  ),
)

#rect(width: 100%, height: 60pt, fill: pat)
```



```
tiling(
  size: auto array,
  spacing: array,
  offset: array,
  relative: auto str,
  content,
) → tiling
```

/3.1 **size** auto or array — Default: auto

The bounding box of each cell of the tiling, specified as a `(x, y)` pair.

If set to auto, the tiling takes on the size of the laid-out content.

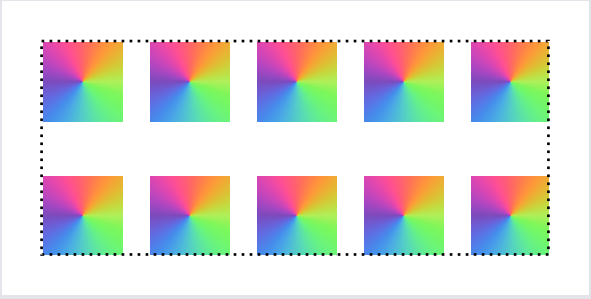
/3.2 **spacing** array — Default: `(0pt, 0pt)`

The spacing between cells of the tiling, specified as a `(x, y)` pair.

If the spacing is lower than the size of the tiling, the tiling will overlap with itself. If it is higher, the tiling will have gaps.

```
#let pat = tiling(
  size: (30pt, 30pt),
  spacing: (10pt, 20pt),
  square(size: 30pt, fill: gradient.conic(..color.map.rainbow)),
)

#rect(
  width: 100%,
  height: 80pt,
  fill: pat,
  stroke: (thickness: 1pt, dash: "dotted"),
)
```



/3.3 **offset** array — Default: `(0% + 0pt, 0% + 0pt)`

Shifts the entire tile grid without affecting the tile size or spacing.

The offset is specified as a `(x, y)` pair. Positive x values move the pattern to the right and positive y values move it down. Relative values are resolved against the tile size plus spacing.

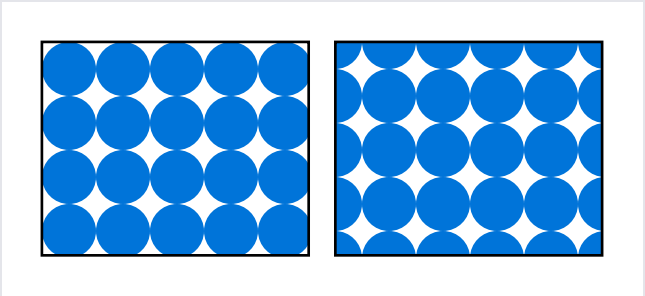
Note that the displacement caused by the offset affects the tiles themselves while displacement of the inner contents (e.g. via `place(dx: .., dy: ..)`) can cause clipping when the content moves outside of the tile's bounding box.

```
#set rect(width: 100%, height: 80pt, stroke: 1pt)

#let pat = tiling(
  size: (20pt, 20pt),
  circle(radius: 10pt, fill: blue),
)

#let pat-with-offset = tiling(
  size: (20pt, 20pt),
  offset: (50%, 50%),
  circle(radius: 10pt, fill: blue),
)

#grid(
  columns: 2,
  column-gutter: 10pt,
  rect(fill: pat),
  rect(fill: pat-with-offset),
)
```



/3.4 **relative** `auto` or `str` Default: `auto`

Determines relative to which element's bounding box the tiling is drawn.

By default, tilings are drawn relative to the shape they are being painted on (`"self"`), unless the tiling is applied on text, in which case they are relative to the closest ancestor container (`"parent"`).

The parent of an element is the innermost `box`[3.10.5] or `block`[3.10.4] that contains the element, or, if there is none, the page itself.

```
#let pat = tiling(
  size: (20pt, 20pt),
  spacing: (5pt, 5pt),
  relative: "self",
  circle(radius: 10pt, fill: teal),
)

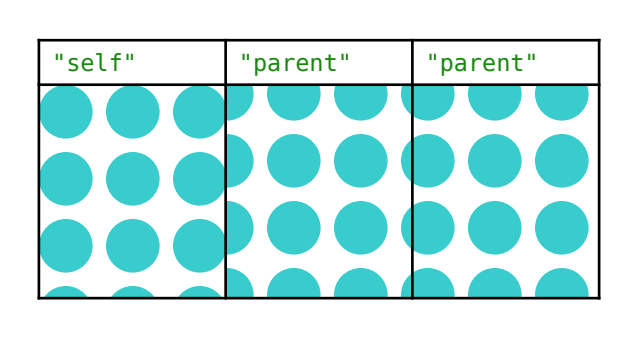
#let pat-with-parent = tiling(
  size: (20pt, 20pt),
  spacing: (5pt, 5pt),
  relative: "parent",
  circle(radius: 10pt, fill: teal),
)

#set raw(lang: "typc")
#table(
```

```
  columns: (1fr, 1fr, 1fr),
  rows: (auto, 80pt),
  table.header(`"self"`, `"parent"`, `"parent"`),

  // This one is local to the cell itself.
  table.cell(fill: pat, none),

  // These two are both page-relative, so the
  // pattern is continous.
  table.cell(fill: pat-with-parent, none),
  table.cell(fill: pat-with-parent, none),
)
```



Variant	Details
`"self"`	Relative to itself (its own bounding box).
`"parent"`	Relative to its parent (the parent's bounding box).

/3.5 **body** `content` *Required* *Positional*

The content of each cell of the tiling.

3.12 Introspection

Interactions between document parts.

This category is home to Typst's introspection capabilities: With the `counter` function, you can access and manipulate page, section, figure, and equation counters or create custom ones. Meanwhile, the `query` function lets you search for elements in the document to construct things like a list of figures or headers which show the current chapter title.

Most of the functions are *contextual*. It is recommended to read the chapter on context[3.4] before continuing here.

Definitions

3.12.1 Counter Counts through pages, elements, and more. . . . 455
3.12.2 Here Provides the current location in the document. . . . 460
3.12.3 Locate Determines the location of an element in the document. . . . 461
3.12.4 Location Identifies an element in the document. . . . 462
3.12.5 Metadata Exposes a value to the query system without producing visible content. . . . 464
3.12.6 Query Finds elements in the document. . . . 465
3.12.7 State Manages stateful parts of your document. . . . 468

3.12.1 `counter`

Counts through pages, elements, and more.

With the counter function, you can access and modify counters for pages, headings, figures, and more. Moreover, you can define custom counters for other things you want to count.

Since counters change throughout the course of the document, their current value is *contextual*. It is recommended to read the chapter on context[3.4] before continuing here.

/1 Accessing a counter

To access the raw value of a counter, we can use the `get`[3.12.1/11.1] function. This function returns an array[3.5.2]: Counters can have multiple levels (in the case of headings for sections, subsections, and so on), and each item in the array corresponds to one level.

```
#set heading(numbering: "1.")

= Introduction
Raw value of heading counter is
#context counter(heading).get()
```

1. Introduction

Raw value of heading counter is (1,)

/2 Displaying a counter

Often, we want to display the value of a counter in a more human-readable way. To do that, we can call the `display`[3.12.1/11.2] function on the counter. This function retrieves the current counter value and formats it either with a provided or with an automatically inferred numbering[3.6.13].

```
#set heading(numbering: "1.")

= Introduction
Some text here.

= Background
The current value is: #context {
  counter(heading).display()
}

Or in roman numerals: #context {
  counter(heading).display("I")
}
```

1. Introduction

Some text here.

2. Background

The current value is: 2.

Or in roman numerals: II

/3 Modifying a counter

To modify a counter, you can use the `step` and `update` methods:

- The `step` method increases the value of the counter by one. Because counters can have multiple levels , it optionally takes a `level` argument. If given, the counter steps at the given depth.
- The `update` method allows you to arbitrarily modify the counter. In its basic form, you give it an integer (or an array for multiple levels). For more flexibility, you can instead also give it a function that receives the current value and returns a new value.

The heading counter is stepped before the heading is displayed, so `Analysis` gets the number seven even though the counter is at six after the second update.

```
#set heading(numbering: "1.")

= Introduction
#counter(heading).step()

= Background
#counter(heading).update(3)
#counter(heading).update(n => n * 2)

= Analysis
Let's skip 7.1.
#counter(heading).step(level: 2)

== Analysis
Still at #context {
  counter(heading).display()
}
```

1. Introduction

3. Background

7. Analysis

Let's skip 7.1.

7.2. Analysis

Still at 7.2.

/4 Element counters

Above, there are various examples of using the `heading`[3.6.10] counter. Headings are just one kind of element that can be counted. In general, counters can count through any kind of *locatable* element[3.12.4/1].

Additionally, a counter can also count just those elements that match a specific `selector`[3.5.25]. For example, `counter(figure.where(kind: image))` counts figures containing images, but ignores other kinds of figures.

/5 Page counter

The page counter is special. It is automatically stepped at each pagebreak. But like other counters, you can also step it manually. For example, you could have Roman page numbers for your preface, then switch to Arabic page numbers for your main content and reset the page counter to one.

```
#set page(numbering: "(i)")

= Preface
The preface is numbered with
roman numerals.

#set page(numbering: "1 / 1")
#counter(page).update(1)

= Main text
Here, the counter is reset to one.
We also display both the current
page and total number of pages in
Arabic numbers.
```

Preface

The preface is numbered with roman numerals.

(i)

Main text

Here, the counter is reset to one. We also display both the current page and total number of pages in Arabic numbers.

1 / 1

/6 Custom counters

To define your own counter, call the `counter` function with a string as a key. This key identifies the counter globally.

```
#let mine = counter("mycounter")
#context mine.display() \
#mine.step()
#context mine.display() \
#mine.update(c => c * 3)
#context mine.display()
```

0
1
3

/7 How to step

When you define and use a custom counter, in general, you should first step the counter and then display it. This way, the stepping behaviour of a counter can depend on the element it is stepped for. If you were writing a counter for, let's say, theorems, your theorem's definition would thus first include the counter step and only then display the counter and the theorem's contents.

```
#let c = counter("theorem")
#let theorem(it) = block[
  #c.step()
  *Theorem #context c.display():*
  #it
]

#theorem[$1 = 1$]
#theorem[$2 < 3$]
```

Theorem 1: $1 = 1$

Theorem 2: $2 < 3$

The rationale behind this is best explained on the example of the heading counter: An update to the heading counter depends on the heading's level. By stepping directly before the heading, we can correctly step from `1` to `1.1` when encountering a level 2 heading. If we were to step after the heading, we wouldn't know what to step to.

Because counters should always be stepped before the elements they count, they always start at zero. This way, they are at one for the first display (which happens after the first step).

/8 Time travel

Counters can travel through time! You can find out the final value of the counter before it is reached and even determine what the value was at any particular location in the document.

```
#let mine = counter("mycounter")

= Values
#context [
  Value here: #mine.get() \
  At intro: #mine.at(<intro>) \
  Final value: #mine.final()
]

#mine.update(n => n + 3)

= Introduction <intro>
#lorem(10)

#mine.step()
#mine.step()
```

Values

Value here: (0,)
At intro: (3,)
Final value: (5,)

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

/9 Other kinds of state

The `counter` type is closely related to state[3.12.7] type. Read its documentation for more details on state management in Typst and why it doesn't just use normal variables for counters.

/10 Constructor

Create a new counter identified by a key.

```
counter( str  label  selector  location  function ) → counter
```

/10.1 key `str` or `label` or `selector` or `location` or `function` *Required* *Positional*

The key that identifies this counter globally.

- If it is a string, creates a custom counter that is only affected by manual updates,
- If it is the page[3.10.17] function, counts through pages,

- If it is a selector[3.5.25], counts through elements that match the selector. For example, you can
  - provide an element function to count elements of that type,
  - provide a where[3.5.15/8.2] selector to count a type of element with specific fields,
  - provide a <label>[3.5.17] to count elements with that label.

  Element and where[3.5.15/8.2] selector counter keys are only supported for locatable elements[3.12.4/1].

/11 Definitions

/11.1 get Contextual

Retrieves the value of the counter at the current location. Always returns an array of integers, even if the counter has just one number.

This is equivalent to `counter.at(here())`.

```
self.get() → int array
```

/11.2 display Contextual

Displays the value of the counter.

You can provide both a custom numbering and a custom location. Both default to `auto`, selecting sensible defaults (the numbering of the counted element and the current location, respectively).

Returns the formatted output.

```
self.display(
  auto str function,
  at: auto label selector location function,
  both: bool,
) → any
```

/11.2.1 numbering auto or str or function *Positional* Default: auto

A numbering pattern or a function[3.6.13], which specifies how to display the counter. If given a function, that function receives each number of the counter as a separate argument. If the amount of numbers varies, e.g. for the heading argument, you can use an argument sink[3.5.1].

If this is omitted or set to `auto`, displays the counter with the numbering style for the counted element or with the pattern `"1.1"` if no such style exists.

/11.2.2 at auto or label or selector or location or function Default: auto

The place at which the counter should be displayed.

If a selector is used, it must match exactly one element in the document. The most useful kinds of selectors for this are labels[3.5.17] and locations[3.12.4].

If this is omitted or set to `auto`, this displays the counter at the current location. This is equivalent to using here()[3.12.2].

The numbering will be executed with a context in which `here()` resolves to the provided location, so that numberings which involve further counters resolve correctly.

/11.2.3 both bool Default: false

If enabled, displays the current and final top-level count together. Both can be styled through a single numbering pattern. This is used by the page numbering property to display the current and total number of pages when a pattern like `"1 / 1"` is given.

/11.3 at Contextual

Retrieves the value of the counter at the given location. Always returns an array of integers, even if the counter has just one number.

The `selector` must match exactly one element in the document. The most useful kinds of selectors for this are labels[3.5.17] and locations[3.12.4].

```
self.at(label selector location function) → int array
```

/11.3.1 **selector** label or selector or location or function *Required* *Positional*

The place at which the counter's value should be retrieved.

/11.4 **final** Contextual

Retrieves the value of the counter at the end of the document. Always returns an array of integers, even if the counter has just one number.

```
self.final() → int array
```

/11.5 **step**

Increases the value of the counter by one.

The update will be in effect at the position where the returned content is inserted into the document. If you don't put the output into the document, nothing happens! This would be the case, for example, if you write `let _ = counter(page).step()`. Counter updates are always applied in layout order and in that case, Typst wouldn't know when to step the counter.

```
self.step(level: int) → content
```

/11.5.1 **level** int Default: 1

The depth at which to step the counter. Defaults to 1.

/11.6 **update**

Updates the value of the counter.

Just like with `step`, the update only occurs if you put the resulting content into the document.

```
self.update(int array function) → content
```

/11.6.1 **update** int or array or function *Required* *Positional*

If given an integer or array of integers, sets the counter to that value. If given a function, that function receives the previous counter value (with each number as a separate argument) and has to return the new value (integer or array).

3.12.2 `here` Contextual

Provides the current location in the document.

You can think of `here` as a low-level building block that directly extracts the current location from the active context[3.4]. Some other functions use it internally: For instance, `counter.get()` is equivalent to `counter.at(here())`.

Within show rules on locatable[3.12.4/1] elements, `here()` will match the location of the shown element.

If you want to display the current page number, refer to the documentation of the `counter`[3.12.1] type. While `here` can be used to determine the physical page number, typically you want the logical page number that may, for instance, have been reset after a preface.

/1 Examples

Determining the current position in the document in combination with the `position`[3.12.4/2.2] method:

```
#context [
  I am located at
  #here().position()
]
```

I am located at (page: 1, x: 15pt, y: 22.24pt)

Running a query[3.12.6] for elements before the current position:

```
= Introduction
= Background

There are
#context query(
  selector(heading).before(here())
).len()
headings before me.

= Conclusion
```

Introduction

Background

There are 2 headings before me.

Conclusion

Refer to the `selector`[3.5.25] type for more details on before/after selectors.

/2 Parameters

```
here() → location
```

3.12.3 `locate` Contextual

Determines the location of an element in the document.

Takes a selector that must match exactly one element and returns that element's `location`[3.12.4]. This location can, in particular, be used to retrieve the physical `page`[3.12.4/2.1] number and `position`[3.12.4/2.2] (page, x, y) for that element.

/1 Examples

Locating a specific element:

```
#context [
  Introduction is at: \
  #locate(<intro>).position()
]

= Introduction <intro>
```

Introduction is at:
(page: 1, x: 15pt, y: 55.87pt)

Introduction

/2 Parameters

```
locate( label  selector  location  function ) → location
```

/2.1 `selector` `label` or `selector` or `location` or `function` *Required* *Positional*

A selector that should match exactly one element. This element will be located.

Especially useful in combination with
– `here`[3.12.2] to locate the current context,
– a `location`[3.12.4] retrieved from some queried element via the `location()`[3.5.8/3.5] method on content.

3.12.4 `location`

Identifies an element in the document.

A location uniquely identifies an element in the document and lets you access its absolute position on the pages. You can retrieve the current location with the `here`[3.12.2] function and the location of a queried or shown element with the `location()`[3.5.8/3.5] method on content.

/1 Locatable elements

Elements that are automatically assigned a location are called *locatable* and can be found with queries[3.12.6]:

- In the Model category[3.6], the following elements are locatable: `asset`[3.6.1], `bibliography`[3.6.2], `cite`[3.6.4], `document`[3.6.6], `emph`[3.6.7], `enum`[3.6.12], `figure`[3.6.8], `figure.caption`[3.6.8/6.1], `footnote`[3.6.9], `footnote.entry`[3.6.9/4.1], `heading`[3.6.10], `link`[3.6.11], `list`[3.6.3], `outline`[3.6.14], `outline.entry`[3.6.14/5.1], `par`[3.6.15], `quote`[3.6.17], `ref`[3.6.18], `strong`[3.6.19], `table`[3.6.20], `terms`[3.6.21], and `title`[3.6.22]. Most of the elements in the *Model* category are locatable because semantic elements like headings and figures are often used with introspection.
- In the Text category[3.7], the `raw`[3.7.6] element and the decoration elements `underline`[3.7.13], `overline`[3.7.5], `strike`[3.7.9], and `highlight`[3.7.1] are locatable as these are also quite semantic in nature.
- In the Introspection category[3.12], the `metadata`[3.12.5] element is locatable as being queried for is its primary purpose.
- In the other categories, most elements are not locatable. Exceptions are `math.equation`[3.8.15], `image`[3.11.7], and `pdf.attach`[3.14.5].

To find out whether a specific element is locatable, you can try to `query`[3.12.6] for it.

Note that you can still observe elements that are not locatable in queries through other means, for instance, when they have a label attached to them.

/2 Definitions

/2.1 `page`

Returns the page number for this location.

Note that this does not return the value of the page counter[3.12.1] at this location, but the true page number (starting from one).

If you want to know the value of the page counter, use `counter(page).at(loc)` instead.

Can be used with `here`[3.12.2] to retrieve the physical page position of the current context:

```
#context [
  I am located on
  page #here().page()
]
```

I am located on page 1

```
self.page() → int
```

/2.2 `position`

Returns a dictionary with the page number and the x, y position for this location. The page number starts at one and the coordinates are measured from the top-left of the page.

If you only need the page number, use `page()` instead as it allows Typst to skip unnecessary work.

```
self.position() → dictionary
```

/2.3 page-numbering

Returns the page numbering pattern of the page at this location. This can be used when displaying the page counter in order to obtain the local numbering. This is useful if you are building custom indices or outlines.

If the page numbering is set to `none` at that location, this function returns `none`.

```
self.page-numbering() → none str function
```

3.12.5 `metadata` Element

Exposes a value to the query system without producing visible content.

This element can be retrieved with the `query`[3.12.6] function and from the command line with `typst query`[3.12.6/3]. Its purpose is to expose an arbitrary value to the introspection system. To identify a metadata value among others, you can attach a `label`[3.5.17] to it and query for that label.

The `metadata` element is especially useful for command line queries because it allows you to expose arbitrary values to the outside world.

```
// Put metadata somewhere.
#metadata("This is a note") <note>

// And find it from anywhere else.
#context {
  query(<note>).first().value
}
```

This is a note

/1 Parameters

```
metadata( any ) → content
```

/1.1 `value` `any` *Required* *Positional*

The value to embed into the document.

3.12.6 `query` Contextual

Finds elements in the document.

The `query` function lets you search your document for elements of a particular type or with a particular label. To use it, you first need to ensure that context[3.4] is available.

/1 Finding elements

In the example below, we manually create a table of contents instead of using the `outline`[3.6.14] function.

To do this, we first query for all headings in the document at level 1 and where `outlined` is true. Querying only for headings at level 1 ensures that, for the purpose of this example, sub-headings are not included in the table of contents. The `outlined` field is used to exclude the "Table of Contents" heading itself.

Note that we open a `context` to be able to use the `query` function.

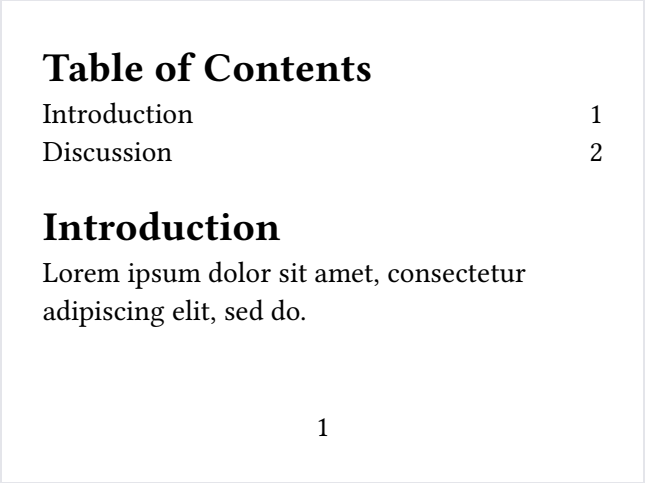
```
#set page(numbering: "1")

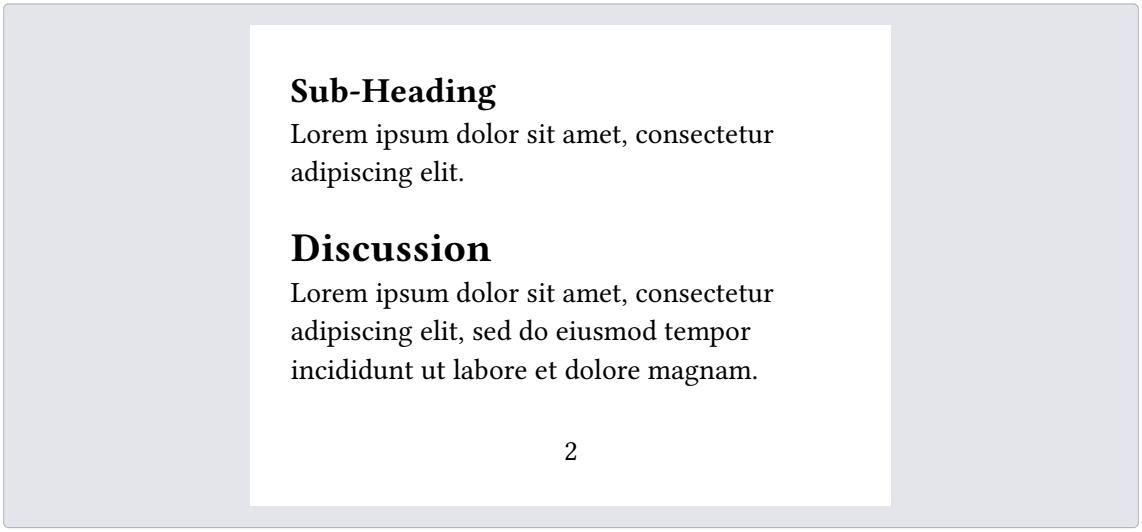
#heading(outlined: false)[
  Table of Contents
]
#context {
  let chapters = query(
    heading.where(
      level: 1,
      outlined: true,
    )
  )
  for chapter in chapters {
    let loc = chapter.location()
    let nr = counter(page).display(at: loc)
    [#chapter.body #h(1fr) #nr \ ]
  }
}

= Introduction
#lorem(10)
#pagebreak()

== Sub-Heading
#lorem(8)

= Discussion
#lorem(18)
```





To get the page numbers, we first get the location of the elements returned by `query` with `location`[3.5.8/3.5]. We then also retrieve the page numbering[3.12.4/2.3] and page counter[3.12.1/5] at that location and apply the numbering to the counter.

/2 ## A word of caution

To resolve all your queries, Typst evaluates and layouts parts of the document multiple times. However, there is no guarantee that your queries can actually be completely resolved. If you aren't careful a query can affect itself—leading to a result that never stabilizes.

In the example below, we query for all headings in the document. We then generate as many headings. In the beginning, there's just one heading, titled `Real`. Thus, `count` is `1` and one `Fake` heading is generated. Typst sees that the query's result has changed and processes it again. This time, `count` is `2` and two `Fake` headings are generated. This goes on and on. As we can see, the output has a finite amount of headings. This is because Typst simply gives up after a few attempts.

In general, you should try not to write queries that affect themselves. The same words of caution also apply to other introspection features like counters[3.12.1] and state[3.12.7].

```
= Real
#context {
  let elems = query(heading)
  let count = elems.len()
  count * [= Fake]
}
```



/3 ## Command line queries

You can also perform queries from the command line, using the `typst eval` command. This command evaluates Typst code, potentially in the context of a document, and outputs the resulting value in serialized form. It takes the code to evaluate as its first argument and (optionally) the path to a document via `--in`.

Consider the following `example.typ` file which contains some invisible metadata[3.12.5]:

```
#metadata("This is a note") <note>
```

You can execute a query on it as follows using Typst's CLI.

```
$ typst eval 'query(<note>)' --in example.typ
[
  {
    "func": "metadata",
    "value": "This is a note",
    "label": "<note>"
  }
]
```

This command tells Typst to compile `example.typ` and then run the code `query(<note>)` with access to the resulting document.

Note: The code is surrounded with quotes to avoid special characters being interpreted by the shell. How to quote strings depends on your platform/shell.

/3.1 Retrieving a specific field

Frequently, you're interested in only one specific field of the resulting elements. In the case of the `metadata` element, the `value` field is the interesting one. You can extract just this field by adjusting the code.

```
$ typst eval 'query(<note>).map(it => it.value)' --in example.typ
["This is a note"]
```

If you are interested in just a single element, you can also use the `first()`[3.5.2/3.2] method to extract just it.

```
$ typst eval 'query(<note>).first().value' --in example.typ
"This is a note"
```

/3.2 Querying for a specific export target

In case you need to query a document when exporting for a specific target, you can use the `--target` argument. Valid values are `paged`, and `html` (if the `html`[3.15] feature is enabled).

/4 Parameters

```
query( label  selector  location  function ) → array
```

/4.1 target `label` or `selector` or `location` or `function` *Required* *Positional*

Can be
- an element function like a `heading` or `figure`,
- a `<label>`,
- a more complex selector like `heading.where(level: 1)`,
- or `selector(heading).before(here())`.

Only locatable[3.12.4/1] element functions are supported.

3.12.7 `state`

Manages stateful parts of your document.

Let's say you have some computations in your document and want to remember the result of your last computation to use it in the next one. You might try something similar to the code below and expect it to output 10, 13, 26, and 21. However this **does not work** in Typst. If you test this code, you will see that Typst complains with the following error message: *Variables from outside the function are read-only and cannot be modified.*

```
// This doesn't work!
#let star = 0
#let compute(expr) = {
  star = eval(
    expr.replace("⭐", str(star))
  )
  [New value is #star.]
}

#compute("10") \
#compute("⭐ + 3") \
#compute("⭐ * 2") \
#compute("⭐ - 5")
```

/1 State and document markup

Why does it do that? Because, in general, this kind of computation with side effects is problematic in document markup and Typst is upfront about that. For the results to make sense, the computation must proceed in the same order in which the results will be laid out in the document. In our simple example, that's the case, but in general it might not be.

Let's look at a slightly different, but similar kind of state: The heading numbering. We want to increase the heading counter at each heading. Easy enough, right? Just add one. Well, it's not that simple. Consider the following example:

```
#set heading(numbering: "1.")
#let template(body) = [
  = Outline
  ...
  #body
]

#show: template

= Introduction
...
```

1. Outline

...

2. Introduction

...

Here, Typst first processes the body of the document after the show rule, sees the `Introduction` heading, then passes the resulting content to the `template` function and only then sees the `Outline`. Just counting up would number the `Introduction` with 1 and the `Outline` with 2.

/2 Managing state in Typst

So what do we do instead? We use Typst's state management system. Calling the `state` function with an identifying string key and an optional initial value gives you a state value which exposes a few functions. The two most important ones are `get` and `update`:

- The `get`[3.12.7/6.1] function retrieves the current value of the state. Because the value can vary over the course of the document, it is a *contextual* function that can only be used when context[3.4] is available.
- The `update`[3.12.7/6.4] function modifies the state. You can give it any value. If given a non-function value, it sets the state to that value. If given a function, that function receives the previous state and has to return the new state.

Our initial example would now look like this:

```
#let star = state("star", 0)
#let compute(expr) = {
  star.update(old =>
    eval(expr.replace("⭐", str(old)))
  )
  [New value is #context star.get().]
}

#compute("10") \
#compute("⭐ + 3") \
#compute("⭐ * 2") \
#compute("⭐ - 5")
```

New value is 10.
New value is 13.
New value is 26.
New value is 21.

State managed by Typst is always updated in layout order, not in evaluation order. The `update` method returns content and its effect occurs at the position where the returned content is inserted into the document.

As a result, we can now also store some of the computations in variables, but they still show the correct results:

```
...

#let more = [
  #compute("⭐ * 2") \
  #compute("⭐ - 5")
]

#compute("10") \
#compute("⭐ + 3") \
#more
```

New value is 10.
New value is 13.
New value is 26.
New value is 21.

This example is of course a bit silly, but in practice this is often exactly what you want! A good example are heading counters, which is why Typst's counting system[3.12.1] is very similar to its state system.

/3 Time Travel

By using Typst's state management system you also get time travel capabilities! We can find out what the value of the state will be at any position in the document from anywhere else. In particular, the `at` method gives us the value of the state at any particular location and the `final` methods gives us the value of the state at the end of the document.

```
...

Value at `<here>` is
#context star.at(<here>)

#compute("10") \
#compute("⭐ + 3") \
*Here.* <here> \
#compute("⭐ * 2") \
#compute("⭐ - 5")
```

Value at <here> is 13

New value is 10.
New value is 13.
Here.
New value is 26.
New value is 21.

/4 A word of caution

To resolve the values of all states, Typst evaluates parts of your code multiple times. However, there is no guarantee that your state manipulation can actually be completely resolved.

For instance, if you generate state updates depending on the final value of a state, the results might never converge. The example below illustrates this. We initialize our state with `1` and then update it to its own

final value plus 1. So it should be 2, but then its final value is 2, so it should be 3, and so on. This example displays a finite value because Typst simply gives up after a few attempts.

```
// This is bad!
#let x = state("key", 1)
#context x.update(x.final() + 1)
#context x.get()
```

5

In general, you should try not to generate state updates from within context expressions. If possible, try to express your updates as non-contextual values or functions that compute the new value from the previous value. Sometimes, it cannot be helped, but in those cases it is up to you to ensure that the result converges.

/5 Constructor

Create a new state identified by a key.

```
state(
  str,
  any,
) → state
```

/5.1 key `str` *Required* *Positional*

The key that identifies this state.

Any updates[3.12.7/6.4] to the state will be identified with the string key. If you construct multiple states with the same `key`, then updating any one will affect all of them.

/5.2 init `any` *Positional* Default: none

The initial value of the state.

If you construct multiple states with the same `key` but different `init` values, they will each use their own initial value but share updates. Specifically, the value of a state at some location in the document will be computed from that state's initial value and all preceding updates for the state's key.

```
#let banana = state("key", "🍌")
#let broccoli = state("key", "🥦")

#banana.update(it => it + "😋")

#context [
  - #state("key", "🍎").get()
  - #banana.get()
  - #broccoli.get()
]
```

- 🍎😋
- 🍌😋
- 🥦😋

/6 Definitions

/6.1 get Contextual

Retrieves the value of the state at the current location.

This is equivalent to `state.at(here())`.

```
self.get() → any
```

/6.2 at Contextual

Retrieves the value of the state at the given selector's unique match.

The `selector` must match exactly one element in the document. The most useful kinds of selectors for this are labels[3.5.17] and locations[3.12.4].

```
self.at( label selector location function ) → any
```

/6.2.1 **selector** `label` or `selector` or `location` or `function` *Required* *Positional*

The place at which the state's value should be retrieved.

/6.3 **final** Contextual

Retrieves the value of the state at the end of the document.

```
self.final() → any
```

/6.4 **update**

Updates the value of the state.

Returns an invisible piece of content[3.5.8] that must be inserted into the document to take effect. This invisible content tells Typst that the specified update should take place wherever the content is inserted into the document.

State is a part of your document and runs like a thread embedded in the document content. The value of a state is the result of all state updates that happened in the document up until that point.

That's why `state.update` returns an invisible sliver of content that you need to return and include in the document — a state update that is not "placed" in the document does not happen, and "when" it happens is determined by where you place it. That's also why you need context[3.4] to read state: You need to use the current document position to know where on the state's "thread" you are.

Storing a state update in a variable (e.g. `let my-update = state("key").update(c => c * 2)`) will have no effect by itself. Only once you insert the variable `#my-update` somewhere into the document content, the update will take effect — at the position where it was inserted. You can also use `#my-update` multiple times at different positions. Then, the update will take effect multiple times as well.

In contrast to `get`[3.12.7/6.1], `at`[3.12.7/6.2], and `final`[3.12.7/6.3], this function does not require context[3.4]. This is because, to create the state update, we do not need to know where in the document we are. We only need this information to resolve the state's value.

```
self.update( function any ) → content
```

/6.4.1 **update** `function` or `any` *Required* *Positional*

A value to update to or a function to update with.

- If given a non-function value, sets the state to that value.
- If given a function, that function receives the state's previous value and has to return the state's new value.

When updating the state based on its previous value, you should prefer the function form instead of retrieving the previous value from the context[3.4]. This allows the compiler to resolve the final state efficiently, minimizing the number of layout iterations[3.4.4] required.

In the following example, `fill.update(f => not f)` will paint odd items in the bullet list[3.6.3/4.1] as expected. However, if it's replaced with `context fill.update(not fill.get())`, then layout will not converge within 5 attempts, as each update will take one additional iteration to propagate.

```
#let fill = state("fill", false)

#show list.item: it => {
  fill.update(f => not f)
  context {
    set text(fill: fuchsia) if fill.get()
    it
  }
}
```

```
#lorem(5).split().map(list.item).join()
```

- Lorem
- ipsum
- dolor
- sit
- amet.

3.13 Data Loading

Data loading from external files.

These functions help you with loading and embedding data, for example from the results of an experiment.

3.13.1 Encoding

Some of the functions are also capable of encoding, e.g. `cbor.encode`[3.13.2/3.1]. They facilitate passing structured data to plugins[3.5.22].

However, each data format has its own native types. Therefore, for an arbitrary Typst value, the encode-to-decode roundtrip might be lossy. In general, numbers, strings, and arrays[3.5.2] or dictionaries[3.5.11] composed of them can be reliably converted, while other types may fall back to strings via `repr`[3.5.24], which is for debugging purposes only[3.5.24/2]. Please refer to the page of each data format for details.

Definitions

3.13.2 CBOR Reads structured data from a CBOR file. . . . 474
3.13.3 CSV Reads structured data from a CSV file. . . . 475
3.13.4 JSON Reads structured data from a JSON file. . . . 476
3.13.5 Read Reads plain text or data from a file. . . . 478
3.13.6 TOML Reads structured data from a TOML file. . . . 479
3.13.7 XML Reads structured data from an XML file. . . . 481
3.13.8 YAML Reads structured data from a YAML file. . . . 483

3.13.2 cbor

Reads structured data from a CBOR file.

The file must contain a valid CBOR serialization. The CBOR values will be converted into corresponding Typst values as listed in the table below[3.13.2/1].

The function returns a dictionary, an array or, depending on the CBOR file, another CBOR data type.

/1 Conversion details

CBOR value	Converted into Typst
integer	`int`[3.5.16] (or `float`[3.5.14])
bytes	`bytes`[3.5.6]
float	`float`[3.5.14]
text	`str`[3.5.27]
bool	`bool`[3.5.5]
null	`none`
array	`array`[3.5.2]
map	`dictionary`[3.5.11]

Typst value	Converted into CBOR
types that can be converted from CBOR	corresponding CBOR value
`symbol`[3.5.28]	text
`content`[3.5.8]	a map describing the content
other types (`length`[3.10.13], etc.)	text via `repr`[3.5.24]

/1.1 Notes

- Be aware that CBOR integers larger than $2^{63}-1$ or smaller than -2^{63} will be converted to floating point numbers, which may result in an approximative value.
- CBOR tags are not supported, and an error will be thrown.
- The `repr` function is for debugging purposes only[3.5.24/2], and its output is not guaranteed to be stable across Typst versions.

/2 Parameters

```
cbor( str  path  bytes ) → any
```

/2.1 **source** `str` or `path` or `bytes` *Required* *Positional*

A path to a CBOR file or raw CBOR bytes.

/3 Definitions

/3.1 encode

Encode structured data into CBOR bytes.

```
cbor.encode( any ) → bytes
```

/3.1.1 **value** `any` *Required* *Positional*

Value to be encoded.

3.13.3 csv

Reads structured data from a CSV file.

The CSV file will be read and parsed into a 2-dimensional array of strings: Each row in the CSV file will be represented as an array of strings, and all rows will be collected into a single array. Header rows will not be stripped.

/1 Example

```
#let results = csv("example.csv")

#table(
  columns: 2,
  [*Condition*], [*Result*],
  ..results.flatten(),
)
```

Condition	Result
0..2	small
3..5	medium
6..	big

/2 Parameters

```
csv(
  str path bytes,
  delimiter: str,
  row-type: type,
) → array
```

/2.1 **source** `str` or `path` or `bytes` *Required* *Positional*

A path to a CSV file or raw CSV bytes.

/2.2 **delimiter** `str` Default: `","`

The delimiter that separates columns in the CSV file. Must be a single ASCII character.

/2.3 **row-type** `type` Default: `array`

How to represent the file's rows.

- If set to `array`, each row is represented as a plain array of strings.
- If set to `dictionary`, each row is represented as a dictionary mapping from header keys to strings. This option only makes sense when a header row is present in the CSV file.

3.13.4 json

Reads structured data from a JSON file.

The file must contain a valid JSON value, such as object or array. The JSON values will be converted into corresponding Typst values as listed in the table below[3.13.4/2].

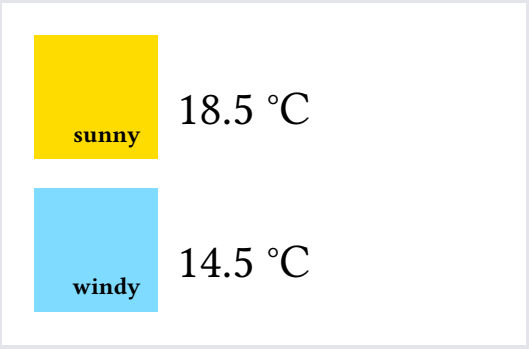
The function returns a dictionary, an array or, depending on the JSON file, another JSON data type.

The JSON files in the example contain objects with the keys `temperature`, `unit`, and `weather`.

/1 Example

```
#let forecast(day) = block[
  #box(square(
    width: 2cm,
    inset: 8pt,
    fill: if day.weather == "sunny" {
      yellow
    } else {
      aqua
    },
    align(
      bottom + right,
      strong(day.weather),
    ),
  ))
  #h(6pt)
  #set text(22pt, baseline: -8pt)
  #day.temperature °#day.unit
]

#forecast(json("monday.json"))
#forecast(json("tuesday.json"))
```



/2 Conversion details

JSON value	Converted into Typst
`null`	`none`
bool	`bool`[3.5.5]
number	`float`[3.5.14] or `int`[3.5.16]
string	`str`[3.5.27]
array	`array`[3.5.2]
object	`dictionary`[3.5.11]

Typst value	Converted into JSON
types that can be converted from JSON	corresponding JSON value
`bytes`[3.5.6]	string via `repr`[3.5.24]
`symbol`[3.5.28]	string
`content`[3.5.8]	an object describing the content
other types (`length`[3.10.13], etc.)	string via `repr`[3.5.24]

/2.1 Notes

– In most cases, JSON numbers will be converted to floats or integers depending on whether they are whole numbers. However, be aware that integers larger than $2^{63}-1$ or smaller than -2^{63} will be converted to floating-point numbers, which may result in an approximative value.

– Bytes are not encoded as JSON arrays for performance and readability reasons. Consider using `cbor.encode`[3.13.2/3.1] for binary data.
– The `repr` function is for debugging purposes only[3.5.24/2], and its output is not guaranteed to be stable across Typst versions.

/3 Parameters

```
json( str  path  bytes ) → any
```

/3.1 source `str` or `path` or `bytes` *Required* *Positional*

A path to a JSON file or raw JSON bytes.

/4 Definitions

/4.1 encode

Encodes structured data into a JSON string.

```
json.encode(
  any ,
  pretty: bool ,
) → str
```

/4.1.1 value `any` *Required* *Positional*

Value to be encoded.

/4.1.2 pretty `bool` Default: `true`

Whether to pretty print the JSON with newlines and indentation.

3.13.5 `read`

Reads plain text or data from a file.

By default, the file will be read as UTF-8 and returned as a string[3.5.27].

If you specify `encoding: none`, this returns raw bytes[3.5.6] instead.

/1 Example

```
An example for a HTML file: \
#let text = read("example.html")
#raw(text, block: true, lang: "html")

Raw bytes:
#read("tiger.jpg", encoding: none)
```

An example for a HTML file:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Example document</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
  </body>
</html>
```

Raw bytes: `bytes(116679)`

/2 Parameters

```
read(
  str path,
  encoding: none str,
) → str bytes
```

/2.1 **path** `str` or `path` *Required* *Positional*

Path to a file.

/2.2 **encoding** `none` or `str` Default: `"utf8"`

The encoding to read the file with.

If set to `none`, this function returns raw bytes.

Variant	Details
`"utf8"`	The Unicode UTF-8 encoding.

3.13.6 `toml`

Reads structured data from a TOML file.

The file must contain a valid TOML table. The TOML values will be converted into corresponding Typst values as listed in the table below[3.13.6/2].

The function returns a dictionary representing the TOML table.

The TOML file in the example consists of a table with the keys `title`, `version`, and `authors`.

/1 Example

```
#let details = toml("details.toml")

Title: #details.title \
Version: #details.version \
Authors: #(details.authors
  .join(", ", last: " and "))
```

Title: Secret project
Version: 2
Authors: Mr Robert and Miss Enola

/2 Conversion details

First of all, TOML documents are tables. Other values must be put in a table to be encoded or decoded.

TOML value	Converted into Typst
string	`str`[3.5.27]
integer	`int`[3.5.16]
float	`float`[3.5.14]
boolean	`bool`[3.5.5]
datetime	`datetime`[3.5.9]
array	`array`[3.5.2]
table	`dictionary`[3.5.11]

Typst value	Converted into TOML
types that can be converted from TOML	corresponding TOML value
`none`	ignored
`bytes`[3.5.6]	string via `repr`[3.5.24]
`symbol`[3.5.28]	string
`content`[3.5.8]	a table describing the content
other types (`length`[3.10.13], etc.)	string via `repr`[3.5.24]

/2.1 Notes

- Be aware that TOML integers larger than $2^{63}-1$ or smaller than -2^{63} cannot be represented losslessly in Typst, and an error will be thrown according to the specification[A].
- Bytes are not encoded as TOML arrays for performance and readability reasons. Consider using `cbor.encode`[3.13.2/3.1] for binary data.
- The `repr` function is for debugging purposes only[3.5.24/2], and its output is not guaranteed to be stable across Typst versions.

/3 Parameters

```
toml( str  path  bytes ) → dictionary
```

[A] https://toml.io/en/v1.0.0#integer

/3.1 **source** `str` or `path` or `bytes` *Required* *Positional*

A path to a TOML file or raw TOML bytes.

/4 Definitions

/4.1 encode

Encodes structured data into a TOML string.

```
toml.encode(
  dictionary,
  pretty: bool,
) → str
```

/4.1.1 **value** `dictionary` *Required* *Positional*

Value to be encoded.

TOML documents are tables. Therefore, only dictionaries are suitable.

/4.1.2 **pretty** `bool` Default: `true`

Whether to pretty-print the resulting TOML.

3.13.7 `xml`

Reads structured data from an XML file.

The XML file is parsed into an array of dictionaries and strings. XML nodes can be elements or strings. Elements are represented as dictionaries with the following keys:

- `tag`: The name of the element as a string.
- `attrs`: A dictionary of the element's attributes as strings.
- `children`: An array of the element's child nodes.

The XML file in the example contains a root `news` tag with multiple `article` tags. Each article has a `title`, `author`, and `content` tag. The `content` tag contains one or more paragraphs, which are represented as `p` tags.

/1 Example

```
#let find-child(elem, tag) = {
  elem.children
    .find(e => "tag" in e and e.tag == tag)
}

#let article(elem) = {
  let title = find-child(elem, "title")
  let author = find-child(elem, "author")
  let pars = find-child(elem, "content")

  [= #title.children.first()]
  text(10pt, weight: "medium")[
    Published by
    #author.children.first()
  ]

  for p in pars.children {
    if type(p) == dictionary {
      parbreak()
      p.children.first()
    }
  }
}

#let data = xml("example.xml")
#for elem in data.first().children {
  if type(elem) == dictionary {
    article(elem)
  }
}
```



/2 **Parameters**

```
xml(str path bytes) → any
```

/2.1 **source** `str` or `path` or `bytes` *Required* *Positional*

A path to an XML file or raw XML bytes.

3.13.8 yaml

Reads structured data from a YAML file.

The file must contain a valid YAML object or array. The YAML values will be converted into corresponding Typst values as listed in the table below[3.13.8/2].

The function returns a dictionary, an array or, depending on the YAML file, another YAML data type.

The YAML files in the example contain objects with authors as keys, each with a sequence of their own submapping with the keys “title” and “published”.

/1 Example

```
#let bookshelf(contents) = {
  for (author, works) in contents {
    author
    for work in works [
      - #work.title (#work.published)
    ]
  }
}

#bookshelf(
  yaml("scifi-authors.yaml")
)
```

Arthur C. Clarke
- Against the Fall of Night (1978)
- The songs of distant earth (1986)

Isaac Asimov
- Quasar, Quasar, Burning Bright (1977)
- Far as Human Eye Could See (1987)

/2 Conversion details

YAML value	Converted into Typst
null-values (`null`, `~` or empty)	`none`
boolean	`bool`[3.5.5]
number	`float`[3.5.14] or `int`[3.5.16]
string	`str`[3.5.27]
sequence	`array`[3.5.2]
mapping	`dictionary`[3.5.11]

Typst value	Converted into YAML
types that can be converted from YAML	corresponding YAML value
`bytes`[3.5.6]	string via `repr`[3.5.24]
`symbol`[3.5.28]	string
`content`[3.5.8]	a mapping describing the content
other types (`length`[3.10.13], etc.)	string via `repr`[3.5.24]

/2.1 Notes

- In most cases, YAML numbers will be converted to floats or integers depending on whether they are whole numbers. However, be aware that integers larger than $2^{63}-1$ or smaller than -2^{63} will be converted to floating-point numbers, which may result in an approximative value.
- Custom YAML tags are ignored, though the loaded value will still be present.
- Bytes are not encoded as YAML sequences for performance and readability reasons. Consider using `cbor.encode`[3.13.2/3.1] for binary data.
- The `repr` function is for debugging purposes only[3.5.24/2], and its output is not guaranteed to be stable across Typst versions.

/3 ## Parameters

```
yaml(str path bytes) → any
```

/3.1 **source** `str` or `path` or `bytes` *Required* *Positional*

A path to a YAML file or raw YAML bytes.

/4 ## Definitions

/4.1 **encode**

Encode structured data into a YAML string.

```
yaml.encode(any) → str
```

/4.1.1 **value** `any` *Required* *Positional*

Value to be encoded.

3.14 PDF

PDF files focus on accurately describing documents visually, but also have facilities for annotating their structure. This hybrid approach makes them a good fit for document exchange: They render exactly the same on every device, but also support extraction of a document's content and structure (at least to an extent). Unlike PNG files, PDFs are not bound to a specific resolution. Hence, you can view them at any size without incurring a loss of quality.

3.14.1 Exporting as PDF

/1 Command Line

PDF is Typst's default export format. Running the `compile` or `watch` subcommand without specifying a format will create a PDF. When exporting to PDF, you have the following configuration options:

- Which PDF standards[3.14.2] Typst should enforce conformance with by specifying `--pdf-standard` followed by one or multiple comma-separated standards. Valid standards are `1.4`, `1.5`, `1.6`, `1.7`, `2.0`, `a-1b`, `a-1a`, `a-2b`, `a-2u`, `a-2a`, `a-3b`, `a-3u`, `a-3a`, `a-4`, `a-4f`, `a-4e`, and `ua-1`. By default, Typst outputs PDF-1.7-compliant files.
- You can disable PDF tagging completely with `--no-pdf-tags`. By default, Typst will always write *Tagged PDF* to provide a baseline level of accessibility. Using this flag, you can turn tags off. This will make your file inaccessible and prevent conformance with accessible conformance levels of PDF/A and all parts of PDF/UA.
- Which pages to export by specifying `--pages` followed by a comma-separated list of numbers or dash-separated number ranges. Ranges can be half-open. Example: `2,3,7-9,11-`.

/2 Web App

Click the quick download button at the top right to export a PDF with default settings. For further configuration, click "File" > "Export as" > "PDF" or click the downwards-facing arrow next to the quick download button and select "Export as PDF". When exporting to PDF, you have the following configuration options:

- Which PDF standards Typst should enforce conformance with. By default, Typst outputs PDF-1.7-compliant files. You can choose the PDF version freely between 1.4 and 2.0. Valid additional standards are `A-1b`, `A-1a`, `A-2b`, `A-2u`, `A-2a`, `A-3b`. `A-3u`, `A-3a`, `A-4`, `A-4f`, `A-4e`, and `UA-1`.
- Which pages to export. Valid options are "All pages", "Current page", and "Custom ranges". Custom ranges are a comma-separated list of numbers or dash-separated number ranges. Ranges can be half-open. Example: `2,3,7-9,11-`.

3.14.2 PDF standards

The International Standards Organization (ISO) has published the base PDF standard and various standards that extend it to make PDFs more suitable for specific use-cases. By default, Typst exports PDF 1.7 files. Adobe Acrobat 8 and later as well as all other commonly used PDF viewers are compatible with this PDF version.

Some features of Typst may not be available depending on the PDF standard you choose. You can combine compatible PDF/A and PDF/UA standards. Notably, PDF/UA-1 and PDF/A-4 are mutually incompatible since the former requires PDF 1.7 or earlier and the latter requires PDF 2.0.

/1 PDF versions

Typst supports five different PDF versions: 1.4, 1.5, 1.6, 1.7 (default), and 2.0. You can choose each of these versions for your document export. However, based on the features you used there may be a minimum version. Likewise, the standards you target can limit which versions you can choose (see below for more).

Here is a list on how each new version improves over PDF 1.4 for Typst documents:

- **PDF 1.5** (2003): Improved color management, text extraction, table accessibility, reflow, and emoji fonts
- **PDF 1.6** (2004): More flexible links
- **PDF 1.7** (2006): Allows attachments[3.14.5], improved reflow

– **PDF 2.0** (2017): Improved both metadata and tag semantics for accessibility

The software used to read your file must support your PDF version. Under normal circumstances, this poses no problem, but it can be a source of errors when working with older hardware. For general exchange, we recommend keeping the default PDF 1.7 setting or choosing PDF 2.0.

When using PDF files as images[3.11.7] in your document, the export PDF version must equal or exceed the image file versions.

/2 PDF/UA

Typst supports writing PDF/UA-conformant files. PDF/UA files are designed for *Universal Access*[4.4.1]. When you choose this PDF standard, Typst will run additional checks when exporting your document. These checks will make sure that you are following accessibility best practices. For example, it will make sure that all your images come with alternative descriptions.

Note that there are some rules in PDF/UA that are crucial for accessibility but cannot be automatically checked. Hence, when exporting a PDF/UA-1 document, make sure you did the following:

– If your document is written in a different language than English, make sure set the text language[3.7.12/2.16] before any content.
– Make sure to use Typst's semantic elements (like headings[3.6.10], figures[3.6.8], and lists[3.6.3]) when appropriate instead of defining custom constructs. This lets Typst know (and export) the role a construct plays in the document. See the Accessibility guide[4.4] for more details.
– Do not exclusively use contrast, color, format, or layout to communicate an idea. Use text or alternative descriptions instead of or in addition to these elements.
– Wrap all decorative elements without a semantic meaning in `pdf.artifact`[3.14.4].
– Do not use images of text. Instead, insert the text directly into your markup.

Typst currently only supports part one (PDF/UA-1) which is based on PDF 1.7 (2006). When exporting to PDF/UA-1, be aware that you will need to manually provide alternative descriptions of mathematics[3.8.8] in natural language.

New accessibility features were added to PDF 2.0 (2017). When set to PDF 2.0 export, Typst will leverage some of these features. PDF 2.0 and PDF/UA-1, however, are mutually incompatible. For accessible documents, we currently recommend exporting to PDF/UA-1 instead of PDF 2.0 for the additional checks and greater compatibility. The second part of PDF/UA is designed for PDF 2.0, but not yet supported by Typst.

/3 PDF/A

Typst optionally supports emitting PDF/A-conformant files. PDF/A files are geared towards maximum compatibility with current and future PDF tooling. They do not rely on difficult-to-implement or proprietary features and contain exhaustive metadata. This makes them suitable for long-term archival.

The PDF/A Standard has multiple versions (*parts* in ISO terminology) and most parts have multiple profiles that indicate the file's conformance level. You can target one part and conformance level at a time. Currently, Typst supports these PDF/A output profiles:

– **PDF/A-1b:** The *basic* conformance level of ISO 19005-1. This version of PDF/A is based on PDF 1.4 (2001) and results in self-contained, archivable PDF files. As opposed to later parts of the PDF/A standard, transparency is not allowed in PDF/A-1 files.

– **PDF/A-1a:** This is the *accessible* conformance level that builds on the basic level PDF/A-1b. To conform to this level, your file must be an accessible *Tagged PDF* file. Note that accessibility is improved with later parts as not all PDF accessibility features are available for PDF 1.4 files. Furthermore, all text in the file must consist of known Unicode code points.

– **PDF/A-2b:** The *basic* conformance level of ISO 19005-2. This version of PDF/A is based on PDF 1.7 (2006) and results in self-contained, archivable PDF files.

– **PDF/A-2u:** This is the *Unicode-mappable* conformance level that builds on the basic level A-2b. It also adds rules that all text in the document must consist of known Unicode code points. If possible, always prefer this standard over PDF/A-2b.

– **PDF/A-2a:** This is the *accessible* conformance level that builds on the Unicode-mappable level A-2u. This conformance level also adds two requirements: Your file must be an accessible *Tagged PDF* file. Typst

automatically adds tags to help you reach this conformance level. Also pay attention to the Accessibility Sections throughout the reference and the Accessibility Guide[4.4] when targeting this conformance level. Finally, PDF/A-2a forbids you from using code points in the Unicode Private Use area[A]. If you want to build an accessible file, also consider additionally targeting PDF/UA-1, which enables more automatic accessibility checks.

- **PDF/A-3b:** The *basic* conformance level of ISO 19005-3. This version of PDF/A is based on PDF 1.7 (2006) and results in archivable PDF files that can contain arbitrary other related files as attachments[3.14.5]. The only difference between it and PDF/A-2b is the capability to attach non-PDF/A-conformant files.
- **PDF/A-3u:** This is the *Unicode-mappable* conformance level that builds on the basic level A-3b. Just like PDF/A-2b, this requires all text to consist of known Unicode code points. These rules do not apply to attachments. If possible, always prefer this standard over PDF/A-3b.
- **PDF/A-3a:** This is the *accessible* conformance level that builds on the Unicode-mappable level A-3u. Just like PDF/A-2a, this requires files to be accessible *Tagged PDF* and to not use characters from the Unicode Private Use area. Just like before, these rules do not apply to attachments.
- **PDF/A-4:** The basic conformance level of ISO 19005-4. This version of PDF/A is based on PDF 2.0 (2017) and results in self-contained, archivable PDF files. PDF/A-4 has no parts relating to accessibility. Instead, the topic has been elaborated on more in the dedicated PDF/UA standard. PDF/A-4 files can conform to PDF/UA-2 (currently not supported in Typst).
- **PDF/A-4f:** The *embedded files* conformance level that builds on the basic level A-4. Files conforming to this level can contain arbitrary other related files as attachments[3.14.5], just as files conforming to part 3 of ISO 19005. The only difference between it and PDF/A-4 is the capability to attach non-PDF/A-conformant files.
- **PDF/A-4e:** The *engineering* conformance level that builds on the embedded files level A-4f. Files conforming to this level can contain 3D objects. Typst does not support 3D content, so this is functionally equivalent to PDF/A-4f from a Typst perspective.

If you want to target PDF/A but are unsure about which particular setting to use, there are some good rules of thumb. First, you must determine your **part,** that's the "version number" of the standard. Ask yourself these questions:

1. **Does pre-2006 software or equipment need to be able to read my file?** If so, choose part one (PDF/A-1).
2. **If not, does my file need attachments[3.14.5]?** If so, you must choose part three (PDF/A-3) or the embedded files level of part four (PDF/A-4f).
3. **Does your file need to use features introduced in PDF 2.0?** Currently, use of PDF 2.0 features in Typst is limited to minor improvements in accessibility, e.g. for the title element[3.6.22]. If you can do without these improvements, maximize compatibility by choosing part two (when you don't need attachments) or part three (when you do need attachments). If you rely on PDF 2.0 features, use part four.

Now, you only need to choose a **conformance level** (the lowercase letter at the end).

- **If you decided on part one, two, or three,** you should typically choose the accessible conformance level of your part, indicated by a lowercase `a` at the end. If your document is inherently inaccessible, e.g. an artist's portfolio that cannot be boiled down to alternative descriptions, choose conformance level `u` instead. Only if this results in a compiler error, e.g. because you used code points from the Unicode Private Use Area, use the basic level `b`.
- **If you have chosen part four,** you should choose the basic conformance level (PDF/A-4) except when needing to embed files.

When choosing between exporting PDF/A and regular PDF, keep in mind that PDF/A files contain additional metadata, and that some readers will prevent the user from modifying a PDF/A file.

3.14.3 PDF-specific functionality

Typst exposes PDF-specific functionality in the global `pdf` module. See below for the definitions it contains.

[A] https://en.wikipedia.org/wiki/Private_Use_Areas

This module contains some functions without a final API. They are designed to enhance accessibility for documents with complex tables. This includes `table-summary`[3.14.8], `header-cell`[3.14.7], and `data-cell`[3.14.6]. All of these functions will be removed in a future Typst release, either through integration into table functions or through full removal. You can enable these functions by passing `--features a11y-extras` or setting the `TYPST_FEATURES` environment variable to `a11y-extras`. In the web app, these features are not available at this time.

Definitions

3.14.4 Artifact Marks content as a PDF artifact. 489
3.14.5 Attach A file that will be attached to the output PDF. 491
3.14.6 Data Cell Explicitly defines this cell as a data cell. 493
3.14.7 Header Cell Explicitly defines a cell as a header cell. 495
3.14.8 Table Summary A summary of the purpose and structure of a complex table. 497

3.14.4 `artifact` Element

Marks content as a PDF artifact.

Artifacts are parts of the document that are not meant to be read by Assistive Technology (AT), such as screen readers. Typical examples include purely decorative images that do not contribute to the meaning of the document, watermarks, or repeated content such as page numbers.

Typst will automatically mark certain content, such as page headers, footers, backgrounds, and foregrounds, as artifacts. Likewise, paths and shapes are automatically marked as artifacts, but their content is not. Line numbers created using `par.line`[3.6.15/5.1] are automatically marked as artifacts, as are repetitions of table headers and footers.

Once something is marked as an artifact, you cannot make any of its contents accessible again. If you need to mark only part of something as an artifact, you may need to use this function multiple times.

If you are unsure what constitutes an artifact, check the Accessibility Guide[4.4.5].

In the future, this function may be moved out of the `pdf` module, making it possible to hide content in HTML export from AT.

/1 Parameters

```
artifact(
  kind: str,
  content,
) → content
```

/1.1 **kind** `str` *Settable* Default: `"other"`

The artifact kind.

You can improve accessibility by using the most specific artifact kind available. Your choice will govern how the PDF reader treats the artifact during reflow and content extraction (e.g. copy and paste).

Artifact types have been introduced in various different PDF specifications. Depending on which PDF version you target, Typst will select the most appropriate artifact type using your selection here.

Variant	Details
`"header"`	Repeats on the top of each page.
`"footer"`	Repeats at the bottom of each page.
`"watermark"`	Text or graphics in the back- or foreground of all pages.
`"page-number"`	Page numbers.
`"line-number"`	Line or paragraph numbers.
`"redaction"`	Placeholders for areas in which there was content in another rendition of the document which has since been removed.
`"bates"`	Bates numbering.
`"page"`	Not part of the document, but rather the page it is printed on. An example would be cut marks or color bars.
`"pagination-other"`	Artifacts arising from paginating the document not covered by other artifact types. This category generally applies if this artifact would not appear in your document if it was a website instead. If your artifact is covered by other categories, prefer them over this.
`"layout"`	Purely cosmetic content or typographical flourishes not contributing to the document's content.
`"background"`	Background of a page or a graphical element. This artifact kind was added in PDF 1.7. However, due to requirements in the PDF 1.7 specification that later specifications lifted, Typst only uses this artifact type in PDF 2.0. If you use it in a PDF 1.7 or earlier, Typst will use the `"other"` type instead.
`"other"`	Other artifacts.

/1.2 **body** `content` *Required* *Positional*

The content that is an artifact.

3.14.5 `attach` Element

A file that will be attached to the output PDF.

This can be used to distribute additional files associated with the PDF within it. PDF readers will display the files in a file listing.

Some international standards use this mechanism to attach machine-readable data (e.g., ZUGFeRD/Factur-X for invoices) that mirrors the visual content of the PDF.

/1 Example

```
#pdf.attach(
  "experiment.csv",
  relationship: "supplement",
  mime-type: "text/csv",
  description: "Raw Oxygen readings from the Arctic experiment",
)
```

/2 Notes

- This element is ignored if exporting to a format other than PDF.
- File attachments are not currently supported for PDF/A-2, even if the attached file conforms to PDF/A-1 or PDF/A-2.

/3 Parameters

```
attach(
  str path,
  bytes,
  relationship: none str,
  mime-type: none str,
  description: none str,
) → content
```

/3.1 **path** `str` or `path` *Required* *Positional*

The path of the file to be attached.

Must always be specified, but is only read from if no data is provided in the following argument.

/3.2 **data** `bytes` *Required* *Positional*

Raw file data, optionally.

If omitted, the data is read from the specified path.

/3.3 **relationship** `none` or `str` *Settable* Default: `none`

The relationship of the attached file to the document.

Ignored if export doesn't target PDF/A-3.

Variant	Details
`"source"`	The PDF document was created from the source file.
`"data"`	The file was used to derive a visual presentation in the PDF.
`"alternative"`	An alternative representation of the document.
`"supplement"`	Additional resources for the document.

/3.4 **mime-type** `none` or `str` *Settable* Default: `none`

The MIME type of the attached file.

/3.5 **description** `none` or `str` *Settable* Default: `none`

A description for the attached file.

3.14.6 data-cell

Explicitly defines this cell as a data cell.

Each cell in a table is either a header cell or a data cell. By default, all cells in `table.header`[3.6.20/4.4] are header cells, and all other cells data cells.

If your header contains a cell that is not a header cell, you can use this function to mark it as a data cell.

The API of this feature is temporary. Hence, calling this function requires enabling the `a11y-extras` feature flag at the moment. In a future Typst release, this functionality may move out of the `pdf` module so that tables in other export targets can contain the same information.

```
#show table.cell.where(x: 0): set text(weight: "bold")
#show table.cell.where(x: 1): set text(style: "italic")
#show table.cell.where(x: 1, y: 0): set text(style: "normal")

#table(
  columns: 3,
  align: (left, left, center),

  table.header[Objective][Key Result][Status],

  table.header(
    level: 2,
    table.cell(colspan: 2)[Improve Customer Satisfaction],
    // Status is data for this objective, not a header
    pdf.data-cell[✓ On Track],
  ),
  [], [Increase NPS to 50+], [45],
  [], [Reduce churn to \<5%], [4.2%],

  table.header(
    level: 2,
    table.cell(colspan: 2)[Grow Revenue],
    pdf.data-cell[⚠️ At Risk],
  ),
  [], [Achieve \$2M ARR], [\$1.8M],
  [], [Close 50 enterprise deals], [38],
)
```

Objective	Key Result	Status
Improve Customer Satisfaction		✓ On Track
	Increase NPS to 50+	45
	Reduce churn to <5%	4.2%
Grow Revenue		⚠️ At Risk
	Achieve $2M ARR	$1.8M
	Close 50 enterprise deals	38

/1 ## Parameters

```
data-cell(content) → content
```

/1.1 **cell** `content` *Required* *Positional*

The table cell.

This can be content or a call to `table.cell`[3.6.20/4.1].

3.14.7 header-cell

Explicitly defines a cell as a header cell.

Header cells help users of Assistive Technology (AT) understand and navigate complex tables. When your table is correctly marked up with header cells, AT can announce the relevant header information on-demand when entering a cell.

By default, Typst will automatically mark all cells within `table.header`[3.6.20/4.4] as header cells. They will apply to the columns below them. You can use that function's `level`[3.6.20/4.4.2] parameter to make header cells labelled by other header cells.

The `pdf.header-cell` function allows you to indicate that a cell is a header cell in the following additional situations:

- You have a **header column** in which each cell applies to its row. In that case, you pass `"row"` as an argument to the scope parameter[3.14.7/1.2] to indicate that the header cell applies to the row.
- You have a cell in `table.header`[3.6.20/4.4], for example at the very start, that labels both its row and column. In that case, you pass `"both"` as an argument to the `scope`[3.14.7/1.2] parameter.
- You have a header cell in a row not containing other header cells. In that case, you can use this function to mark it as a header cell.

The API of this feature is temporary. Hence, calling this function requires enabling the `a11y-extras` feature flag at the moment. In a future Typst release, this functionality may move out of the `pdf` module so that tables in other export targets can contain the same information.

```
#show table.cell.where(x: 0): set text(weight: "medium")
#show table.cell.where(y: 0): set text(weight: "bold")

#table(
  columns: 3,
  align: (start, end, end),

  table.header(
    // Top-left cell: Labels both the nutrient rows
    // and the serving size columns.
    pdf.header-cell(scope: "both")[Nutrient],
    [Per 100g],
    [Per Serving],
  ),

  // First column cells are row headers
  pdf.header-cell(scope: "row")[Calories],
  [250 kcal], [375 kcal],
  pdf.header-cell(scope: "row")[Protein],
  [8g], [12g],
  pdf.header-cell(scope: "row")[Fat],
  [12g], [18g],
  pdf.header-cell(scope: "row")[Carbs],
  [30g], [45g],
)
```

Nutrient	Per 100g	Per Serving
Calories	250 kcal	375 kcal
Protein	8g	12g
Fat	12g	18g
Carbs	30g	45g

/1 Parameters

```
header-cell(
  level: int,
  scope: str,
  content,
) → content
```

/1.1 level `int` Default: 1

The nesting level of this header cell.

/1.2 scope `str` Default: "column"

What track of the table this header cell applies to.

Variant	Details
"both"	The header cell refers to both the row and the column.
"column"	The header cell refers to the column.
"row"	The header cell refers to the row.

/1.3 cell `content` *Required* *Positional*

The table cell.

This can be content or a call to `table.cell`[3.6.20/4.1].

3.14.8 table-summary

A summary of the purpose and structure of a complex table.

This will be available for Assistive Technology (AT), such as screen readers, when exporting to PDF, but not for sighted readers of your file.

This field is intended for instructions that help the user navigate the table using AT. It is not an alternative description, so do not duplicate the contents of the table within. Likewise, do not use this for the core takeaway of the table. Instead, include that in the text around the table or, even better, in a figure caption[3.6.8/6.1].

If in doubt whether your table is complex enough to warrant a summary, err on the side of not including one. If you are certain that your table is complex enough, consider whether a sighted user might find it challenging. They might benefit from the instructions you put here, so consider printing them visibly in the document instead.

The API of this feature is temporary. Hence, calling this function requires enabling the `a11y-extras` feature flag at the moment. Even if this functionality should be available without a feature flag in the future, the summary will remain exclusive to PDF export.

```
#figure(
  pdf.table-summary(
    // The summary just provides orientation and structural
    // information for AT users.
    summary: "The first two columns list the names of each participant. The last
column contains cells spanning multiple rows for their assigned group.",
    table(
      columns: 3,
      table.header[First Name][Given Name][Group],
      [Mike], [Davis], table.cell(rowspan: 3)[Sales],
      [Anna], [Smith],
      [John], [Johnson],
      [Sara], [Wilkins], table.cell(rowspan: 2)[Operations],
      [Tom], [Brown],
    ),
  ),
  // This is the key takeaway of the table, so we put it in the caption.
  caption: [The Sales org now has a new member],
)
```

First Name	Given Name	Group
Mike	Davis	Sales
Anna	Smith	
John	Johnson	
Sara	Wilkins	Operations
Tom	Brown	

Table 1: The Sales org now has a new member

/1 Parameters

```
table-summary(
  summary: str,
  content,
) → content
```

/1.1 **summary** `str`

/1.2 **table** `content` *Required* *Positional*

The table.

3.15 HTML

INFO

Typst's HTML export is currently under active development. The feature is still very incomplete and only available for experimentation behind a feature flag. Do not use this feature for production use cases. In the CLI, you can experiment with HTML export by passing `--features html` or setting the `TYPST_FEATURES` environment variable to `html`. In the web app, HTML export is not available at this time. Visit the tracking issue[A] to follow progress on HTML export and learn more about planned features.

HTML files describe a document structurally. The aim of Typst's HTML export is to capture the structure of an input document and produce semantically rich HTML that retains this structure. The resulting HTML should be accessible, human-readable, and editable by hand and downstream tools.

PDF, PNG, and SVG export, in contrast, all produce *visual* representations of a fully-laid out document. This divergence in the formats' intents means that Typst cannot simply produce perfect HTML for your existing Typst documents. It cannot always know what the best semantic HTML representation of your content is.

Instead, it gives *you* full control: You can check the current export format through the `target`[3.5.30] function and when it is set to HTML, generate raw HTML elements[3.15.3]. The primary intended use of these elements is in templates and show rules. This way, the document's contents can be fully agnostic to the export target and content can be shared between PDF and HTML export.

In the `html` export format, Typst will output a single HTML file. You can also use Typst to create a website comprising multiple HTML documents and additional assets. Typst calls such a collection a *bundle* and supports this use case with the dedicated `bundle` target[3.18]. Refer to its documentation to learn more. Note that both in the `html` and the `bundle` target, Typst will currently always emit a standalone HTML file. Support for emitting fragments that can be integrated into other HTML documents is planned for the future.

Typst currently does not output CSS style sheets, instead focussing on emitting semantic markup. You can of course write your own CSS styles and still benefit from sharing your *content* between PDF and HTML. For the future, we plan to give you the option of automatically emitting CSS, taking more of your existing set rules into account.

3.15.1 Exporting as HTML

/1 Command Line

Pass `--format html` to the `compile` or `watch` subcommand or provide an output file name that ends with `.html`. Note that you must also pass `--features html` or set `TYPST_FEATURES=html` to enable this experimental export target.

When using `typst watch`, Typst will launch a live-reloading HTTP server. You can configure it as follows:

- Pass `--port` to change the port. (Defaults to the first free port in the range 3000-3005.)
- Pass `--no-reload` to disable injection of a live reload script. (The HTML that is written to disk isn't affected either way.)
- Pass `--no-serve` to disable the server altogether.

/2 Web App

Not currently available.

3.15.2 HTML-specific functionality

Typst exposes HTML-specific functionality in the global `html` module. See below for the definitions it contains.

Definitions

3.15.3 Elem An HTML element that can contain Typst content. 501

[A] https://github.com/typst/typst/issues/5512

3.15.4 Frame An element that lays out its content as an inline SVG. 502
3.15.5 Typed HTML A typed layer over raw HTML elements. 503

3.15.3 `elem` Element

An HTML element that can contain Typst content.

Typst's HTML export automatically generates the appropriate tags for most elements. However, sometimes, it is desirable to retain more control. For example, when using Typst to generate your blog, you could use this function to wrap each article in an `<article>` tag.

Typst is aware of what is valid HTML. A tag and its attributes must form syntactically valid HTML. Some tags, like `meta` do not accept content. Hence, you must not provide a body for them. We may add more checks in the future, so be sure that you are generating valid HTML when using this function.

Normally, Typst will generate `html`, `head`, and `body` tags for you. If you instead create them with this function, Typst will omit its own tags.

```
#html.elem("div", attrs: (style: "background: aqua"))[
  A div with _Typst content_ inside!
]
```

/1 Parameters

```
elem(
  str,
  attrs: dictionary,
  none content,
) → content
```

/1.1 **tag** `str` *Required* *Positional*

The element's tag.

/1.2 **attrs** `dictionary` *Settable* Default: `(:)`

The element's HTML attributes.

/1.3 **body** `none` or `content` *Positional* *Settable* Default: `none`

The contents of the HTML element.

The body can be arbitrary Typst content.

3.15.4 `frame` Element

An element that lays out its content as an inline SVG.

Sometimes, converting Typst content to HTML is not desirable. This can be the case for plots and other content that relies on positioning and styling to convey its message.

This function allows you to use the Typst layout engine that would also be used for PDF, SVG, and PNG export to render a part of your document exactly how it would appear when exported in one of these formats. It embeds the content as an inline SVG.

/1 Parameters

```
frame(content) → content
```

/1.1 body `content` *Required* *Positional*

The content that shall be laid out.

3.15.5 Typed HTML

A typed layer over raw HTML elements.

The `html` module provides a typed layer over the raw `html.elem`[3.15.3] function that allows you to conveniently create HTML elements. HTML attributes are exposed as function parameters that accept Typst types and automatically take care of converting those into the appropriate HTML.

Some parameters are common to all typed HTML functions. These are listed at the bottom in the Global Attributes[3.15.5/3] section instead of explicitly on each element for readability.

/1 Example

```
#html.video(
  controls: true,
  width: 1280,
  height: 720,
  src: "sunrise.mp4",
)[
  Your browser does not support the video tag.
]
```

/2 Functions

/2.1 a

Hyperlink.

```
html.a(
  download: str,
  href: str,
  hreflang: str,
  ping: str array,
  referrerpolicy: none str,
  rel: str array,
  target: str,
  type: str,
  content,
) → content
```

/2.1.1 download str

Whether to download the resource instead of navigating to it, and its filename if so.

/2.1.2 href str

Address of the hyperlink.

/2.1.3 hreflang str

Language of the linked resource.

/2.1.4 ping str or array

URLs to ping.

/2.1.5 referrerpolicy none or str

Referrer policy for fetches initiated by the element.

Variant	Details
`"no-referrer"`	
`"no-referrer-when-downgrade"`	
`"same-origin"`	

Variant	Details
`"origin"`	
`"strict-origin"`	
`"origin-when-cross-origin"`	
`"strict-origin-when-cross-origin"`	
`"unsafe-url"`	

/2.1.6 **`rel`** `str` or `array`

Relationship between the location in the document containing the hyperlink and the destination resource.

Variant	Details
`"alternate"`	
`"canonical"`	
`"author"`	
`"bookmark"`	
`"dns-prefetch"`	
`"expect"`	
`"external"`	
`"help"`	
`"icon"`	
`"manifest"`	
`"modulepreload"`	
`"license"`	
`"next"`	
`"nofollow"`	
`"noopener"`	
`"noreferrer"`	
`"opener"`	
`"pingback"`	
`"preconnect"`	
`"prefetch"`	
`"preload"`	
`"prev"`	
`"privacy-policy"`	
`"search"`	
`"stylesheet"`	
`"tag"`	
`"terms-of-service"`	

/2.1.7 **`target`** `str`

Navigable for hyperlink navigation.

Variant	Details
`"_blank"`	
`"_self"`	
`"_parent"`	
`"_top"`	

/2.1.8 **`type`** `str`

Hint for the type of the referenced resource.

/2.1.9 **body** `content` *Positional*

The contents of the HTML element.

/2.2 abbr

Abbreviation.

```
html.abbr(content) → content
```

/2.2.1 **body** `content` *Positional*

The contents of the HTML element.

/2.3 address

Contact information for a page or article element.

```
html.address(content) → content
```

/2.3.1 **body** `content` *Positional*

The contents of the HTML element.

/2.4 area

Hyperlink or dead area on an image map.

```
html.area(
  alt: str,
  coords: array,
  download: str,
  href: str,
  ping: str array,
  referrerpolicy: none str,
  rel: str array,
  shape: str,
  target: str,
) → content
```

/2.4.1 **alt** `str`

Replacement text for use when images are not available.

/2.4.2 **coords** `array`

Coordinates for the shape to be created in an image map. Expects an array of floating point numbers.

/2.4.3 **download** `str`

Whether to download the resource instead of navigating to it, and its filename if so.

/2.4.4 **href** `str`

Address of the hyperlink.

/2.4.5 **ping** `str` or `array`

URLs to ping.

/2.4.6 **referrerpolicy** `none` or `str`

Referrer policy for fetches initiated by the element.

Variant	Details
`"no-referrer"`	
`"no-referrer-when-downgrade"`	
`"same-origin"`	

Variant	Details
`"origin"`	
`"strict-origin"`	
`"origin-when-cross-origin"`	
`"strict-origin-when-cross-origin"`	
`"unsafe-url"`	

/2.4.7 **`rel`** `str` or `array`

Relationship between the location in the document containing the hyperlink and the destination resource.

Variant	Details
`"alternate"`	
`"canonical"`	
`"author"`	
`"bookmark"`	
`"dns-prefetch"`	
`"expect"`	
`"external"`	
`"help"`	
`"icon"`	
`"manifest"`	
`"modulepreload"`	
`"license"`	
`"next"`	
`"nofollow"`	
`"noopener"`	
`"noreferrer"`	
`"opener"`	
`"pingback"`	
`"preconnect"`	
`"prefetch"`	
`"preload"`	
`"prev"`	
`"privacy-policy"`	
`"search"`	
`"stylesheet"`	
`"tag"`	
`"terms-of-service"`	

/2.4.8 **`shape`** `str`

The kind of shape to be created in an image map.

Variant	Details
`"circle"`	
`"default"`	
`"poly"`	
`"rect"`	

/2.4.9 **`target`** `str`

Navigable for hyperlink navigation.

Variant	Details
`"_blank"`	
`"_self"`	
`"_parent"`	
`"_top"`	

/2.5 article

Self-contained syndicatable or reusable composition.

```
html.article(content) → content
```

/2.5.1 body `content` *Positional*

The contents of the HTML element.

/2.6 aside

Sidebar for tangentially related content.

```
html.aside(content) → content
```

/2.6.1 body `content` *Positional*

The contents of the HTML element.

/2.7 audio

Audio player.

```
html.audio(
  autoplay: bool,
  controls: bool,
  crossorigin: str,
  loop: bool,
  muted: bool,
  preload: none auto str,
  src: str,
  content,
) → content
```

/2.7.1 autoplay `bool`

Hint that the media resource can be started automatically when the page is loaded.

/2.7.2 controls `bool`

Show user agent controls.

/2.7.3 crossorigin `str`

How the element handles crossorigin requests.

Variant	Details
`"anonymous"`	
`"use-credentials"`	

/2.7.4 loop `bool`

Whether to loop the media resource.

/2.7.5 muted `bool`

Whether to mute the media resource by default.

/2.7.6 **preload** `none` or `auto` or `str`

Hints how much buffering the media resource will likely need.

Variant	Details
`"metadata"`	

/2.7.7 **src** `str`

Address of the resource.

/2.7.8 **body** `content` *Positional*

The contents of the HTML element.

/2.8 **b**

Keywords.

```
html.b(content) → content
```

/2.8.1 **body** `content` *Positional*

The contents of the HTML element.

/2.9 **base**

Base URL and default target navigable for hyperlinks and forms.

```
html.base(
  href: str,
  target: str,
) → content
```

/2.9.1 **href** `str`

Document base URL.

/2.9.2 **target** `str`

Default navigable for hyperlink navigation and form submission.

Variant	Details
`"_blank"`	
`"_self"`	
`"_parent"`	
`"_top"`	

/2.10 **bdi**

Text directionality isolation.

```
html.bdi(content) → content
```

/2.10.1 **body** `content` *Positional*

The contents of the HTML element.

/2.11 **bdo**

Text directionality formatting.

```
html.bdo(content) → content
```

/2.11.1 **body** `content` *Positional*

The contents of the HTML element.

/2.12 blockquote

A section quoted from another source.

```
html.blockquote(
  cite: str,
  content,
) → content
```

/2.12.1 cite `str`

Link to the source of the quotation or more information about the edit.

/2.12.2 body `content` *Positional*

The contents of the HTML element.

/2.13 body

Document body.

```
html.body(content) → content
```

/2.13.1 body `content` *Positional*

The contents of the HTML element.

/2.14 br

Line break, e.g. in poem or postal address.

```
html.br() → content
```

/2.15 button

Button control.

```
html.button(
  command: str,
  commandfor: str,
  disabled: bool,
  form: str,
  formaction: str,
  formenctype: str,
  formmethod: str,
  formnovalidate: bool,
  formtarget: str,
  name: str,
  popovertarget: str,
  popovertargetaction: str,
  type: str,
  value: str,
  content,
) → content
```

/2.15.1 command `str`

Indicates to the targeted element which action to take.

Variant	Details
`"toggle-popover"`	

Variant	Details
`"show-popover"`	
`"hide-popover"`	
`"close"`	
`"request-close"`	
`"show-modal"`	

/2.15.2 **`commandfor`** `str`

Targets another element to be invoked.

/2.15.3 **`disabled`** `bool`

Whether the form control is disabled.

/2.15.4 **`form`** `str`

Associates the element with a form element.

/2.15.5 **`formaction`** `str`

URL to use for form submission.

/2.15.6 **`formenctype`** `str`

Entry list encoding type to use for form submission.

Variant	Details
`"application/x-www-form-urlencoded"`	
`"multipart/form-data"`	
`"text/plain"`	

/2.15.7 **`formmethod`** `str`

Variant to use for form submission.

Variant	Details
`"GET"`	
`"POST"`	
`"dialog"`	

/2.15.8 **`formnovalidate`** `bool`

Bypass form control validation for form submission.

/2.15.9 **`formtarget`** `str`

Navigable for form submission.

Variant	Details
`"_blank"`	
`"_self"`	
`"_parent"`	
`"_top"`	

/2.15.10 **`name`** `str`

Name of the element to use for form submission and in the form.elements API.

/2.15.11 **`popovertarget`** `str`

Targets a popover element to toggle, show, or hide.

/2.15.12 **`popovertargetaction`** `str`

Indicates whether a targeted popover element is to be toggled, shown, or hidden.

Variant	Details
`"toggle"`	
`"show"`	
`"hide"`	

/2.15.13 **type** `str`

Type of button.

Variant	Details
`"submit"`	
`"reset"`	
`"button"`	

/2.15.14 **value** `str`

Value to be used for form submission.

/2.15.15 **body** `content` *Positional*

The contents of the HTML element.

/2.16 **canvas**

Scriptable bitmap canvas.

```
html.canvas(
  height: int,
  width: int,
  content,
) → content
```

/2.16.1 **height** `int`

Vertical dimension.

/2.16.2 **width** `int`

Horizontal dimension.

/2.16.3 **body** `content` *Positional*

The contents of the HTML element.

/2.17 **caption**

Table caption.

```
html.caption(content) → content
```

/2.17.1 **body** `content` *Positional*

The contents of the HTML element.

/2.18 **cite**

Title of a work.

```
html.cite(content) → content
```

/2.18.1 **body** `content` *Positional*

The contents of the HTML element.

/2.19 **code**

Computer code.

```
html.code(content) → content
```

/2.19.1 **body** `content` *Positional*

The contents of the HTML element.

/2.20 **col**

Table column.

```
html.col(span: int) → content
```

/2.20.1 **span** `int`

Number of columns spanned by the element.

/2.21 **colgroup**

Group of columns in a table.

```
html.colgroup(
  span: int,
  content,
) → content
```

/2.21.1 **span** `int`

Number of columns spanned by the element.

/2.21.2 **body** `content` *Positional*

The contents of the HTML element.

/2.22 **data**

Machine-readable equivalent.

```
html.data(
  value: str,
  content,
) → content
```

/2.22.1 **value** `str`

Machine-readable value.

/2.22.2 **body** `content` *Positional*

The contents of the HTML element.

/2.23 **datalist**

Container for options for combo box control.

```
html.datalist(content) → content
```

/2.23.1 **body** `content` *Positional*

The contents of the HTML element.

/2.24 **dd**

Content for corresponding dt element(s).

```
html.dd(content) → content
```

/2.24.1 **body** `content` *Positional*

The contents of the HTML element.

/2.25 **del**

A removal from the document.

```
html.del(
  cite: str,
  datetime: datetime,
  content,
) → content
```

/2.25.1 **cite** `str`

Link to the source of the quotation or more information about the edit.

/2.25.2 **datetime** `datetime`

Date and (optionally) time of the change.

/2.25.3 **body** `content` *Positional*

The contents of the HTML element.

/2.26 **details**

Disclosure control for hiding details.

```
html.details(
  name: str,
  open: bool,
  content,
) → content
```

/2.26.1 **name** `str`

Name of group of mutually-exclusive details elements.

/2.26.2 **open** `bool`

Whether the details are visible.

/2.26.3 **body** `content` *Positional*

The contents of the HTML element.

/2.27 **dfn**

Defining instance.

```
html.dfn(content) → content
```

/2.27.1 **body** `content` *Positional*

The contents of the HTML element.

/2.28 **dialog**

Dialog box or window.

```
html.dialog(
  open: bool,
  content,
) → content
```

/2.28.1 **open** `bool`

Whether the dialog box is showing.

/2.28.2 **body** `content` *Positional*

The contents of the HTML element.

/2.29 div

Generic flow container, or container for name-value groups in dl elements.

```
html.div(content) → content
```

/2.29.1 **body** `content` *Positional*

The contents of the HTML element.

/2.30 dl

Association list consisting of zero or more name-value groups.

```
html.dl(content) → content
```

/2.30.1 **body** `content` *Positional*

The contents of the HTML element.

/2.31 dt

Legend for corresponding dd element(s).

```
html.dt(content) → content
```

/2.31.1 **body** `content` *Positional*

The contents of the HTML element.

/2.32 em

Stress emphasis.

```
html.em(content) → content
```

/2.32.1 **body** `content` *Positional*

The contents of the HTML element.

/2.33 embed

Plugin.

```
html.embed(
  height: int,
  src: str,
  type: str,
  width: int,
) → content
```

/2.33.1 **height** `int`

Vertical dimension.

/2.33.2 **src** `str`

Address of the resource.

/2.33.3 **type** `str`

Type of embedded resource.

/2.33.4 **width** `int`

Horizontal dimension.

/2.34 **fieldset**

Group of form controls.

```
html.fieldset(
  disabled: bool,
  form: str,
  name: str,
  content,
) → content
```

/2.34.1 **disabled** `bool`

Whether the descendant form controls, except any inside legend, are disabled.

/2.34.2 **form** `str`

Associates the element with a form element.

/2.34.3 **name** `str`

Name of the element to use for form submission and in the form.elements API.

/2.34.4 **body** `content` *Positional*

The contents of the HTML element.

/2.35 **figcaption**

Caption for figure.

```
html.figcaption(content) → content
```

/2.35.1 **body** `content` *Positional*

The contents of the HTML element.

/2.36 **figure**

Figure with optional caption.

```
html.figure(content) → content
```

/2.36.1 **body** `content` *Positional*

The contents of the HTML element.

/2.37 **footer**

Footer for a page or section.

```
html.footer(content) → content
```

/2.37.1 **body** `content` *Positional*

The contents of the HTML element.

/2.38 **form**

User-submittable form.

```
html.form(
  accept-charset: str,
  action: str,
  autocomplete: bool,
```

```
  enctype: str,
  method: str,
  name: str,
  novalidate: bool,
  rel: str array,
  target: str,
  content,
) -> content
```

/2.38.1 **accept-charset** `str`

Character encodings to use for form submission.

Variant	Details
`"utf-8"`	

/2.38.2 **action** `str`

URL to use for form submission.

/2.38.3 **autocomplete** `bool`

Default setting for autofill feature for controls in the form.

/2.38.4 **enctype** `str`

Entry list encoding type to use for form submission.

Variant	Details
`"application/x-www-form-urlencoded"`	
`"multipart/form-data"`	
`"text/plain"`	

/2.38.5 **method** `str`

Variant to use for form submission.

Variant	Details
`"GET"`	
`"POST"`	
`"dialog"`	

/2.38.6 **name** `str`

Name of form to use in the document.forms API.

/2.38.7 **novalidate** `bool`

Bypass form control validation for form submission.

/2.38.8 **rel** `str` or `array`

Relationship between the document containing the form and its action destination

Variant	Details
`"alternate"`	
`"canonical"`	
`"author"`	
`"bookmark"`	
`"dns-prefetch"`	
`"expect"`	
`"external"`	

Variant	Details
`"help"`	
`"icon"`	
`"manifest"`	
`"modulepreload"`	
`"license"`	
`"next"`	
`"nofollow"`	
`"noopener"`	
`"noreferrer"`	
`"opener"`	
`"pingback"`	
`"preconnect"`	
`"prefetch"`	
`"preload"`	
`"prev"`	
`"privacy-policy"`	
`"search"`	
`"stylesheet"`	
`"tag"`	
`"terms-of-service"`	

/2.38.9 **target** `str`

Navigable for form submission.

Variant	Details
`"_blank"`	
`"_self"`	
`"_parent"`	
`"_top"`	

/2.38.10 **body** `content` *Positional*

The contents of the HTML element.

/2.39 **h1**

Heading.

```
html.h1(content) → content
```

/2.39.1 **body** `content` *Positional*

The contents of the HTML element.

/2.40 **h2**

Heading.

```
html.h2(content) → content
```

/2.40.1 **body** `content` *Positional*

The contents of the HTML element.

/2.41 **h3**

Heading.

```
html.h3(content) → content
```

/2.41.1 **body** `content` *Positional*

The contents of the HTML element.

/2.42 **h4**

Heading.

```
html.h4(content) → content
```

/2.42.1 **body** `content` *Positional*

The contents of the HTML element.

/2.43 **h5**

Heading.

```
html.h5(content) → content
```

/2.43.1 **body** `content` *Positional*

The contents of the HTML element.

/2.44 **h6**

Heading.

```
html.h6(content) → content
```

/2.44.1 **body** `content` *Positional*

The contents of the HTML element.

/2.45 **head**

Container for document metadata.

```
html.head(content) → content
```

/2.45.1 **body** `content` *Positional*

The contents of the HTML element.

/2.46 **header**

Introductory or navigational aids for a page or section.

```
html.header(content) → content
```

/2.46.1 **body** `content` *Positional*

The contents of the HTML element.

/2.47 **hgroup**

Heading container.

```
html.hgroup(content) → content
```

/2.47.1 **body** `content` *Positional*

The contents of the HTML element.

/2.48 **hr**

Thematic break.

```
html.hr() → content
```

/2.49 **html**

Root element.

```
html.html(content) → content
```

/2.49.1 **body** `content` *Positional*

The contents of the HTML element.

/2.50 **i**

Alternate voice.

```
html.i(content) → content
```

/2.50.1 **body** `content` *Positional*

The contents of the HTML element.

/2.51 **iframe**

Child navigable.

```
html.iframe(
  allow: str,
  allowfullscreen: bool,
  height: int,
  loading: str,
  name: str,
  referrerpolicy: none str,
  sandbox: str array,
  src: str,
  srcdoc: str,
  width: int,
  content,
) → content
```

/2.51.1 **allow** `str`

Permissions policy to be applied to the iframe's contents.

/2.51.2 **allowfullscreen** `bool`

Whether to allow the iframe's contents to use requestFullscreen().

/2.51.3 **height** `int`

Vertical dimension.

/2.51.4 **loading** `str`

Used when determining loading deferral.

Variant	Details
`"lazy"`	
`"eager"`	

/2.51.5 **name** `str`

Name of content navigable.

Variant	Details
`"_blank"`	
`"_self"`	
`"_parent"`	
`"_top"`	

/2.51.6 **referrerpolicy** `none` or `str`

Referrer policy for fetches initiated by the element.

Variant	Details
`"no-referrer"`	
`"no-referrer-when-downgrade"`	
`"same-origin"`	
`"origin"`	
`"strict-origin"`	
`"origin-when-cross-origin"`	
`"strict-origin-when-cross-origin"`	
`"unsafe-url"`	

/2.51.7 **sandbox** `str` or `array`

Security rules for nested content.

Variant	Details
`"allow-downloads"`	
`"allow-forms"`	
`"allow-modals"`	
`"allow-orientation-lock"`	
`"allow-pointer-lock"`	
`"allow-popups"`	
`"allow-popups-to-escape-sandbox"`	
`"allow-presentation"`	
`"allow-same-origin"`	
`"allow-scripts"`	
`"allow-top-navigation"`	
`"allow-top-navigation-by-user-activation"`	
`"allow-top-navigation-to-custom-protocols"`	

/2.51.8 **src** `str`

Address of the resource.

/2.51.9 **srcdoc** `str`

A document to render in the iframe.

/2.51.10 **width** `int`

Horizontal dimension.

/2.51.11 **body** `content` *Positional*

The contents of the HTML element.

/2.52 **img**

Image.

```
html.img(
  alt: str,
  crossorigin: str,
  decoding: auto str,
  fetchpriority: auto str,
  height: int,
  ismap: bool,
  loading: str,
  referrerpolicy: none str,
  sizes: array,
  src: str,
  srcset: array,
  usemap: str,
  width: int,
) -> content
```

/2.52.1 **alt** `str`

Replacement text for use when images are not available.

/2.52.2 **crossorigin** `str`

How the element handles crossorigin requests.

Variant	Details
`"anonymous"`	
`"use-credentials"`	

/2.52.3 **decoding** `auto` or `str`

Decoding hint to use when processing this image for presentation.

Variant	Details
`"sync"`	
`"async"`	

/2.52.4 **fetchpriority** `auto` or `str`

Sets the priority for fetches initiated by the element.

Variant	Details
`"high"`	
`"low"`	

/2.52.5 **height** `int`

Vertical dimension.

/2.52.6 **ismap** `bool`

Whether the image is a server-side image map.

/2.52.7 **loading** `str`

Used when determining loading deferral.

Variant	Details
`"lazy"`	
`"eager"`	

/2.52.8 **referrerpolicy** `none` or `str`

Referrer policy for fetches initiated by the element.

Variant	Details
`"no-referrer"`	
`"no-referrer-when-downgrade"`	
`"same-origin"`	
`"origin"`	
`"strict-origin"`	
`"origin-when-cross-origin"`	
`"strict-origin-when-cross-origin"`	
`"unsafe-url"`	

/2.52.9 **sizes** `array`

Image sizes for different page layouts. Expects an array of dictionaries with the keys `condition` (string) and `size` (length).

/2.52.10 **src** `str`

Address of the resource.

/2.52.11 **srcset** `array`

Images to use in different situations, e.g., high-resolution displays, small monitors, etc. Expects an array of dictionaries with the keys `src` (string) and `width` (integer) or `density` (float).

/2.52.12 **usemap** `str`

Name of image map to use.

/2.52.13 **width** `int`

Horizontal dimension.

/2.53 **input**

Form control.

```
html.input(
  accept: str array,
  alpha: bool,
  alt: str,
  autocomplete: str array,
  checked: bool,
  colorspace: str,
  dirname: str,
  disabled: bool,
  form: str,
  formaction: str,
  formenctype: str,
  formmethod: str,
  formnovalidate: bool,
  formtarget: str,
  height: int,
  list: str,
  max: float datetime str,
  maxlength: int,
  min: float datetime str,
  minlength: int,
  multiple: bool,
```

```
  name: str,
  pattern: str,
  placeholder: str,
  popovertarget: str,
  popovertargetaction: str,
  readonly: bool,
  required: bool,
  size: int,
  src: str,
  step: float str,
  type: str,
  value: float color datetime str array,
  width: int,
) → content
```

/2.53.1 **accept** `str` or `array`

Hint for expected file type in file upload controls.

/2.53.2 **alpha** `bool`

Allow the color's alpha component to be set.

/2.53.3 **alt** `str`

Replacement text for use when images are not available.

/2.53.4 **autocomplete** `str` or `array`

Hint for form autofill feature.

Variant	Details
`"shipping"`	
`"billing"`	
`"name"`	
`"honorific-prefix"`	
`"given-name"`	
`"additional-name"`	
`"family-name"`	
`"honorific-suffix"`	
`"nickname"`	
`"username"`	
`"new-password"`	
`"current-password"`	
`"one-time-code"`	
`"organization-title"`	
`"organization"`	
`"street-address"`	
`"address-line1"`	
`"address-line2"`	
`"address-line3"`	
`"address-level4"`	
`"address-level3"`	
`"address-level2"`	
`"address-level1"`	
`"country"`	
`"country-name"`	

Variant	Details
`"postal-code"`	
`"cc-name"`	
`"cc-given-name"`	
`"cc-additional-name"`	
`"cc-family-name"`	
`"cc-number"`	
`"cc-exp"`	
`"cc-exp-month"`	
`"cc-exp-year"`	
`"cc-csc"`	
`"cc-type"`	
`"transaction-currency"`	
`"transaction-amount"`	
`"language"`	
`"bday"`	
`"bday-day"`	
`"bday-month"`	
`"bday-year"`	
`"sex"`	
`"url"`	
`"photo"`	
`"home"`	
`"work"`	
`"mobile"`	
`"fax"`	
`"pager"`	
`"tel"`	
`"tel-country-code"`	
`"tel-national"`	
`"tel-area-code"`	
`"tel-local"`	
`"tel-local-prefix"`	
`"tel-local-suffix"`	
`"tel-extension"`	
`"email"`	
`"impp"`	

/2.53.5 **checked** `bool`

Whether the control is checked.

/2.53.6 **colorspace** `str`

The color space of the serialized color.

Variant	Details
`"limited-srgb"`	
`"display-p3"`	

/2.53.7 **dirname** `str`

Name of form control to use for sending the element's directionality in form submission.

/2.53.8 **disabled** `bool`

Whether the form control is disabled.

/2.53.9 **form** `str`

Associates the element with a form element.

/2.53.10 **formaction** `str`

URL to use for form submission.

/2.53.11 **formenctype** `str`

Entry list encoding type to use for form submission.

Variant	Details
`"application/x-www-form-urlencoded"`	
`"multipart/form-data"`	
`"text/plain"`	

/2.53.12 **formmethod** `str`

Variant to use for form submission.

Variant	Details
`"GET"`	
`"POST"`	
`"dialog"`	

/2.53.13 **formnovalidate** `bool`

Bypass form control validation for form submission.

/2.53.14 **formtarget** `str`

Navigable for form submission.

Variant	Details
`"_blank"`	
`"_self"`	
`"_parent"`	
`"_top"`	

/2.53.15 **height** `int`

Vertical dimension.

/2.53.16 **list** `str`

List of autocomplete options.

/2.53.17 **max** `float` or `datetime` or `str`

Maximum value.

/2.53.18 **maxlength** `int`

Maximum length of value.

/2.53.19 **min** `float` or `datetime` or `str`

Minimum value.

/2.53.20 **minlength** `int`

Minimum length of value.

/2.53.21 **multiple** `bool`

Whether to allow multiple values.

/2.53.22 **name** `str`

Name of the element to use for form submission and in the form.elements API.

/2.53.23 **pattern** `str`

Pattern to be matched by the form control's value.

/2.53.24 **placeholder** `str`

User-visible label to be placed within the form control.

/2.53.25 **popovertarget** `str`

Targets a popover element to toggle, show, or hide.

/2.53.26 **popovertargetaction** `str`

Indicates whether a targeted popover element is to be toggled, shown, or hidden.

Variant	Details
`"toggle"`	
`"show"`	
`"hide"`	

/2.53.27 **readonly** `bool`

Whether to allow the value to be edited by the user.

/2.53.28 **required** `bool`

Whether the control is required for form submission.

/2.53.29 **size** `int`

Size of the control.

/2.53.30 **src** `str`

Address of the resource.

/2.53.31 **step** `float` or `str`

Granularity to be matched by the form control's value.

Variant	Details
`"any"`	

/2.53.32 **type** `str`

Type of form control.

Variant	Details
`"hidden"`	
`"text"`	
`"search"`	
`"tel"`	
`"url"`	
`"email"`	
`"password"`	
`"date"`	
`"month"`	
`"week"`	
`"time"`	
`"datetime-local"`	
`"number"`	
`"range"`	

Variant	Details
`"color"`	
`"checkbox"`	
`"radio"`	
`"file"`	
`"submit"`	
`"image"`	
`"reset"`	
`"button"`	

/2.53.33 **value** `float` or `color` or `datetime` or `str` or `array`

Value of the form control.

/2.53.34 **width** `int`

Horizontal dimension.

/2.54 ins

An addition to the document.

```
html.ins(
  cite: str,
  datetime: datetime,
  content,
) → content
```

/2.54.1 **cite** `str`

Link to the source of the quotation or more information about the edit.

/2.54.2 **datetime** `datetime`

Date and (optionally) time of the change.

/2.54.3 **body** `content` *Positional*

The contents of the HTML element.

/2.55 kbd

User input.

```
html.kbd(content) → content
```

/2.55.1 **body** `content` *Positional*

The contents of the HTML element.

/2.56 label

Caption for a form control.

```
html.label(
  for: str,
  content,
) → content
```

/2.56.1 **for** `str`

Associate the label with form control.

/2.56.2 **body** `content` *Positional*

The contents of the HTML element.

/2.57 `legend`

Caption for fieldset.

```
html.legend(content) → content
```

/2.57.1 `body` `content` *Positional*

The contents of the HTML element.

/2.58 `li`

List item.

```
html.li(
  value: int,
  content,
) → content
```

/2.58.1 `value` `int`

Ordinal value of the list item.

/2.58.2 `body` `content` *Positional*

The contents of the HTML element.

/2.59 `link`

Link metadata.

```
html.link(
  as: str,
  blocking: str array,
  color: color,
  crossorigin: str,
  disabled: bool,
  fetchpriority: auto str,
  href: str,
  hreflang: str,
  imagesizes: array,
  imagesrcset: array,
  integrity: str,
  media: str,
  referrerpolicy: none str,
  rel: str array,
  sizes: array,
  type: str,
) → content
```

/2.59.1 `as` `str`

Potential destination for a preload request (for rel="preload" and rel="modulepreload").

Variant	Details
`"audio"`	
`"audioworklet"`	
`"document"`	
`"embed"`	
`"font"`	
`"frame"`	

Variant	Details
`"iframe"`	
`"image"`	
`"json"`	
`"manifest"`	
`"object"`	
`"paintworklet"`	
`"report"`	
`"script"`	
`"serviceworker"`	
`"sharedworker"`	
`"style"`	
`"track"`	
`"video"`	
`"webidentity"`	
`"worker"`	
`"xslt"`	

/2.59.2 **`blocking`** `str` or `array`

Whether the element is potentially render-blocking.

Variant	Details
`"blocking"`	

/2.59.3 **`color`** `color`

Color to use when customizing a site's icon (for rel="mask-icon").

/2.59.4 **`crossorigin`** `str`

How the element handles crossorigin requests.

Variant	Details
`"anonymous"`	
`"use-credentials"`	

/2.59.5 **`disabled`** `bool`

Whether the link is disabled.

/2.59.6 **`fetchpriority`** `auto` or `str`

Sets the priority for fetches initiated by the element.

Variant	Details
`"high"`	
`"low"`	

/2.59.7 **`href`** `str`

Address of the hyperlink.

/2.59.8 **`hreflang`** `str`

Language of the linked resource.

/2.59.9 **`imagesizes`** `array`

Image sizes for different page layouts (for rel="preload"). Expects an array of dictionaries with the keys `condition` (string) and `size` (length).

/2.59.10 **imagesrcset** `array`

Images to use in different situations, e.g., high-resolution displays, small monitors, etc. (for rel="preload"). Expects an array of dictionaries with the keys `src` (string) and `width` (integer) or `density` (float).

/2.59.11 **integrity** `str`

Integrity metadata used in Subresource Integrity checks.

/2.59.12 **media** `str`

Applicable media.

/2.59.13 **referrerpolicy** `none` or `str`

Referrer policy for fetches initiated by the element.

Variant	Details
`"no-referrer"`	
`"no-referrer-when-downgrade"`	
`"same-origin"`	
`"origin"`	
`"strict-origin"`	
`"origin-when-cross-origin"`	
`"strict-origin-when-cross-origin"`	
`"unsafe-url"`	

/2.59.14 **rel** `str` or `array`

Relationship between the document containing the hyperlink and the destination resource.

Variant	Details
`"alternate"`	
`"canonical"`	
`"author"`	
`"bookmark"`	
`"dns-prefetch"`	
`"expect"`	
`"external"`	
`"help"`	
`"icon"`	
`"manifest"`	
`"modulepreload"`	
`"license"`	
`"next"`	
`"nofollow"`	
`"noopener"`	
`"noreferrer"`	
`"opener"`	
`"pingback"`	
`"preconnect"`	
`"prefetch"`	
`"preload"`	
`"prev"`	
`"privacy-policy"`	
`"search"`	
`"stylesheet"`	
`"tag"`	

Variant	Details
`"terms-of-service"`	

/2.59.15 **sizes** `array`

Sizes of the icons (for rel="icon"). Expects an array of sizes. Each size is specified as an array of two integers (width and height).

/2.59.16 **type** `str`

Hint for the type of the referenced resource.

/2.60 **main**

Container for the dominant contents of the document.

```
html.main(content) → content
```

/2.60.1 **body** `content` *Positional*

The contents of the HTML element.

/2.61 **map**

Image map.

```
html.map(
  name: str,
  content,
) → content
```

/2.61.1 **name** `str`

Name of image map to reference from the usemap attribute.

/2.61.2 **body** `content` *Positional*

The contents of the HTML element.

/2.62 **mark**

Highlight.

```
html.mark(content) → content
```

/2.62.1 **body** `content` *Positional*

The contents of the HTML element.

/2.63 **menu**

Menu of commands.

```
html.menu(content) → content
```

/2.63.1 **body** `content` *Positional*

The contents of the HTML element.

/2.64 **meta**

Text metadata.

```
html.meta(
  charset: str,
  content: str,
  http-equiv: str,
```

```
  media: str,
  name: str,
) → content
```

/2.64.1 **charset** `str`

Character encoding declaration.

Variant	Details
`"utf-8"`	

/2.64.2 **content** `str`

Value of the element.

/2.64.3 **http-equiv** `str`

Pragma directive.

Variant	Details
`"content-type"`	
`"default-style"`	
`"refresh"`	
`"x-ua-compatible"`	
`"content-security-policy"`	

/2.64.4 **media** `str`

Applicable media.

/2.64.5 **name** `str`

Metadata name.

/2.65 **meter**

Gauge.

```
html.meter(
  high: float,
  low: float,
  max: float,
  min: float,
  optimum: float,
  value: float,
  content,
) → content
```

/2.65.1 **high** `float`

Low limit of high range.

/2.65.2 **low** `float`

High limit of low range.

/2.65.3 **max** `float`

Upper bound of range.

/2.65.4 **min** `float`

Lower bound of range.

/2.65.5 **optimum** `float`

Optimum value in gauge.

/2.65.6 **value** `float`

Current value of the element.

/2.65.7 **body** `content` *Positional*

The contents of the HTML element.

/2.66 **nav**

Section with navigational links.

```
html.nav(content) → content
```

/2.66.1 **body** `content` *Positional*

The contents of the HTML element.

/2.67 **noscript**

Fallback content for script.

```
html.noscript(content) → content
```

/2.67.1 **body** `content` *Positional*

The contents of the HTML element.

/2.68 **object**

Image, child navigable, or plugin.

```
html.object(
  data: str,
  form: str,
  height: int,
  name: str,
  type: str,
  width: int,
  content,
) → content
```

/2.68.1 **data** `str`

Address of the resource.

/2.68.2 **form** `str`

Associates the element with a form element.

/2.68.3 **height** `int`

Vertical dimension.

/2.68.4 **name** `str`

Name of content navigable.

Variant	Details
`"_blank"`	
`"_self"`	
`"_parent"`	
`"_top"`	

/2.68.5 **type** `str`

Type of embedded resource.

/2.68.6 **width** `int`

Horizontal dimension.

/2.68.7 **body** `content` *Positional*

The contents of the HTML element.

/2.69 **ol**

Ordered list.

```
html.ol(
  reversed: bool,
  start: int,
  type: str,
  content,
) → content
```

/2.69.1 **reversed** `bool`

Number the list backwards.

/2.69.2 **start** `int`

Starting value of the list.

/2.69.3 **type** `str`

Kind of list marker.

Variant	Details
`"1"`	
`"a"`	
`"A"`	
`"i"`	
`"I"`	

/2.69.4 **body** `content` *Positional*

The contents of the HTML element.

/2.70 **optgroup**

Group of options in a list box.

```
html.optgroup(
  disabled: bool,
  label: str,
  content,
) → content
```

/2.70.1 **disabled** `bool`

Whether the form control is disabled.

/2.70.2 **label** `str`

User-visible label.

/2.70.3 **body** `content` *Positional*

The contents of the HTML element.

/2.71 **option**

Option in a list box or combo box control.

```
html.option(
  disabled: bool,
  label: str,
  selected: bool,
  value: str,
  content,
) → content
```

/2.71.1 **disabled** `bool`

Whether the form control is disabled.

/2.71.2 **label** `str`

User-visible label.

/2.71.3 **selected** `bool`

Whether the option is selected by default.

/2.71.4 **value** `str`

Value to be used for form submission.

/2.71.5 **body** `content` *Positional*

The contents of the HTML element.

/2.72 **output**

Calculated output value.

```
html.output(
  for: str array,
  form: str,
  name: str,
  content,
) → content
```

/2.72.1 **for** `str` or `array`

Specifies controls from which the output was calculated.

/2.72.2 **form** `str`

Associates the element with a form element.

/2.72.3 **name** `str`

Name of the element to use for form submission and in the form.elements API.

/2.72.4 **body** `content` *Positional*

The contents of the HTML element.

/2.73 **p**

Paragraph.

```
html.p(content) → content
```

/2.73.1 **body** `content` *Positional*

The contents of the HTML element.

/2.74 **picture**

Image.

```
html.picture(content) → content
```

/2.74.1 **body** `content` *Positional*

The contents of the HTML element.

/2.75 **pre**

Block of preformatted text.

```
html.pre(content) → content
```

/2.75.1 **body** `content` *Positional*

The contents of the HTML element.

/2.76 **progress**

Progress bar.

```
html.progress(
  max: float,
  value: float,
  content,
) → content
```

/2.76.1 **max** `float`

Upper bound of range.

/2.76.2 **value** `float`

Current value of the element.

/2.76.3 **body** `content` *Positional*

The contents of the HTML element.

/2.77 **q**

Quotation.

```
html.q(
  cite: str,
  content,
) → content
```

/2.77.1 **cite** `str`

Link to the source of the quotation or more information about the edit.

/2.77.2 **body** `content` *Positional*

The contents of the HTML element.

/2.78 **rp**

Parenthesis for ruby annotation text.

```
html.rp(content) → content
```

/2.78.1 **body** `content` *Positional*

The contents of the HTML element.

/2.79 **rt**

Ruby annotation text.

```
html.rt(content) → content
```

/2.79.1 **body** `content` *Positional*

The contents of the HTML element.

/2.80 **ruby**

Ruby annotation(s).

```
html.ruby(content) → content
```

/2.80.1 **body** `content` *Positional*

The contents of the HTML element.

/2.81 **s**

Inaccurate text.

```
html.s(content) → content
```

/2.81.1 **body** `content` *Positional*

The contents of the HTML element.

/2.82 **samp**

Computer output.

```
html.samp(content) → content
```

/2.82.1 **body** `content` *Positional*

The contents of the HTML element.

/2.83 **script**

Embedded script.

```
html.script(
  async: bool,
  blocking: str array,
  crossorigin: str,
  defer: bool,
  fetchpriority: auto str,
  integrity: str,
  nomodule: bool,
  referrerpolicy: none str,
  src: str,
  type: str,
  str,
) → content
```

/2.83.1 **async** `bool`

Execute script when available, without blocking while fetching.

/2.83.2 **blocking** `str` or `array`

Whether the element is potentially render-blocking.

Variant	Details
`"blocking"`	

/2.83.3 **crossorigin** `str`

How the element handles crossorigin requests.

Variant	Details
`"anonymous"`	
`"use-credentials"`	

/2.83.4 **defer** `bool`

Defer script execution.

/2.83.5 **fetchpriority** `auto` or `str`

Sets the priority for fetches initiated by the element.

Variant	Details
`"high"`	
`"low"`	

/2.83.6 **integrity** `str`

Integrity metadata used in Subresource Integrity checks.

/2.83.7 **nomodule** `bool`

Prevents execution in user agents that support module scripts.

/2.83.8 **referrerpolicy** `none` or `str`

Referrer policy for fetches initiated by the element.

Variant	Details
`"no-referrer"`	
`"no-referrer-when-downgrade"`	
`"same-origin"`	
`"origin"`	
`"strict-origin"`	
`"origin-when-cross-origin"`	
`"strict-origin-when-cross-origin"`	
`"unsafe-url"`	

/2.83.9 **src** `str`

Address of the resource.

/2.83.10 **type** `str`

Type of script.

Variant	Details
`"module"`	

/2.83.11 **body** `str` *Positional*

The text content of the HTML element.

/2.84 **search**

Container for search controls.

```
html.search(content) → content
```

/2.84.1 **body** `content` *Positional*

The contents of the HTML element.

/2.85 **section**

Generic document or application section.

```
html.section(content) → content
```

/2.85.1 **body** `content` *Positional*

The contents of the HTML element.

/2.86 **select**

List box control.

```
html.select(
  autocomplete: str array,
  disabled: bool,
  form: str,
  multiple: bool,
  name: str,
  required: bool,
  size: int,
  content,
) → content
```

/2.86.1 **autocomplete** `str` or `array`

Hint for form autofill feature.

Variant	Details
`"shipping"`	
`"billing"`	
`"name"`	
`"honorific-prefix"`	
`"given-name"`	
`"additional-name"`	
`"family-name"`	
`"honorific-suffix"`	
`"nickname"`	
`"username"`	
`"new-password"`	
`"current-password"`	
`"one-time-code"`	
`"organization-title"`	
`"organization"`	
`"street-address"`	
`"address-line1"`	
`"address-line2"`	
`"address-line3"`	
`"address-level4"`	
`"address-level3"`	
`"address-level2"`	
`"address-level1"`	
`"country"`	
`"country-name"`	
`"postal-code"`	

Variant	Details
`"cc-name"`	
`"cc-given-name"`	
`"cc-additional-name"`	
`"cc-family-name"`	
`"cc-number"`	
`"cc-exp"`	
`"cc-exp-month"`	
`"cc-exp-year"`	
`"cc-csc"`	
`"cc-type"`	
`"transaction-currency"`	
`"transaction-amount"`	
`"language"`	
`"bday"`	
`"bday-day"`	
`"bday-month"`	
`"bday-year"`	
`"sex"`	
`"url"`	
`"photo"`	
`"home"`	
`"work"`	
`"mobile"`	
`"fax"`	
`"pager"`	
`"tel"`	
`"tel-country-code"`	
`"tel-national"`	
`"tel-area-code"`	
`"tel-local"`	
`"tel-local-prefix"`	
`"tel-local-suffix"`	
`"tel-extension"`	
`"email"`	
`"impp"`	

/2.86.2 **`disabled`** `bool`

Whether the form control is disabled.

/2.86.3 **`form`** `str`

Associates the element with a form element.

/2.86.4 **`multiple`** `bool`

Whether to allow multiple values.

/2.86.5 **`name`** `str`

Name of the element to use for form submission and in the form.elements API.

/2.86.6 **`required`** `bool`

Whether the control is required for form submission.

/2.86.7 **`size`** `int`

Size of the control.

/2.86.8 **body** `content` *Positional*

The contents of the HTML element.

/2.87 **slot**

Shadow tree slot.

```
html.slot(
  name: str,
  content,
) → content
```

/2.87.1 **name** `str`

Name of shadow tree slot.

/2.87.2 **body** `content` *Positional*

The contents of the HTML element.

/2.88 **small**

Side comment.

```
html.small(content) → content
```

/2.88.1 **body** `content` *Positional*

The contents of the HTML element.

/2.89 **source**

Image source for img or media source for video or audio.

```
html.source(
  height: int,
  media: str,
  sizes: array,
  src: str,
  srcset: array,
  type: str,
  width: int,
) → content
```

/2.89.1 **height** `int`

Vertical dimension.

/2.89.2 **media** `str`

Applicable media.

/2.89.3 **sizes** `array`

Image sizes for different page layouts. Expects an array of dictionaries with the keys `condition` (string) and `size` (length).

/2.89.4 **src** `str`

Address of the resource.

/2.89.5 **srcset** `array`

Images to use in different situations, e.g., high-resolution displays, small monitors, etc. Expects an array of dictionaries with the keys `src` (string) and `width` (integer) or `density` (float).

/2.89.6 **type** `str`

Type of embedded resource.

/2.89.7 **width** `int`

Horizontal dimension.

/2.90 **span**

Generic phrasing container.

```
html.span(content) → content
```

/2.90.1 **body** `content` *Positional*

The contents of the HTML element.

/2.91 **strong**

Importance.

```
html.strong(content) → content
```

/2.91.1 **body** `content` *Positional*

The contents of the HTML element.

/2.92 **style**

Embedded styling information.

```
html.style(
  blocking: str array,
  media: str,
  str,
) → content
```

/2.92.1 **blocking** `str` or `array`

Whether the element is potentially render-blocking.

Variant	Details
`"blocking"`	

/2.92.2 **media** `str`

Applicable media.

/2.92.3 **body** `str` *Positional*

The text content of the HTML element.

/2.93 **sub**

Subscript.

```
html.sub(content) → content
```

/2.93.1 **body** `content` *Positional*

The contents of the HTML element.

/2.94 **summary**

Caption for details.

```
html.summary(content) → content
```

/2.94.1 **body** `content` *Positional*

The contents of the HTML element.

/2.95 **sup**

Superscript.

```
html.sup(content) → content
```

/2.95.1 **body** `content` *Positional*

The contents of the HTML element.

/2.96 **table**

Table.

```
html.table(content) → content
```

/2.96.1 **body** `content` *Positional*

The contents of the HTML element.

/2.97 **tbody**

Group of rows in a table.

```
html.tbody(content) → content
```

/2.97.1 **body** `content` *Positional*

The contents of the HTML element.

/2.98 **td**

Table cell.

```
html.td(
  colspan: int,
  headers: str array,
  rowspan: int,
  content,
) → content
```

/2.98.1 **colspan** `int`

Number of columns that the cell is to span.

/2.98.2 **headers** `str` or `array`

The header cells for this cell.

/2.98.3 **rowspan** `int`

Number of rows that the cell is to span.

/2.98.4 **body** `content` *Positional*

The contents of the HTML element.

/2.99 **template**

Template.

```
html.template(
  shadowrootclonable: bool,
  shadowrootcustomelementregistry: bool,
  shadowrootdelegatesfocus: bool,
  shadowrootmode: str,
  shadowrootserializable: bool,
```

```
  content,
) → content
```

/2.99.1 **shadowrootclonable** `bool`

Sets clonable on a declarative shadow root.

/2.99.2 **shadowrootcustomelementregistry** `bool`

Enables declarative shadow roots to indicate they will use a custom element registry.

/2.99.3 **shadowrootdelegatesfocus** `bool`

Sets delegates focus on a declarative shadow root.

/2.99.4 **shadowrootmode** `str`

Enables streaming declarative shadow roots.

Variant	Details
`"open"`	
`"closed"`	

/2.99.5 **shadowrootserializable** `bool`

Sets serializable on a declarative shadow root.

/2.99.6 **body** `content` *Positional*

The contents of the HTML element.

/2.100 **textarea**

Multiline text controls.

```
html.textarea(
  autocomplete: str array,
  cols: int,
  dirname: str,
  disabled: bool,
  form: str,
  maxlength: int,
  minlength: int,
  name: str,
  placeholder: str,
  readonly: bool,
  required: bool,
  rows: int,
  wrap: str,
  content,
) → content
```

/2.100.1 **autocomplete** `str` or `array`

Hint for form autofill feature.

Variant	Details
`"shipping"`	
`"billing"`	
`"name"`	
`"honorific-prefix"`	
`"given-name"`	
`"additional-name"`	

Variant	Details
`"family-name"`	
`"honorific-suffix"`	
`"nickname"`	
`"username"`	
`"new-password"`	
`"current-password"`	
`"one-time-code"`	
`"organization-title"`	
`"organization"`	
`"street-address"`	
`"address-line1"`	
`"address-line2"`	
`"address-line3"`	
`"address-level4"`	
`"address-level3"`	
`"address-level2"`	
`"address-level1"`	
`"country"`	
`"country-name"`	
`"postal-code"`	
`"cc-name"`	
`"cc-given-name"`	
`"cc-additional-name"`	
`"cc-family-name"`	
`"cc-number"`	
`"cc-exp"`	
`"cc-exp-month"`	
`"cc-exp-year"`	
`"cc-csc"`	
`"cc-type"`	
`"transaction-currency"`	
`"transaction-amount"`	
`"language"`	
`"bday"`	
`"bday-day"`	
`"bday-month"`	
`"bday-year"`	
`"sex"`	
`"url"`	
`"photo"`	
`"home"`	
`"work"`	
`"mobile"`	
`"fax"`	
`"pager"`	
`"tel"`	
`"tel-country-code"`	
`"tel-national"`	
`"tel-area-code"`	
`"tel-local"`	

Variant	Details
`"tel-local-prefix"`	
`"tel-local-suffix"`	
`"tel-extension"`	
`"email"`	
`"impp"`	

/2.100.2 **`cols`** `int`

Maximum number of characters per line.

/2.100.3 **`dirname`** `str`

Name of form control to use for sending the element's directionality in form submission.

/2.100.4 **`disabled`** `bool`

Whether the form control is disabled.

/2.100.5 **`form`** `str`

Associates the element with a form element.

/2.100.6 **`maxlength`** `int`

Maximum length of value.

/2.100.7 **`minlength`** `int`

Minimum length of value.

/2.100.8 **`name`** `str`

Name of the element to use for form submission and in the form.elements API.

/2.100.9 **`placeholder`** `str`

User-visible label to be placed within the form control.

/2.100.10 **`readonly`** `bool`

Whether to allow the value to be edited by the user.

/2.100.11 **`required`** `bool`

Whether the control is required for form submission.

/2.100.12 **`rows`** `int`

Number of lines to show.

/2.100.13 **`wrap`** `str`

How the value of the form control is to be wrapped for form submission.

Variant	Details
`"soft"`	
`"hard"`	

/2.100.14 **`body`** `content` *Positional*

The contents of the HTML element.

/2.101 **`tfoot`**

Group of footer rows in a table.

```
html.tfoot(content) → content
```

/2.101.1 **`body`** `content` *Positional*

The contents of the HTML element.

/2.102 **th**

Table header cell.

```
html.th(
  abbr: str,
  colspan: int,
  headers: str array,
  rowspan: int,
  scope: str,
  content,
) → content
```

/2.102.1 **abbr** `str`

Alternative label to use for the header cell when referencing the cell in other contexts.

/2.102.2 **colspan** `int`

Number of columns that the cell is to span.

/2.102.3 **headers** `str` or `array`

The header cells for this cell.

/2.102.4 **rowspan** `int`

Number of rows that the cell is to span.

/2.102.5 **scope** `str`

Specifies which cells the header cell applies to.

Variant	Details
`"row"`	
`"col"`	
`"rowgroup"`	
`"colgroup"`	

/2.102.6 **body** `content` *Positional*

The contents of the HTML element.

/2.103 **thead**

Group of heading rows in a table.

```
html.thead(content) → content
```

/2.103.1 **body** `content` *Positional*

The contents of the HTML element.

/2.104 **time**

Machine-readable equivalent of date- or time-related data.

```
html.time(
  datetime: datetime duration,
  content,
) → content
```

/2.104.1 **datetime** `datetime` or `duration`

Machine-readable value.

/2.104.2 **body** `content` *Positional*

The contents of the HTML element.

/2.105 **title**

Document title.

```
html.title(content) → content
```

/2.105.1 **body** `content` *Positional*

The contents of the HTML element.

/2.106 **tr**

Table row.

```
html.tr(content) → content
```

/2.106.1 **body** `content` *Positional*

The contents of the HTML element.

/2.107 **track**

Timed text track.

```
html.track(
  default: bool,
  kind: str,
  label: str,
  src: str,
  srclang: str,
) → content
```

/2.107.1 **default** `bool`

Enable the track if no other text track is more suitable.

/2.107.2 **kind** `str`

The type of text track.

Variant	Details
`"subtitles"`	
`"captions"`	
`"descriptions"`	
`"chapters"`	
`"metadata"`	

/2.107.3 **label** `str`

User-visible label.

/2.107.4 **src** `str`

Address of the resource.

/2.107.5 **srclang** `str`

Language of the text track.

/2.108 **u**

Unarticulated annotation.

```
html.u(content) → content
```

/2.108.1 **body** `content` *Positional*

The contents of the HTML element.

/2.109 **ul**

List.

```
html.ul(content) → content
```

/2.109.1 **body** `content` *Positional*

The contents of the HTML element.

/2.110 **var**

Variable.

```
html.var(content) → content
```

/2.110.1 **body** `content` *Positional*

The contents of the HTML element.

/2.111 **video**

Video player.

```
html.video(
  autoplay: bool,
  controls: bool,
  crossorigin: str,
  height: int,
  loop: bool,
  muted: bool,
  playsinline: bool,
  poster: str,
  preload: none auto str,
  src: str,
  width: int,
  content,
) → content
```

/2.111.1 **autoplay** `bool`

Hint that the media resource can be started automatically when the page is loaded.

/2.111.2 **controls** `bool`

Show user agent controls.

/2.111.3 **crossorigin** `str`

How the element handles crossorigin requests.

Variant	Details
`"anonymous"`	
`"use-credentials"`	

/2.111.4 **height** `int`

Vertical dimension.

/2.111.5 **loop** `bool`

Whether to loop the media resource.

/2.111.6 **muted** `bool`

Whether to mute the media resource by default.

/2.111.7 **playsinline** `bool`

Encourage the user agent to display video content within the element's playback area.

/2.111.8 **poster** `str`

Poster frame to show prior to video playback.

/2.111.9 **preload** `none` or `auto` or `str`

Hints how much buffering the media resource will likely need.

Variant	Details
`"metadata"`	

/2.111.10 **src** `str`

Address of the resource.

/2.111.11 **width** `int`

Horizontal dimension.

/2.111.12 **body** `content` *Positional*

The contents of the HTML element.

/2.112 **wbr**

Line breaking opportunity.

```
html.wbr() → content
```

/3 Global Attributes

These parameters are common to all typed HTML functions. They are listed here once instead of explicitly on each element for readability.

/3.1 **accesskey** `str` or `array`

Keyboard shortcut to activate or focus element. Expects a single-codepoint string or an array thereof.

/3.2 **aria-activedescendant** `str`

Identifies the currently active element when DOM focus is on a composite widget, textbox, group, or application.

/3.3 **aria-atomic** `bool`

Indicates whether assistive technologies will present all, or only parts of, the changed region based on the change notifications defined by the aria-relevant attribute.

/3.4 **aria-autocomplete** `none` or `str`

Indicates whether inputting text could trigger display of one or more predictions of the user's intended value for an input and specifies how predictions would be presented if they are made.

Variant	Details
`"inline"`	When a user is providing input, text suggesting one way to complete the provided input may be dynamically inserted after the caret.

Variant	Details
`"list"`	When a user is providing input, an element containing a collection of values that could complete the provided input may be displayed.
`"both"`	When a user is providing input, an element containing a collection of values that could complete the provided input may be displayed. If displayed, one value in the collection is automatically selected, and the text needed to complete the automatically selected value appears after the caret in the input.

/3.5 **aria-busy** `bool`

Indicates an element is being modified and that assistive technologies MAY want to wait until the modifications are complete before exposing them to the user.

/3.6 **aria-checked** `bool` or `str`

Indicates the current “checked” state of checkboxes, radio buttons, and other widgets. See related aria-pressed and aria-selected.

Variant	Details
`"mixed"`	An intermediate value between true and false.

/3.7 **aria-colcount** `int`

Defines the total number of columns in a table, grid, or treegrid. See related aria-colindex.

/3.8 **aria-colindex** `int`

Defines an element’s column index or position with respect to the total number of columns within a table, grid, or treegrid. See related aria-colcount and aria-colspan.

/3.9 **aria-colspan** `int`

Defines the number of columns spanned by a cell or gridcell within a table, grid, or treegrid. See related aria-colindex and aria-rowspan.

/3.10 **aria-controls** `str` or `array`

Identifies the element (or elements) whose contents or presence are controlled by the current element. See related aria-owns.

/3.11 **aria-current** `bool` or `str`

Indicates the element that represents the current item within a container or set of related elements.

Variant	Details
`"page"`	Represents the current page within a set of pages.
`"step"`	Represents the current step within a process.
`"location"`	Represents the current location within an environment or context.
`"date"`	Represents the current date within a collection of dates.
`"time"`	Represents the current time within a set of times.

/3.12 **aria-describedby** `str` or `array`

Identifies the element (or elements) that describes the object. See related aria-labelledby.

/3.13 **aria-details** `str`

Identifies the element that provides a detailed, extended description for the object. See related aria-describedby.

/3.14 **aria-disabled** `bool`

Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable. See related aria-hidden and aria-readonly.

/3.15 **aria-errormessage** `str`

Identifies the element that provides an error message for the object. See related aria-invalid and aria-describedby.

/3.16 **aria-expanded** `none` or `bool`

Indicates whether the element, or another grouping element it controls, is currently expanded or collapsed.

/3.17 **aria-flowto** `str` or `array`

Identifies the next element (or elements) in an alternate reading order of content which, at the user's discretion, allows assistive technology to override the general default of reading in document source order.

/3.18 **aria-haspopup** `bool` or `str`

Indicates the availability and type of interactive popup element, such as menu or dialog, that can be triggered by an element.

Variant	Details
`"menu"`	Indicates the popup is a menu.
`"listbox"`	Indicates the popup is a listbox.
`"tree"`	Indicates the popup is a tree.
`"grid"`	Indicates the popup is a grid.
`"dialog"`	Indicates the popup is a dialog.

/3.19 **aria-hidden** `none` or `bool`

Indicates whether the element is exposed to an accessibility API. See related aria-disabled.

/3.20 **aria-invalid** `bool` or `str`

Indicates the entered value does not conform to the format expected by the application. See related aria-errormessage.

Variant	Details
`"grammar"`	A grammatical error was detected.
`"spelling"`	A spelling error was detected.

/3.21 **aria-keyshortcuts** `str`

Indicates keyboard shortcuts that an author has implemented to activate or give focus to an element.

/3.22 **aria-label** `str`

Defines a string value that labels the current element. See related aria-labelledby.

/3.23 **aria-labelledby** `str` or `array`

Identifies the element (or elements) that labels the current element. See related aria-describedby.

/3.24 **aria-level** `int`

Defines the hierarchical level of an element within a structure.

/3.25 **aria-live** `str`

Indicates that an element will be updated, and describes the types of updates the user agents, assistive technologies, and user can expect from the live region.

Variant	Details
`"assertive"`	Indicates that updates to the region have the highest priority and should be presented the user immediately.
`"off"`	Indicates that updates to the region should not be presented to the user unless the used is currently focused on that region.

Variant	Details
`"polite"`	Indicates that updates to the region should be presented at the next graceful opportunity, such as at the end of speaking the current sentence or when the user pauses typing.

/3.26 **aria-modal** `bool`

Indicates whether an element is modal when displayed.

/3.27 **aria-multiline** `bool`

Indicates whether a text box accepts multiple lines of input or only a single line.

/3.28 **aria-multiselectable** `bool`

Indicates that the user may select more than one item from the current selectable descendants.

/3.29 **aria-orientation** `str`

Indicates whether the element's orientation is horizontal, vertical, or unknown/ambiguous.

Variant	Details
`"horizontal"`	The element is oriented horizontally.
`"undefined"`	The element's orientation is unknown/ambiguous.
`"vertical"`	The element is oriented vertically.

/3.30 **aria-owns** `str` or `array`

Identifies an element (or elements) in order to define a visual, functional, or contextual parent/child relationship between DOM elements where the DOM hierarchy cannot be used to represent the relationship. See related aria-controls.

/3.31 **aria-placeholder** `str`

Defines a short hint (a word or short phrase) intended to aid the user with data entry when the control has no value. A hint could be a sample value or a brief description of the expected format.

/3.32 **aria-posinset** `int`

Defines an element's number or position in the current set of listitems or treeitems. Not required if all elements in the set are present in the DOM. See related aria-setsize.

/3.33 **aria-pressed** `bool` or `str`

Indicates the current "pressed" state of toggle buttons. See related aria-checked and aria-selected.

Variant	Details
`"mixed"`	An intermediate value between true and false.

/3.34 **aria-readonly** `bool`

Indicates that the element is not editable, but is otherwise operable. See related aria-disabled.

/3.35 **aria-relevant** `str` or `array`

Indicates what notifications the user agent will trigger when the accessibility tree within a live region is modified. See related aria-atomic.

Variant	Details
`"additions"`	Element nodes are added to the accessibility tree within the live region.
`"additions text"`	Equivalent to the combination of values, "additions text".
`"all"`	Equivalent to the combination of all values, "additions removals text".

Variant	Details
`"removals"`	Text content, a text alternative, or an element node within the live region is removed from the accessibility tree.
`"text"`	Text content or a text alternative is added to any descendant in the accessibility tree of the live region.

/3.36 **`aria-required`** `bool`

Indicates that user input is required on the element before a form may be submitted.

/3.37 **`aria-roledescription`** `str`

Defines a human-readable, author-localized description for the role of an element.

/3.38 **`aria-rowcount`** `int`

Defines the total number of rows in a table, grid, or treegrid. See related aria-rowindex.

/3.39 **`aria-rowindex`** `int`

Defines an element's row index or position with respect to the total number of rows within a table, grid, or treegrid. See related aria-rowcount and aria-rowspan.

/3.40 **`aria-rowspan`** `int`

Defines the number of rows spanned by a cell or gridcell within a table, grid, or treegrid. See related aria-rowindex and aria-colspan.

/3.41 **`aria-selected`** `none` or `bool`

Indicates the current "selected" state of various widgets. See related aria-checked and aria-pressed.

/3.42 **`aria-setsize`** `int`

Defines the number of items in the current set of listitems or treeitems. Not required if all elements in the set are present in the DOM. See related aria-posinset.

/3.43 **`aria-sort`** `none` or `str`

Indicates if items in a table or grid are sorted in ascending or descending order.

Variant	Details
`"ascending"`	Items are sorted in ascending order by this column.
`"descending"`	Items are sorted in descending order by this column.
`"other"`	A sort algorithm other than ascending or descending has been applied.

/3.44 **`aria-valuemax`** `float`

Defines the maximum allowed value for a range widget.

/3.45 **`aria-valuemin`** `float`

Defines the minimum allowed value for a range widget.

/3.46 **`aria-valuenow`** `float`

Defines the current value for a range widget. See related aria-valuetext.

/3.47 **`aria-valuetext`** `str`

Defines the human readable text alternative of aria-valuenow for a range widget.

/3.48 **`autocapitalize`** `none` or `bool` or `str`

Recommended autocapitalization behavior (for supported input methods).

Variant	Details
`"sentences"`	
`"words"`	
`"characters"`	

/3.49 **autocorrect** `bool`

Recommended autocorrection behavior (for supported input methods).

/3.50 **autofocus** `bool`

Automatically focus the element when the page is loaded.

/3.51 **class** `str` or `array`

Classes to which the element belongs.

/3.52 **contenteditable** `bool` or `str`

Whether the element is editable.

Variant	Details
`"plaintext-only"`	

/3.53 **dir** `auto` or `direction`

The text directionality of the element.

/3.54 **draggable** `bool`

Whether the element is draggable.

/3.55 **enterkeyhint** `str`

Hint for selecting an enter key action.

Variant	Details
`"enter"`	
`"done"`	
`"go"`	
`"next"`	
`"previous"`	
`"search"`	
`"send"`	

/3.56 **hidden** `bool` or `str`

Whether the element is relevant.

Variant	Details
`"until-found"`	

/3.57 **id** `str`

The element's ID.

/3.58 **inert** `bool`

Whether the element is inert.

/3.59 **inputmode** `none` or `str`

Hint for selecting an input modality.

Variant	Details
`"text"`	
`"tel"`	
`"email"`	
`"url"`	
`"numeric"`	
`"decimal"`	
`"search"`	

/3.60 **is** `str`

Creates a customized built-in element.

/3.61 **itemid** `str`

Global identifier for a microdata item.

/3.62 **itemprop** `str` or `array`

Property names of a microdata item.

/3.63 **itemref** `str` or `array`

Referenced elements.

/3.64 **itemscope** `bool`

Introduces a microdata item.

/3.65 **itemtype** `str` or `array`

Item types of a microdata item.

/3.66 **lang** `none` or `str`

Language of the element.

/3.67 **nonce** `str`

Cryptographic nonce used in Content Security Policy checks.

/3.68 **popover** `auto` or `str`

Makes the element a popover element.

Variant	Details
`"manual"`	

/3.69 **role** `none` or `str`

An ARIA role.

Variant	Details
`"alert"`	A type of live region with important, and usually time-sensitive, information. See related alertdialog and status.
`"alertdialog"`	A type of dialog that contains an alert message, where initial focus goes to an element within the dialog. See related alert and dialog.
`"application"`	A structure containing one or more focusable elements requiring user input, such as keyboard or gesture events, that do not follow a standard interaction pattern supported by a widget role.
`"article"`	A section of a page that consists of a composition that forms an independent part of a document, page, or site.

Variant	Details
`"banner"`	A region that contains mostly site-oriented content, rather than page-specific content.
`"button"`	An input that allows for user-triggered actions when clicked or pressed. See related link.
`"cell"`	A cell in a tabular container. See related gridcell.
`"checkbox"`	A checkable input that has three possible values: true, false, or mixed.
`"columnheader"`	A cell containing header information for a column.
`"combobox"`	A composite widget containing a single-line textbox and another element, such as a listbox or grid, that can dynamically pop up to help the user set the value of the textbox.
`"command"`	A form of widget that performs an action but does not receive input data.
`"complementary"`	A supporting section of the document, designed to be complementary to the main content at a similar level in the DOM hierarchy, but remains meaningful when separated from the main content.
`"composite"`	A widget that may contain navigable descendants or owned children.
`"contentinfo"`	A large perceivable region that contains information about the parent document.
`"definition"`	A definition of a term or concept. See related term.
`"dialog"`	A dialog is a descendant window of the primary window of a web application. For HTML pages, the primary application window is the entire web document, i.e., the body element.
`"directory"`	A list of references to members of a group, such as a static table of contents.
`"document"`	An element containing content that assistive technology users may want to browse in a reading mode.
`"feed"`	A scrollable list of articles where scrolling may cause articles to be added to or removed from either end of the list.
`"figure"`	A perceivable section of content that typically contains a graphical document, images, code snippets, or example text. The parts of a figure MAY be user-navigable.
`"form"`	A landmark region that contains a collection of items and objects that, as a whole, combine to create a form. See related search.
`"grid"`	A composite widget containing a collection of one or more rows with one or more cells where some or all cells in the grid are focusable by using methods of two-dimensional navigation, such as directional arrow keys.
`"gridcell"`	A cell in a grid or treegrid.
`"group"`	A set of user interface objects which are not intended to be included in a page summary or table of contents by assistive technologies.
`"heading"`	A heading for a section of the page.
`"img"`	A container for a collection of elements that form an image.
`"input"`	A generic type of widget that allows user input.
`"landmark"`	A perceivable section containing content that is relevant to a specific, author-specified purpose and sufficiently important that users will likely want to be able to navigate to the section easily and to have it listed in a summary of the page. Such a page summary could be generated dynamically by a user agent or assistive technology.
`"link"`	An interactive reference to an internal or external resource that, when activated, causes the user agent to navigate to that resource. See related button.
`"list"`	A section containing listitem elements. See related listbox.
`"listbox"`	A widget that allows the user to select one or more items from a list of choices. See related combobox and list.

Variant	Details
`"listitem"`	A single item in a list or directory.
`"log"`	A type of live region where new information is added in meaningful order and old information may disappear. See related marquee.
`"main"`	The main content of a document.
`"marquee"`	A type of live region where non-essential information changes frequently. See related log.
`"math"`	Content that represents a mathematical expression.
`"menu"`	A type of widget that offers a list of choices to the user.
`"menubar"`	A presentation of menu that usually remains visible and is usually presented horizontally.
`"menuitem"`	An option in a set of choices contained by a menu or menubar.
`"menuitemcheckbox"`	A menuitem with a checkable state whose possible values are true, false, or mixed.
`"menuitemradio"`	A checkable menuitem in a set of elements with the same role, only one of which can be checked at a time.
`"navigation"`	A collection of navigational elements (usually links) for navigating the document or related documents.
`"note"`	A section whose content is parenthetic or ancillary to the main content of the resource.
`"option"`	A selectable item in a select list.
`"presentation"`	An element whose implicit native role semantics will not be mapped to the accessibility API. See synonym none.
`"progressbar"`	An element that displays the progress status for tasks that take a long time.
`"radio"`	A checkable input in a group of elements with the same role, only one of which can be checked at a time.
`"radiogroup"`	A group of radio buttons.
`"range"`	An input representing a range of values that can be set by the user.
`"region"`	A perceivable section containing content that is relevant to a specific, author-specified purpose and sufficiently important that users will likely want to be able to navigate to the section easily and to have it listed in a summary of the page. Such a page summary could be generated dynamically by a user agent or assistive technology.
`"roletype"`	The base role from which all other roles in this taxonomy inherit.
`"row"`	A row of cells in a tabular container.
`"rowgroup"`	A structure containing one or more row elements in a tabular container.
`"rowheader"`	A cell containing header information for a row in a grid.
`"scrollbar"`	A graphical object that controls the scrolling of content within a viewing area, regardless of whether the content is fully displayed within the viewing area.
`"search"`	A landmark region that contains a collection of items and objects that, as a whole, combine to create a search facility. See related form and searchbox.
`"searchbox"`	A type of textbox intended for specifying search criteria. See related textbox and search.
`"section"`	A renderable structural containment unit in a document or application.
`"sectionhead"`	A structure that labels or summarizes the topic of its related section.
`"select"`	A form widget that allows the user to make selections from a set of choices.
`"separator"`	A divider that separates and distinguishes sections of content or groups of menuitems.
`"slider"`	A user input where the user selects a value from within a given range.
`"spinbutton"`	A form of range that expects the user to select from among discrete choices.

Variant	Details
`"status"`	A type of live region whose content is advisory information for the user but is not important enough to justify an alert, often but not necessarily presented as a status bar.
`"structure"`	A document structural element.
`"switch"`	A type of checkbox that represents on/off values, as opposed to checked/unchecked values. See related checkbox.
`"tab"`	A grouping label providing a mechanism for selecting the tab content that is to be rendered to the user.
`"table"`	A section containing data arranged in rows and columns. See related grid.
`"tablist"`	A list of tab elements, which are references to tabpanel elements.
`"tabpanel"`	A container for the resources associated with a tab, where each tab is contained in a tablist.
`"term"`	A word or phrase with a corresponding definition. See related definition.
`"textbox"`	A type of input that allows free-form text as its value.
`"timer"`	A type of live region containing a numerical counter which indicates an amount of elapsed time from a start point, or the time remaining until an end point.
`"toolbar"`	A collection of commonly used function buttons or controls represented in compact visual form.
`"tooltip"`	A contextual popup that displays a description for an element.
`"tree"`	A type of list that may contain sub-level nested groups that can be collapsed and expanded.
`"treegrid"`	A grid whose rows can be expanded and collapsed in the same manner as for a tree.
`"treeitem"`	An option item of a tree. This is an element within a tree that may be expanded or collapsed if it contains a sub-level group of tree item elements.
`"widget"`	An interactive component of a graphical user interface (GUI).
`"window"`	A browser or application window.

/3.70 **`slot`** `str`

The element's desired slot.

/3.71 **`spellcheck`** `bool`

Whether the element is to have its spelling and grammar checked.

/3.72 **`style`** `str`

Presentational and formatting instructions.

/3.73 **`tabindex`** `int`

Whether the element is focusable and sequentially focusable, and the relative order of the element for the purposes of sequential focus navigation.

/3.74 **`title`** `str`

Advisory information for the element.

/3.75 **`translate`** `bool`

Whether the element is to be translated when the page is localized.

/3.76 **`writingsuggestions`** `bool`

Whether the element can offer writing suggestions or not.

3.16 PNG

Instead of creating a PDF, Typst can also directly render pages to PNG raster graphics. PNGs are losslessly compressed images that can contain one page at a time. When exporting a multi-page document, Typst will emit multiple PNGs. PNGs are a good choice when you want to use Typst's output in an image editing software or when you can use none of Typst's other export formats.

In contrast to Typst's other export formats, PNGs are bound to a specific resolution. When exporting to PNG, you can configure the resolution as pixels per inch (PPI). If the medium you view the PNG on has a finer resolution than the PNG you exported, you will notice a loss of quality. Typst calculates the resolution of your PNGs based on each page's physical dimensions and the PPI. If you need guidance for choosing a PPI value, consider the following:

- A value of 300 or 600 is typical for desktop printing.
- Professional prints of detailed graphics can go up to 1200 PPI.
- If your document is only viewed at a distance, e.g. a poster, you may choose a smaller value than 300.
- If your document is viewed on screens, a typical PPI value for a smartphone is 400-500.

Because PNGs only contain a pixel raster, the text within cannot be extracted automatically (without OCR), for example by copy/paste or a screen reader. If you need the text to be accessible, export a PDF or HTML file instead.

PNGs can have transparent backgrounds. By default, Typst will output a PNG with an opaque white background. You can make the background transparent using `#set page(fill: none)`. Learn more on the page function's reference page[3.10.17/4.9].

3.16.1 Exporting as PNG

/1 Command Line

Pass `--format png` to the `compile` or `watch` subcommand or provide an output file name that ends with `.png`.

If your document has more than one page, Typst will create multiple image files. The output file name must then be a template string containing at least one of
- `{p}`, which will be replaced by the page number
- `{0p}`, which will be replaced by the zero-padded page number (so that all numbers have the same length)
- `{t}`, which will be replaced by the total number of pages

When exporting to PNG, you have the following configuration options:

- Which resolution to render at by specifying `--ppi` followed by a number of pixels per inch. The default is `144`.
- Which pages to export by specifying `--pages` followed by a comma-separated list of numbers or dash-separated number ranges. Ranges can be half-open. Example: `2,3,7-9,11-`.

/2 Web App

Click "File" > "Export as" > "PNG" or click the downwards-facing arrow next to the quick download button and select "Export as PNG". When exporting to PNG, you have the following configuration options:

- The resolution at which the pages should be rendered, as a number of pixels per inch. The default is `144`.
- Which pages to export. Valid options are "All pages", "Current page", and "Custom ranges". Custom ranges are a comma-separated list of numbers or dash-separated number ranges. Ranges can be half-open. Example: `2,3,7-9,11-`.

3.17 SVG

Instead of creating a PDF, Typst can also directly render pages to scalable vector graphics (SVGs), which are the preferred format for embedding vector graphics in web pages. Like PDF files, SVGs display your document exactly how you have laid it out in Typst. Likewise, they share the benefit of not being bound to a specific resolution. Hence, you can print or view SVG files on any device without incurring a loss of quality. (Note that font printing quality may be better with a PDF.) In contrast to a PDF, an SVG cannot contain multiple pages. When exporting a multi-page document, Typst will emit multiple SVGs.

SVGs can represent text in two ways: By embedding the text itself and rendering it with the fonts available on the viewer's computer or by embedding the shapes of each glyph in the font used to create the document. To ensure that the SVG file looks the same across all devices it is viewed on, Typst chooses the latter method. This means that the text in the SVG cannot be extracted automatically, for example by copy/paste or a screen reader. If you need the text to be accessible, export a PDF or HTML file instead.

SVGs can have transparent backgrounds. By default, Typst will output an SVG with an opaque white background. You can make the background transparent using `#set page(fill: none)`. Learn more on the page function's reference page[3.10.17/4.9].

3.17.1 Exporting as SVG

/1 Command Line

Pass `--format svg` to the `compile` or `watch` subcommand or provide an output file name that ends with `.svg`.

If your document has more than one page, Typst will create multiple image files. The output file name must then be a template string containing at least one of
- `{p}`, which will be replaced by the page number
- `{0p}`, which will be replaced by the zero-padded page number (so that all numbers have the same length)
- `{t}`, which will be replaced by the total number of pages

When exporting to SVG, you have the following configuration options:

- Which pages to export by specifying `--pages` followed by a comma-separated list of numbers or dash-separated number ranges. Ranges can be half-open. Example: `2,3,7-9,11-`.

/2 Web App

Click "File" > "Export as" > "SVG" or click the downwards-facing arrow next to the quick download button and select "Export as SVG". When exporting to SVG, you have the following configuration options:

- Which pages to export. Valid options are "All pages", "Current page", and "Custom ranges". Custom ranges are a comma-separated list of numbers or dash-separated number ranges. Ranges can be half-open. Example: `2,3,7-9,11-`.

3.18 Bundle

INFO

Bundle export is only available for experimentation behind a feature flag. Do not use this feature for production use cases. In the CLI, you can experiment with it by passing `--features bundle` or setting the `TYPST_FEATURES` environment variable to `bundle`. To use both the `bundle` and the `html` feature at the same time, specify both separated with a comma (i.e. `bundle,html`). In the web app, bundle export is not available at this time.

With Typst's bundle export, you can emit multiple output files from a single Typst project. Bundle output is useful for creating multi-page websites with HTML export, but it is not limited to HTML export. You can create bundles containing any combination of HTML pages[3.15], PDFs[3.14], PNGs[3.16], SVGs[3.17], and arbitrary assets[3.6.1].

3.18.1 Exporting as a bundle

/1 Command Line

Pass `--format bundle` to the `compile` or `watch` subcommand. Note that you must also pass `--features bundle` or set `TYPST_FEATURES=bundle` to enable this experimental export target.

When using `typst watch`, Typst will launch a live-reloading HTTP server serving your files. You can configure it as follows:

- Pass `--port` to change the port. (Defaults to the first free port in the range 3000-3005.)
- Pass `--no-reload` to disable injection of a live reload script into HTML pages. (The HTML that is written to disk isn't affected either way.) Non-HTML documents do not support live reload.
- Pass `--no-serve` to disable the server altogether.

/2 Web App

Not currently available.

3.18.2 Creating files

A bundle is a collection of files. Each of these bundle files falls into one of two categories: Document or asset. A `document`[3.6.6] takes content[3.5.8] that is exported with one of Typst's other export formats. Meanwhile, an `asset`[3.6.1] takes raw byte data[3.5.6] of your choice that will be written to disk as-is. Both elements take the desired output path as their first argument.

The example below shows a basic example of how bundle export could be used in practice:

```
#document("index.html", title: [Home])[
  #title()
  - #link(<blog>)[Go to blog]
]

#document("blog.html", title: [Blog])[
  #title()
  Welcome to my blog!

  ...

  This blog also exists as a
  #link(<blog-pdf>)[single PDF].
] <blog>

#document("blog.pdf", title: [Blog])[
  ...
] <blog-pdf>

#asset(
```

```
  "favicon.ico",
  read("images/favicon.ico", encoding: none),
)
```

In the example, we create two HTML documents: A home page and a blog. The home page links to the blog through a label link. Typst's built-in linking mechanism natively supports cross-document links[3.6.11/6] and resolves the correct relative paths for you. The bundle also contains a PDF version of the blog, which is linked from the HTML version. In practice, you could now share the content between the HTML and PDF version by storing it in a variable and using it in both. This is omitted here for brevity. Finally, the bundle contains an icon asset for the website. In this case, we're providing the asset's data by reading a file from disk. Alternatively, it's also possible to generate asset data from within Typst (e.g. via a function like `json.encode`[3.13.4/4.1]).

Documents and assets are normal elements, so you can use them with Typst's usual scripting, styling, and introspection mechanisms. For more details, refer to the `document`[3.6.6] and `asset`[3.6.1] documentation.

3.18.3 Introspection

Introspections always observe the full bundle rather than individual documents. For instance, querying for headings will give you all headings in all documents rather than the ones in the current document. Similarly, labels are global to the bundle—you can locate and link to labels in other documents[3.6.11/6]. Counters and states are likewise global.[A] In particular, if you enable things like heading numbering, the numbering will progress consecutively across the full bundle.

If you're using bundle export to build one conceptual work that is split up across multiple output files, this is typically what you want. If, however, you're using bundle export to export multiple conceptually separate works, you might want introspections to consider each document in isolation. Currently, you'll have to do this manually (e.g. by resetting counters, adjusting selectors, etc.) We plan to provide more tools[B] for managing the precise scope of introspection in the future.

[A] An exception to this forms the page counter, which is (naturally) per document.
[B] https://github.com/typst/typst/issues/7735#issuecomment-3908841853

4 Guides

Welcome to the Guides section! Here, you'll find helpful material for specific user groups or use cases. Please see the list below for the available guides. Feel free to propose other topics for guides!

List of Guides

4.1 Guide for LaTeX Users 565
4.2 Page Setup Guide 575
4.3 Table Guide 586
4.4 Accessibility Guide 612

4.1 Guide for LaTeX Users

This page is a good starting point if you have used LaTeX before and want to try out Typst. We will explore the main differences between these two systems from a user perspective. Although Typst is not built upon LaTeX and has a different syntax, you will learn how to use your LaTeX skills to get a head start.

Just like LaTeX, Typst is a markup-based typesetting system: You compose your document in a text file and mark it up with commands and other syntax. Then, you use a compiler to typeset the source file into a PDF. However, Typst also differs from LaTeX in several aspects: For one, Typst uses more dedicated syntax (like you may know from Markdown) for common tasks. Typst's commands are also more principled: They all work the same, so unlike in LaTeX, you just need to understand a few general concepts instead of learning different conventions for each package. Moreover Typst compiles faster than LaTeX: Compilation usually takes milliseconds, not seconds, so the web app and the compiler can both provide instant previews.

In the following, we will cover some of the most common questions a user switching from LaTeX will have when composing a document in Typst. If you prefer a step-by-step introduction to Typst, check out our tutorial[2].

4.1.1 Installation

You have two ways to use Typst: In our web app[A] or by installing the compiler[B] on your computer. When you use the web app, we provide a batteries-included collaborative editor and run Typst in your browser, no installation required.

If you choose to use Typst on your computer instead, you can download the compiler as a single, small binary which any user can run, no root privileges required. Unlike popular LaTeX distributions such as TeX Live, packages are downloaded when you first use them and then cached locally, keeping your Typst installation lean. You can use your own editor and decide where to store your files with the local compiler.

4.1.2 How do I create a new, empty document?

That's easy. You just create a new, empty text file (the file extension is `.typ`). No boilerplate is needed to get started. Simply start by writing your text. It will be set on an empty A4-sized page. If you are using the web app, click "+ Empty document" to create a new project with a file and enter the editor. Paragraph breaks[3.6.16] work just as they do in LaTeX, just use a blank line.

```
Hey there!

Here are two paragraphs. The
output is shown to the right.
```

Hey there!

Here are two paragraphs. The output is shown to the right.

If you want to start from an preexisting LaTeX document instead, you can use Pandoc[C] to convert your source code to Typst markup. This conversion is also built into our web app, so you can upload your `.tex` file to start your project in Typst.

4.1.3 How do I create section headings, emphasis, …?

LaTeX uses the command `\section` to create a section heading. Nested headings are indicated with `\subsection`, `\subsubsection`, etc. Depending on your document class, there is also `\part` or `\chapter`.

In Typst, headings[3.6.10] are less verbose: You prefix the line with the heading on it with an equals sign and a space to get a first-order heading: `= Introduction`. If you need a second-order heading, you use two equals signs: `== In this paper`. You can nest headings as deeply as you'd like by adding more equals signs.

Emphasis (usually rendered as italic text) is expressed by enclosing text in `_underscores_` and strong emphasis (usually rendered in boldface) by using `*stars*` instead.

[A] https://typst.app/signup/
[B] https://github.com/typst/typst/releases
[C] https://pandoc.org

Here is a list of common markup commands used in LaTeX and their Typst equivalents. You can also check out the full syntax cheat sheet[3.1].

Element	LaTeX	Typst	See
Strong emphasis	`\textbf{strong}`	`*strong*`	`strong`[3.6.19]
Emphasis	`\emph{emphasis}`	`_emphasis_`	`emph`[3.6.7]
Link	`\url{https://typst.app}`	`https://typst.app/`	`link`[3.6.11]
Label	`\label{intro}`	`<intro>`	`label`[3.5.17]
Reference	`\ref{intro}`	`@intro`	`ref`[3.6.18]
Citation	`\cite{humphrey97}`	`@humphrey97`	`cite`[3.6.4]
Monospace (typewriter)	`\texttt{mono}`	`text` or `mono` functions	`text`[3.7.12], `mono`[3.8.26/1.4]
Code	`lstlisting` environment	`` `print(f"{x}")` ``	`raw`[3.7.6]
Verbatim	`verbatim` environment	`` `#typst-code()` ``	`raw`[3.7.6]
Bullet list	`itemize` environment	`- List`	`list`[3.6.3]
Numbered list	`enumerate` environment	`+ List`	`enum`[3.6.12]
Term list	`description` environment	`/ Term: List`	`terms`[3.6.21]
Figure	`figure` environment	`figure` function	`figure`[3.6.8]
Table	`table` environment	`table` function	`table`[3.6.20]
Equation	`$x$`, `align` / `equation` environments	`$x$`, `$ x = y $`	`equation`[3.8.15]

Lists[3.6.3] do not rely on environments in Typst. Instead, they have lightweight syntax like headings. To create an unordered list (`itemize`), prefix each line of an item with a hyphen:

````
To write this list in Typst...

```latex
\begin{itemize}
  \item Fast
  \item Flexible
  \item Intuitive
\end{itemize}
```

...just type this:

- Fast
- Flexible
- Intuitive
````

To write this list in Typst...

```latex
\begin{itemize}
  \item Fast
  \item Flexible
  \item Intuitive
\end{itemize}
```

...just type this:

- Fast
- Flexible
- Intuitive

Nesting lists works just by using proper indentation. Adding a blank line in between items results in a more widely[3.6.3/3.1] spaced list.

To get a numbered list[3.6.12] (`enumerate`) instead, use a `+` instead of the hyphen. For a term list[3.6.21] (`description`), write `/ Term: Description` instead.

Note that the `raw` function[3.7.6] and syntax (e.g. `` `raw` ``) only work for verbatim (unformatted) text. If you require formatting, you can use the `text` function[3.7.12] with a monospace font instead, like in the example below:

```
#text(
  font: "DejaVu Sans Mono",
  size: 0.8em,
)[monospace *bold*]
```

monospace **bold**

4.1.4 How do I use a command?

LaTeX heavily relies on commands (prefixed by backslashes). It uses these *macros* to affect the typesetting process and to insert and manipulate content. Some commands accept arguments, which are most frequently enclosed in curly braces: `\cite{rasmus}`.

Typst differentiates between markup mode and code mode[3.3.2]. The default is markup mode, where you compose text and apply syntactic constructs such as `*stars for bold text*`. Code mode, on the other hand, parallels programming languages like Python, providing the option to input and execute segments of code.

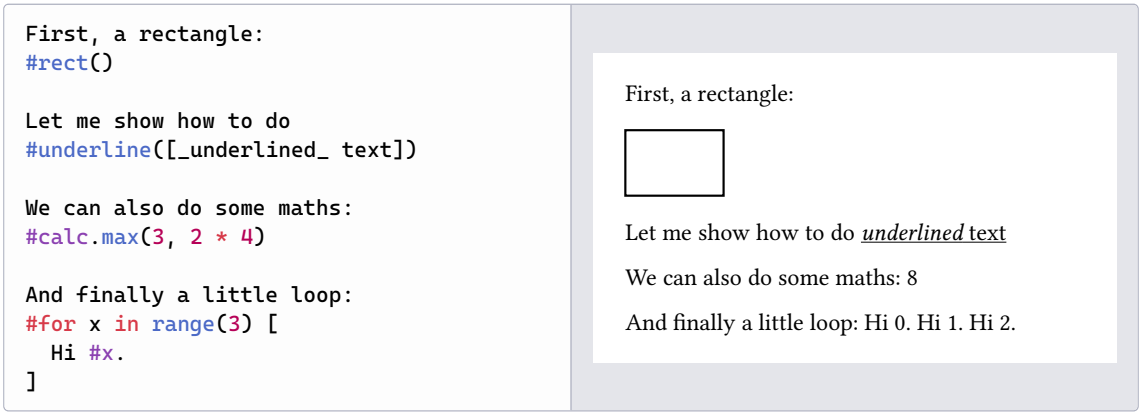
Within Typst's markup, you can switch to code mode for a single command (or rather, *expression*) using a hash (#). This is how you call functions to, for example, split your project into different files[3.3.8] or render text based on some condition[3.3.4]. Within code mode, it is possible to include normal markup *content*[3.5.8] by using square brackets. Within code mode, this content is treated just as any other normal value for a variable.

```
First, a rectangle:
#rect()

Let me show how to do
#underline([_underlined_ text])

We can also do some maths:
#calc.max(3, 2 * 4)

And finally a little loop:
#for x in range(3) [
  Hi #x.
]
```

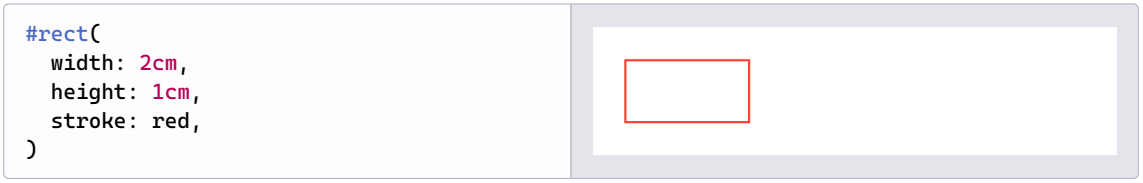


A function call always involves the name of the function (`rect`[3.11.10], `underline`[3.7.13], `calc.max`[3.5.7/1.34], `range`[3.5.2/3.13]) followed by parentheses (as opposed to LaTeX where the square brackets and curly braces are optional if the macro requires no arguments). The expected list of arguments passed within those parentheses depends on the concrete function and is specified in the reference[3].

/1 Arguments

A function can have multiple arguments. Some arguments are positional, i.e., you just provide the value: The function `#lower("SCREAM")` returns its argument in all-lowercase. Many functions use named arguments instead of positional arguments to increase legibility. For example, the dimensions and stroke of a rectangle are defined with named arguments:

```
#rect(
  width: 2cm,
  height: 1cm,
  stroke: red,
)
```



You specify a named argument by first entering its name (above, it's `width`, `height`, and `stroke`), then a colon, followed by the value (`2cm`, `1cm`, `red`). You can find the available named arguments in the reference page[3] for each function or in the autocomplete panel when typing. Named arguments are similar to how some LaTeX environments are configured, for example, you would type `\begin{enumerate}[label={\alph*)}]` to start a list with the labels `a)`, `b)`, and so on.

Often, you want to provide some content[3.5.8] to a function. For example, the LaTeX command `\underline{Alternative A}` would translate to `#underline([Alternative A])` in Typst. The square brackets indicate that a value is content[3.5.8]. Within these brackets, you can use normal markup. However, that's a lot of parentheses for a pretty simple construct. This is why you can also move trailing content arguments after the parentheses (and omit the parentheses if they would end up empty).

```
Typst is an #underline[alternative]
to LaTeX.

#rect(fill: aqua)[Get started here!]
```

Typst is an alternative to LaTeX.

Get started here!

/2 Data types

You likely already noticed that the arguments have distinctive data types. Typst supports many data types[3.5.31]. Below, there is a table with some of the most important ones and how to write them. In order to specify values of any of these types, you have to be in code mode!

Data type	Example
Content[3.5.8]	`[*fast* typesetting]`
String[3.5.27]	`"Pietro S. Author"`
Integer[3.5.16]	`23`
Floating point number[3.5.14]	`1.459`
Absolute length[3.10.13]	`12pt`, `5in`, `0.3cm`, …
Relative length[3.10.20]	`65%`

The difference between content and string is that content can contain markup, including function calls, while a string really is just a plain sequence of characters.

Typst provides control flow constructs[3.3.4] and operators[3.3.10] such as + for adding things or == for checking equality between two variables.

You can also store values, including functions, in your own variables[3.3.3]. This can be useful to perform computations on them, create reusable automations, or reference a value multiple times. The variable binding is accomplished with the let keyword, which works similar to `\newcommand`:

```
// Store the integer `5`.
#let five = 5

// Define a function that
// increments a value.
#let inc(i) = i + 1

// Reference the variables.
I have #five fingers.

If I had one more, I'd have
#inc(five) fingers. Whoa!
```

I have 5 fingers.

If I had one more, I'd have 6 fingers. Whoa!

/3 Commands to affect the remaining document

In LaTeX, some commands like `\textbf{bold text}` receive an argument in curly braces and only affect that argument. Other commands such as `\bfseries bold text` act as switches (LaTeX calls this a declaration), altering the appearance of all subsequent content within the document or current scope.

In Typst, the same function can be used both to affect the appearance for the remainder of the document, a block (or scope), or just its arguments. For example, `#text(weight: "bold")[bold text]` will only embolden its argument, while `#set text(weight: "bold")` will embolden any text until the end of the current block, or the end of the document, if there is none. The effects of a function are immediately obvious based on whether it is used in a call or a set rule.[3.2.1]

```
I am starting out with small text.

#set text(14pt)

This is a bit #text(18pt)[larger,]
don't you think?
```

I am starting out with small text.

This is a bit larger, don't you think?

Set rules may appear anywhere in the document. They can be thought of as default argument values of their respective function:

```
#set enum(numbering: "I.")

Good results can only be obtained by
+ following best practices
+ being aware of current results
  of other researchers
+ checking the data for biases
```

Good results can only be obtained by

I. following best practices
II. being aware of current results of other researchers
III. checking the data for biases

The `+` is syntactic sugar (think of it as an abbreviation) for a call to the `enum`[3.6.12] function, to which we apply a set rule above. Most syntax is linked to a function in this way.[3.1] If you need to style an element beyond what its arguments enable, you can completely redefine its appearance with a show rule[3.2.2] (somewhat comparable to `\renewcommand`).

You can achieve the effects of LaTeX commands like `\textbf`, `\textsf`, `\rmfamily`, `\mdseries`, and `\itshape` with the `font`[3.7.12/2.1], `style`[3.7.12/2.3], and `weight`[3.7.12/2.4] arguments of the `text` function. The text function can be used in a set rule (declaration style) or with a content argument. To replace `\textsc`, you can use the `smallcaps`[3.7.7] function, which renders its content argument as smallcaps. Should you want to use it declaration style (like `\scshape`), you can use an *everything* show rule[3.2.2] that applies the function to the rest of the scope:

```
#show: smallcaps

Boisterous Accusations
```

Boisterous Accusations

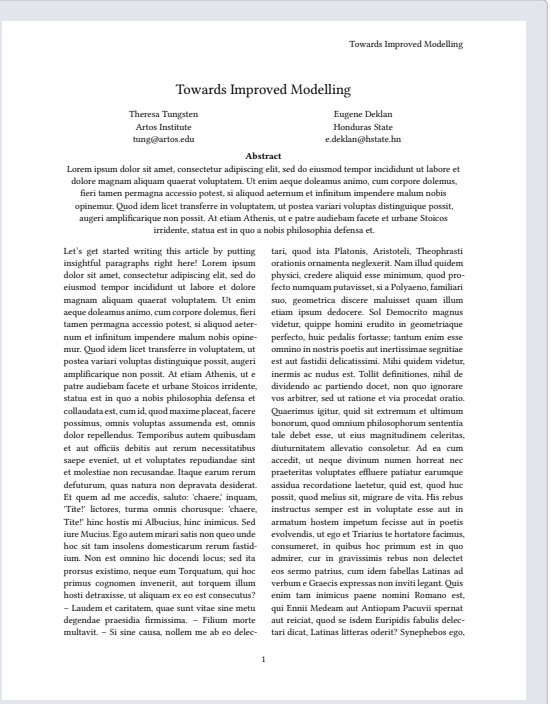
4.1.5 How do I load a document class?

In LaTeX, you start your main `.tex` file with the `\documentclass{article}` command to define how your document is supposed to look. In that command, you may have replaced `article` with another value such as `report` and `amsart` to select a different look.

When using Typst, you style your documents with functions[3.5.15]. Typically, you use a template that provides a function that styles your whole document. First, you import the function from a template file. Then, you apply it to your whole document. This is accomplished with a show rule[3.2.2] that wraps the following document in a given function. The following example illustrates how it works:

```
#import "conf.typ": conf
#show: conf.with(
  title: [
    Towards Improved Modelling
  ],
  authors: (
    (
      name: "Theresa Tungsten",
      affiliation: "Artos Institute",
      email: "tung@artos.edu",
    ),
    (
      name: "Eugene Deklan",
      affiliation: "Honduras State",
      email: "e.deklan@hstate.hn",
    ),
  ),
  abstract: lorem(80),
)

Let's get started writing this
article by putting insightful
paragraphs right here!
```



The `import`[3.3.8] statement makes functions[3.5.15] (and other definitions) from another file available. In this example, it imports the `conf` function from the `conf.typ` file. This function formats a document as a conference article. We use a show rule to apply it to the document and also configure some metadata of the article. After applying the show rule, we can start writing our article right away!

You can also use templates from Typst Universe (which is Typst's equivalent of CTAN) using an import statement like this: `#import "@preview/elsearticle:0.2.1": elsearticle`. Check the documentation of an individual template to learn the name of its template function. Templates and packages from Typst Universe are automatically downloaded when you first use them.

In the web app, you can choose to create a project from a template on Typst Universe or even create your own using the template wizard. Locally, you can use the `typst init` CLI to create a new project from a template. Check out the list of templates[A] published on Typst Universe. You can also take a look at the `awesome-typst` repository[B] to find community templates that aren't available through Universe.

You can also create your own, custom templates.[2.4] They are shorter and more readable than the corresponding LaTeX `.sty` files by orders of magnitude, so give it a try!

INFO

Functions are Typst's "commands" and can transform their arguments to an output value, including document *content*. Functions are "pure", which means that they cannot have any effects beyond creating an output value / output content. This is in stark contrast to LaTeX macros that can have arbitrary effects on your document.

To let a function style your whole document, the show rule processes everything that comes after it and calls the function specified after the colon with the result as an argument. The `.with` part is a *method* that takes the `conf` function and pre-configures some of its arguments before passing it on to the show rule.

4.1.6 How do I load packages?

Typst is "batteries included," so the equivalent of many popular LaTeX packages is built right-in. Below, we compiled a table with frequently loaded packages and their corresponding Typst functions.

[A] https://typst.app/universe/search/?kind=templates
[B] https://github.com/qjcg/awesome-typst

LaTeX Package	Typst Alternative
graphicx, svg	`image`[3.11.7] function
tabularx, tabularray	`table`[3.6.20], `grid`[3.10.10] functions
fontenc, inputenc, unicode-math	Just start writing!
babel, polyglossia	`text`[3.7.12/2.16] function: `#set text(lang: "zh")`
amsmath	Math mode[3.8]
amsfonts, amssymb	`sym`[3.9] module and syntax[3.1.3]
geometry, fancyhdr	`page`[3.10.17] function
xcolor	`text`[3.7.12/2.7] function: `#set text(fill: rgb("#0178A4"))`
hyperref	`link`[3.6.11] function
bibtex, biblatex, natbib	`cite`[3.6.4], `bibliography`[3.6.2] functions
lstlisting, minted	`raw`[3.7.6] function and syntax
parskip	`block`[3.10.4/2.9] and `par`[3.6.15/4.6] functions
csquotes	Set the `text`[3.7.12/2.16] language and type " or '
caption	`figure`[3.6.8] function
enumitem	`list`[3.6.3], `enum`[3.6.12], `terms`[3.6.21] functions
nicefrac	`frac.style`[3.8.16/3.3] property

Although *many* things are built-in, not everything can be. That's why Typst has its own package ecosystem[A] where the community share its creations and automations. Let's take, for instance, the *CeTZ* package: This package allows you to create complex drawings and plots. To use CeTZ in your document, you can just write:

```
#import "@preview/cetz:0.4.1"
```

(The `@preview` is a *namespace* that is used while the package manager is still in its early and experimental state. It will be replaced in the future.)

Aside from the official package hub, you might also want to check out the awesome-typst repository[B], which compiles a curated list of resources created for Typst.

If you need to load functions and variables from another file within your project, for example to use a template, you can use the same `import`[3.3.8] statement with a file name rather than a package specification. To instead include the textual content of another file, you can use an `include`[3.3.8] statement. It will retrieve the content of the specified file and put it in your document.

4.1.7 How do I input maths?

To enter math mode in Typst, just enclose your equation in dollar signs. You can enter display mode by adding spaces or newlines between the equation's contents and its enclosing dollar signs.

```
The sum of the numbers from
$1$ to $n$ is:

$ sum_(k=1)^n k = (n(n+1))/2 $
```

The sum of the numbers from 1 to n is:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Math mode[3.8] works differently than regular markup or code mode. Numbers and single characters are displayed verbatim, while multiple consecutive (non-number) characters will be interpreted as Typst variables.

Typst pre-defines a lot of useful variables in math mode. All Greek (`alpha`, `beta`, ...) and some Hebrew letters (`aleph`, `beth`, ...) are available through their name. Some symbols are additionally available through shorthands, such as `<=`, `>=`, and `->`.

[A] https://typst.app/universe
[B] https://github.com/qjcg/awesome-typst

Refer to the symbol pages[3.9] for a full list of the symbols. If a symbol is missing, you can also access it through a Unicode escape sequence[3.1.6].

Alternate and related forms of symbols can often be selected by appending a modifier[3.5.28] after a period. For example, `arrow.l.squiggly` inserts a squiggly left-pointing arrow. If you want to insert multiletter text in your expression instead, enclose it in double quotes:

```
$ delta "if" x <= 5 $
```

$$\delta \text{ if } x \leq 5$$

In Typst, delimiters will scale automatically for their expressions, just as if `\left` and `\right` commands were implicitly inserted in LaTeX. You can customize delimiter behaviour using the `lr` function[3.8.17/2.1]. To prevent a pair of delimiters from scaling, you can escape them with backslashes.

Typst will automatically set terms around a slash `/` as a fraction while honoring operator precedence. All round parentheses not made redundant by the fraction will appear in the output. Fractions are typeset vertically unless customized with `frac.style`[3.8.16/3.3]. You can also produce a slash as is by escaping it with a backslash (`\/`).

```
$ f(x) = (x + 1) / x $
```

$$f(x) = \frac{x + 1}{x}$$

Sub- and superscripts[3.8.10/3.1] work similarly in Typst and LaTeX. `$x^2$` will produce a superscript, `$x_2$` yields a subscript. If you want to include more than one value in a sub- or superscript, enclose their contents in parentheses: `$x_(a -> epsilon)$`.

Since variables in math mode do not need to be prepended with a `#` (or a `\` like in LaTeX), you can also call functions without these special characters:

```
$ f(x, y) := cases(
  1 "if" (x dot y)/2 <= 0,
  2 "if" x "is even",
  3 "if" x in NN,
  4 "else",
) $
```

$$f(x, y) := \begin{cases} 1 \text{ if } \frac{x \cdot y}{2} \leq 0 \\ 2 \text{ if } x \text{ is even} \\ 3 \text{ if } x \in \mathbb{N} \\ 4 \text{ else} \end{cases}$$

The above example uses the `cases` function[3.8.13] to describe f. Within the cases function, arguments are delimited using commas and the arguments are also interpreted as math. If you need to interpret arguments as Typst values instead, prefix them with a `#`:

```
$ (a + b)^2
  = a^2
  + text(fill: #maroon, 2 a b)
  + b^2 $
```

$$(a + b)^2 = a^2 + \text{2 a b} + b^2$$

You can use all Typst functions within math mode and insert any content. If you want them to work normally, with code mode in the argument list, you can prefix their call with a `#`. Nobody can stop you from using rectangles or emoji as your variables anymore:

```
$ sum^10_(🤓=1)
  #rect(width: 4mm, height: 2mm)/🤓
  = 🧠 maltese $
```

$$\sum_{🤓=1}^{10} \frac{\square}{🤓} = 🧠✠$$

If you'd like to enter your mathematical symbols directly as Unicode, that is possible, too!

Math calls can have two-dimensional argument lists using `;` as a delimiter. The most common use for this is the `mat` function[3.8.18] that creates matrices:

```
$ mat(
  1, 2, ..., 10;
  2, 2, ..., 10;
  dots.v, dots.v, dots.down, dots.v;
  10, 10, ..., 10;
) $
```

$$\begin{pmatrix} 1 & 2 & \dots & 10 \\ 2 & 2 & \dots & 10 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix}$$

4.1.8 How do I get the "LaTeX look?"

Papers set in LaTeX have an unmistakeable look. This is mostly due to their font, Computer Modern, justification, narrow line spacing, and wide margins.

The example below
- sets wide margins[3.10.17/4.5]
- enables justification[3.6.15/4.3], tighter lines[3.6.15/4.1] and first-line-indent[3.6.15/4.6]
- sets the font[3.7.12/2.1] to "New Computer Modern", an OpenType derivative of Computer Modern for both text and code blocks[3.7.6]
- decreases the font weight[3.7.12/2.4] in math mode
- disables paragraph spacing[3.10.4/2.9]
- increases spacing[3.10.4/2.9] around headings[3.6.10]

```
#set page(margin: 1.75in)
#set par(leading: 0.55em, spacing: 0.55em, first-line-indent: 1.8em, justify: true)
#set text(font: "New Computer Modern")
#show raw: set text(font: "New Computer Modern Mono")
#show math.equation: set text(weight: "regular")
#show heading: set block(above: 1.4em, below: 1em)
```

This should be a good starting point! If you want to go further, why not create a reusable template?

4.1.9 Bibliographies

Typst includes a fully-featured bibliography system that is compatible with BibTeX files. You can continue to use your `.bib` literature libraries by loading them with the `bibliography`[3.6.2] function. Another possibility is to use Typst's YAML-based native format[A].

Typst uses the Citation Style Language to define and process citation and bibliography styles. You can compare CSL files to BibLaTeX's `.bbx` files. The compiler already includes over 80 citation styles[3.6.2/4.4], but you can use any CSL-compliant style from the CSL repository[B] or write your own.

You can cite an entry in your bibliography or reference a label in your document with the same syntax: `@key` (this would reference an entry called `key`). Alternatively, you can use the `cite`[3.6.4] function.

Alternative forms for your citation, such as year only and citations for natural use in prose (cf. `\citet` and `\textcite`) are available with `#cite(<key>, form: "prose")`[3.6.4/3.3].

You can find more information on the documentation page of the `bibliography`[3.6.2] function.

4.1.10 What limitations does Typst currently have compared to LaTeX?

Although Typst can be a LaTeX replacement for many today, there are still features that Typst does not (yet) support. Here is a list of them which, where applicable, contains possible workarounds.

- **Well-established plotting ecosystem.** LaTeX users often create elaborate charts along with their documents in PGF/TikZ. The Typst ecosystem does not yet offer the same breadth of available options, but the ecosystem around the `cetz` package[C] is catching up quickly.

[A] https://github.com/typst/hayagriva/blob/main/docs/file-format.md
[B] https://github.com/citation-style-language/styles

– **Change page margins without a pagebreak.** In LaTeX, margins can always be adjusted, even without a pagebreak. To change margins in Typst, you use the page function[3.10.17] which will force a page break. If you just want a few paragraphs to stretch into the margins, then reverting to the old margins, you can use the pad function[3.10.16] with negative padding.

[C]https://typst.app/universe/package/cetz

4.2 Page Setup Guide

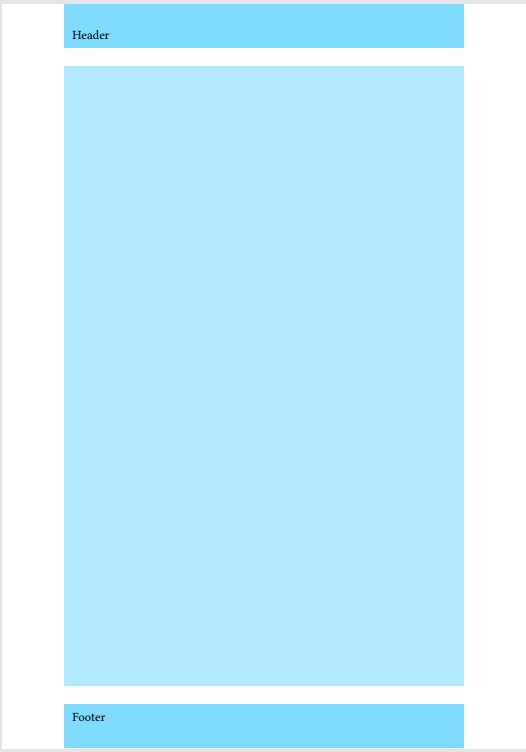
Your page setup is a big part of the first impression your document gives. Line lengths, margins, and columns influence appearance[A] and legibility[B] while the right headers and footers will help your reader easily navigate your document. This guide will help you to customize pages, margins, headers, footers, and page numbers so that they are the right fit for your content and you can get started with writing.

In Typst, each page has a width, a height, and margins on all four sides. The top and bottom margins may contain a header and footer. The set rule of the page[3.10.17] element is where you control all of the page setup. If you make changes with this set rule, Typst will ensure that there is a new and conforming empty page afterward, so it may insert a page break. Therefore, it is best to specify your page[3.10.17] set rule at the start of your document or in your template.

```
#set rect(
  width: 100%,
  height: 100%,
  inset: 4pt,
)

#set page(
  paper: "iso-b7",
  header: rect(fill: aqua)[Header],
  footer: rect(fill: aqua)[Footer],
  number-align: center,
)

#rect(fill: aqua.lighten(40%))
```



This example visualizes the dimensions for page content, headers, and footers. The page content is the page size (ISO B7) minus each side's default margin. In the top and the bottom margin, there are stroked rectangles visualizing the header and footer. They do not touch the main content, instead, they are offset by 30% of the respective margin. You can control this offset by specifying the `header-ascent`[3.10.17/4.14] and `footer-descent`[3.10.17/4.16] arguments.

Below, the guide will go more into detail on how to accomplish common page setup requirements with examples.

4.2.1 Customize page size and margins

Typst's default page size is A4 paper. Depending on your region and your use case, you will want to change this. You can do this by using the page[3.10.17] set rule and passing it a string argument to use a common page size. Options include the complete ISO 216 series (e.g. `"a4"` and `"iso-c2"`), customary US formats like `"us-legal"` or `"us-letter"`, and more. Check out the reference for the page's paper argument[3.10.17/4.1] to learn about all available options.

[A] https://practicaltypography.com/page-margins.html
[B] https://designregression.com/article/line-length-revisited-following-the-research

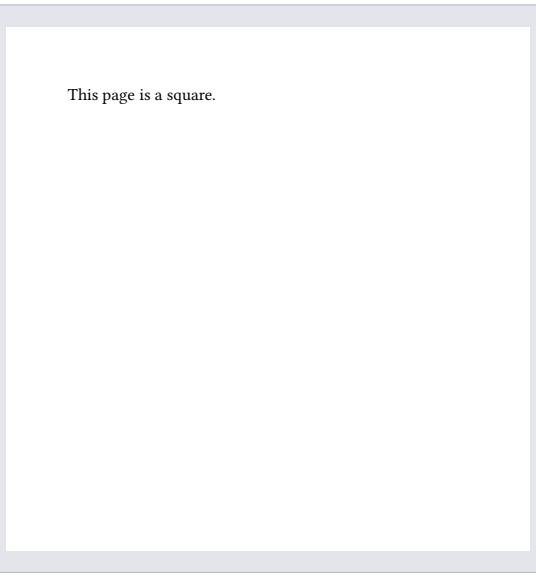
```
#set page("us-letter")

This page likes freedom.
```


If you need to customize your page size to some dimensions, you can specify the named arguments `width`[3.10.17/4.2] and `height`[3.10.17/4.3] instead.

```
#set page(width: 12cm, height: 12cm)

This page is a square.
```



/1 Change the page's margins

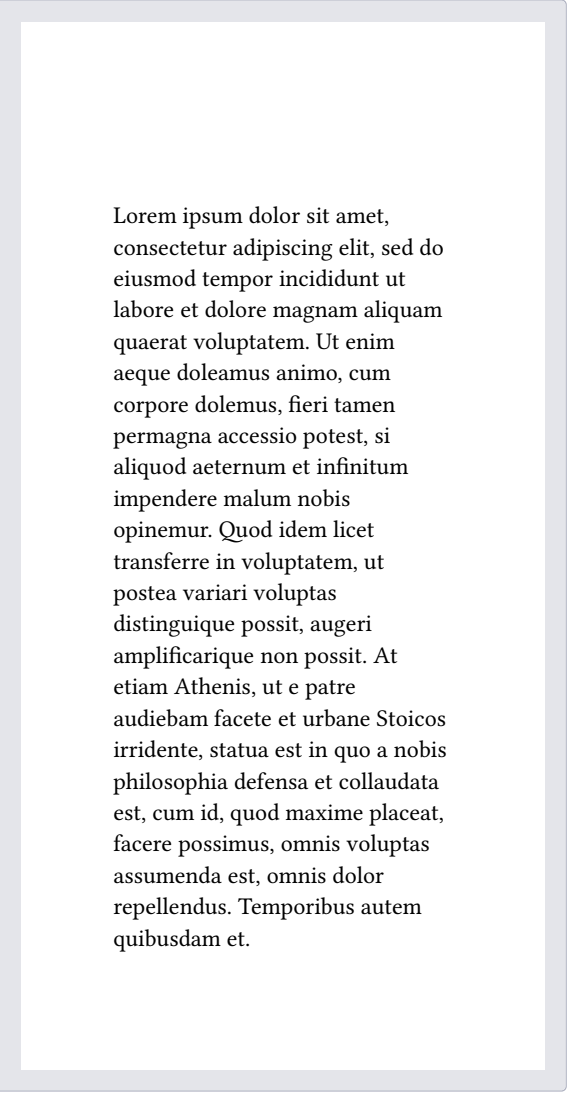
Margins are a vital ingredient for good typography: Typographers consider lines that fit between 45 and 75 characters best length for legibility[A] and your margins and columns[4.2.4] help define line widths. By default, Typst will create margins proportional to the page size of your document. To set custom margins, you will use the `margin`[3.10.17/4.5] argument in the `page`[3.10.17] set rule.

The `margin` argument will accept a length if you want to set all margins to the same width. However, you often want to set different margins on each side. To do this, you can pass a dictionary:

[A] https://webtypography.net/2.1.2

```
#set page(margin: (
  top: 3cm,
  bottom: 2cm,
  x: 1.5cm,
))

#lorem(100)
```



The page margin dictionary can have keys for each side (`top`, `bottom`, `left`, `right`), but you can also control left and right together by setting the `x` key of the margin dictionary, like in the example. Likewise, the top and bottom margins can be adjusted together by setting the `y` key.

If you do not specify margins for all sides in the margin dictionary, the old margins will remain in effect for the unset sides. To prevent this and set all remaining margins to a common size, you can use the `rest` key. For example, `#set page(margin: (left: 1.5in, rest: 1in))` will set the left margin to 1.5 inches and the remaining margins to one inch.

/2 Different margins on alternating pages

Sometimes, you'll need to alternate horizontal margins for even and odd pages, for example, to have more room towards the spine of a book than on the outsides of its pages. Typst keeps track of whether a page is to the left or right of the binding. You can use this information and set the `inside` or `outside` keys of the margin dictionary. The `inside` margin points towards the spine, and the `outside` margin points towards the edge of the bound book.

```
#set page(margin: (inside: 2.5cm, outside: 2cm, y: 1.75cm))
```

Typst will assume that documents written in Left-to-Right scripts are bound on the left while books written in Right-to-Left scripts are bound on the right. However, you will need to change this in some cases: If your first page is output by a different app, the binding is reversed from Typst's perspective. Also, some books, like English-language Mangas are customarily bound on the right, despite English using Left-to-Right script.

To change the binding side and explicitly set where the `inside` and `outside` are, set the `binding`[3.10.17/4.7] argument in the `page`[3.10.17] set rule.

```
// Produce a book bound on the right,
// even though it is set in Spanish.
#set text(lang: "es")
#set page(binding: right)
```

If `binding` is `left`, `inside` margins will be on the left on odd pages, and vice versa.

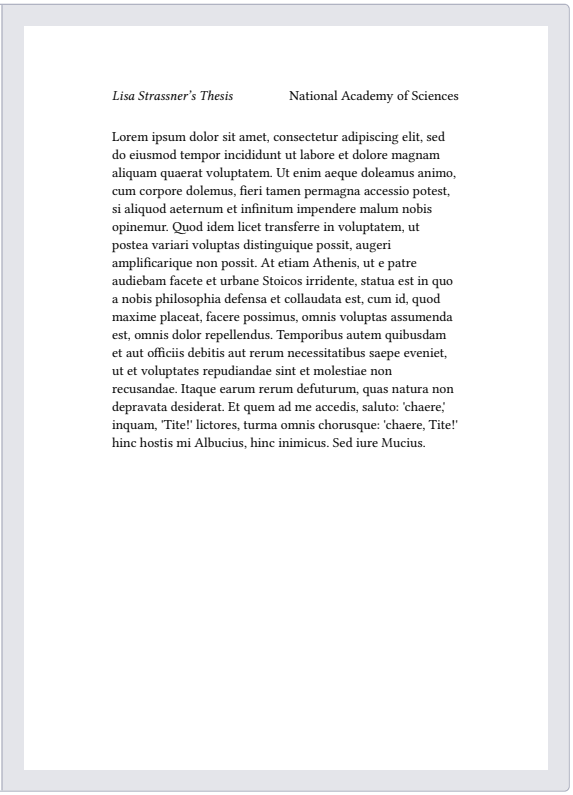
4.2.2 Add headers and footers

Headers and footers are inserted in the top and bottom margins of every page. You can add custom headers and footers or just insert a page number.

In case you need more than just a page number, the best way to insert a header and a footer are the `header`[3.10.17/4.13] and `footer`[3.10.17/4.15] arguments of the `page`[3.10.17] set rule. You can pass any content as their values:

```
#set page(header: [
  _Lisa Strassner's Thesis_
  #h(1fr)
  National Academy of Sciences
])

#lorem(150)
```



Headers are bottom-aligned by default so that they do not collide with the top edge of the page. You can change this by wrapping your header in the `align`[3.10.1] function.

/1 Different header and footer on specific pages

You'll need different headers and footers on some pages. For example, you may not want a header and footer on the title page. The example below shows how to conditionally remove the header on the first page:

```
#set page(header: context {
  if counter(page).get().first() > 1 [
    _Lisa Strassner's Thesis_
    #h(1fr)
    National Academy of Sciences
  ]
})
```

```
#lorem(150)
```

This example may look intimidating, but let's break it down: By using the `context` keyword, we are telling Typst that the header depends on where we are in the document. We then ask Typst if the page counter[3.12.1] is larger than one at our (context-dependent) current position. The page counter starts at one, so we are skipping the header on a single page. Counters may have multiple levels. This feature is used for items like headings, but the page counter will always have a single level, so we can just look at the first one.

You can, of course, add an `else` to this example to add a different header to the first page instead.

/2 Adapt headers and footers on pages with specific elements

The technique described in the previous section can be adapted to perform more advanced tasks using Typst's labels. For example, pages with big tables could omit their headers to help keep clutter down. We will mark our tables with a `<big-table>` label[3.5.17] and use the query system[3.12.6] to find out if such a label exists on the current page:

```
#set page(header: context {
  let matches = query(<big-table>)
  let current = counter(page).get()
  let has-table = matches.any(m =>
    counter(page).at(m.location()) == current
  )

  if not has-table [
    _Lisa Strassner's Thesis_
    #h(1fr)
    National Academy of Sciences
  ]
})

#lorem(100)
#pagebreak()

#table(
  columns: 2 * (1fr,),
  [A], [B],
  [C], [D],
) <big-table>
```

Here, we query for all instances of the `<big-table>` label. We then check that none of the tables are on the page at our current position. If so, we print the header. This example also uses variables to be more concise. Just as above, you could add an `else` to add another header instead of deleting it.

4.2.3 Add and customize page numbers

Page numbers help readers keep track of and reference your document more easily. The simplest way to insert page numbers is the `numbering`[3.10.17/4.10] argument of the page[3.10.17] set rule. You can pass a *numbering pattern*[3.6.13/3.1] string that shows how you want your pages to be numbered.

```
#set page(numbering: "1")

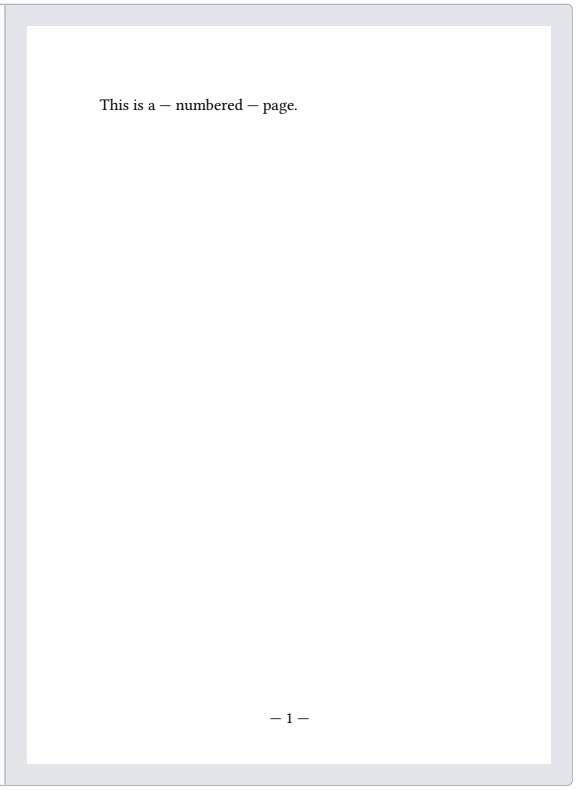
This is a numbered page.
```



Above, you can check out the simplest conceivable example. It adds a single Arabic page number at the center of the footer. You can specify other characters than `"1"` to get other numerals. For example, `"i"` will yield lowercase Roman numerals. Any character that is not interpreted as a number will be output as-is. For example, put dashes around your page number by typing this:

```
#set page(numbering: "— 1 —")

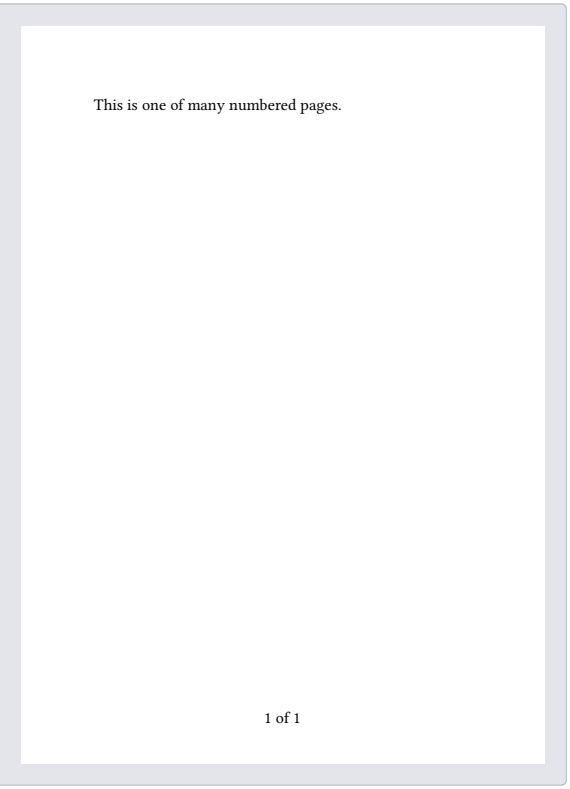
This is a — numbered — page.
```



You can add the total number of pages by entering a second number character in the string.

```
#set page(numbering: "1 of 1")

This is one of many numbered pages.
```



Go to the `numbering` function reference[3.6.13/3.1] to learn more about the arguments you can pass here.

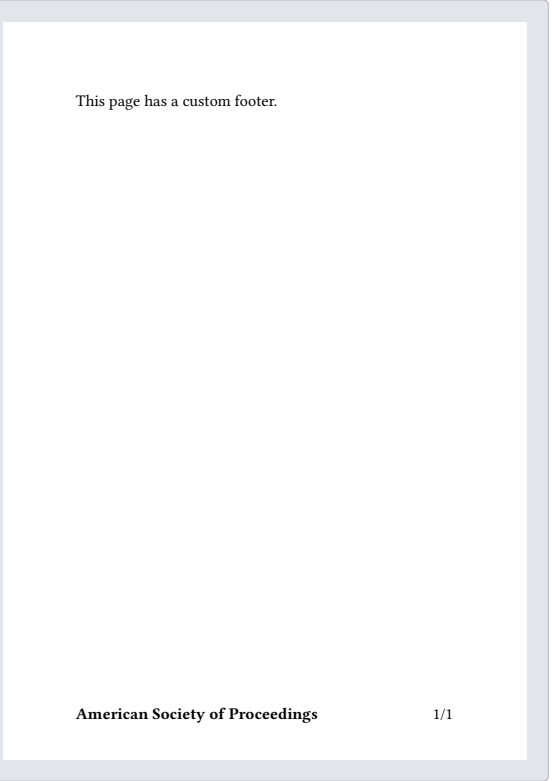
In case you need to right- or left-align the page number, use the `number-align`[3.10.17/4.12] argument of the `page`[3.10.17] set rule. Alternating alignment between even and odd pages is not currently supported using this property. To do this, you'll need to specify a custom footer and query the page counter as described in the section on conditionally omitting headers and footers.

/1 ## Custom footer with page numbers

Sometimes, you need to add other content than a page number to your footer. However, once a footer is specified, the `numbering`[3.10.17/4.10] argument of the `page`[3.10.17] set rule is ignored. This section shows you how to add a custom footer with page numbers and more.

```
#set page(footer: context [
  *American Society of Proceedings*
  #h(1fr)
  #counter(page).display(
    "1/1",
    both: true,
  )
])

This page has a custom footer.
```

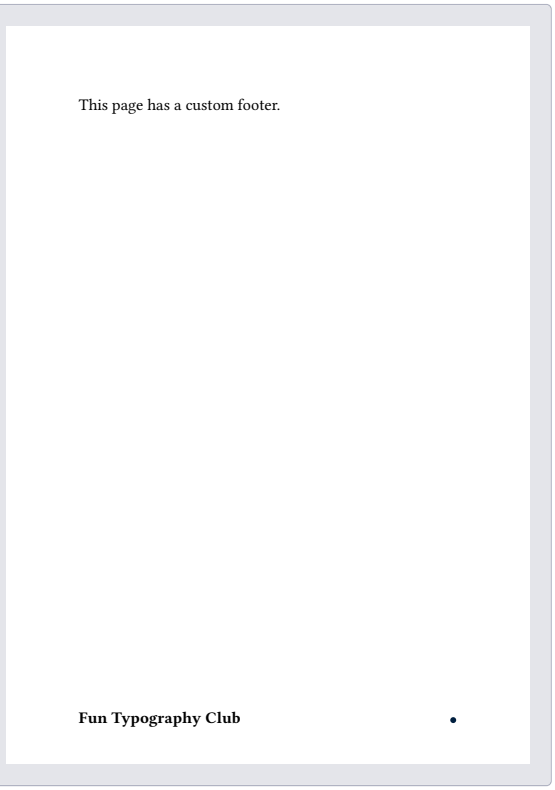


First, we add some strongly emphasized text on the left and add free space to fill the line. Then, we call `counter(page)` to retrieve the page counter and use its `display` function to show its current value. We also set `both` to `true` so that our numbering pattern applies to the current *and* final page number.

We can also get more creative with the page number. For example, let's insert a circle for each page.

```
#set page(footer: context [
  *Fun Typography Club*
  #h(1fr)
  #let (num,) = counter(page).get()
  #let circles = num * (
    box(circle(
      radius: 2pt,
      fill: navy,
    )),
  )
  #box(
    inset: (bottom: 1pt),
    circles.join(h(1pt))
  )
])

This page has a custom footer.
```



In this example, we use the number of pages to create an array of circles[3.11.2]. The circles are wrapped in a box[3.10.5] so they can all appear on the same line because they are blocks and would otherwise create paragraph breaks. The length of this array[3.5.2] depends on the current page number.

We then insert the circles at the right side of the footer, with 1pt of space between them. The join method of an array will attempt to *join*[3.3.2] the different values of an array into a single value, interspersed with its argument. In our case, we get a single content value with circles and spaces between them that we can use with the align function. Finally, we use another box to ensure that the text and the circles can share a line and use the `inset` argument[3.10.5/2.7] to raise the circles a bit so they line up nicely with the text.

/2 Reset the page number and skip pages

Do you, at some point in your document, need to reset the page number? Maybe you want to start with the first page only after the title page. Or maybe you need to skip a few page numbers because you will insert pages into the final printed product.

The right way to modify the page number is to manipulate the page counter[3.12.1]. The simplest manipulation is to set the counter back to 1.

```
#counter(page).update(1)
```

This line will reset the page counter back to one. It should be placed at the start of a page because it will otherwise create a page break. You can also update the counter given its previous value by passing a function:

```
#counter(page).update(n => n + 5)
```

In this example, we skip five pages. `n` is the current value of the page counter and `n + 5` is the return value of our function.

In case you need to retrieve the actual page number instead of the value of the page counter, you can use the `page`[3.12.4/2.1] method on the return value of the `here`[3.12.2] function:

```
#counter(page).update(n => n + 5)

// This returns one even though the
// page counter was incremented by 5.
#context here().page()
```

1

You can also obtain the page numbering pattern from the location returned by `here` with the `page-numbering`[3.12.4/2.3] method.

4.2.4 Add columns

Add columns to your document to fit more on a page while maintaining legible line lengths. Columns are vertical blocks of text which are separated by some whitespace. This space is called the gutter.

To lay out your content in columns, just specify the desired number of columns in a `page`[3.10.17/4.8] set rule. To adjust the amount of space between the columns, add a set rule on the `columns` function[3.10.7], specifying the `gutter` parameter.

```
#set page(columns: 2)
#set columns(gutter: 12pt)

#lorem(30)
```

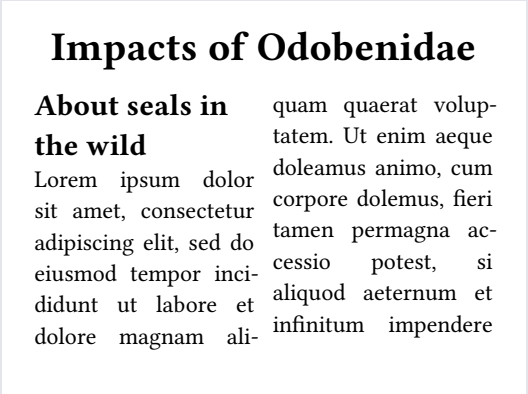
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri.

Very commonly, scientific papers have a single-column title and abstract, while the main body is set in two-columns. To achieve this effect, Typst's `place` function[3.10.19] can temporarily escape the two-column layout by specifying `float: true` and `scope: "parent"`:

```
#set document(
  title: [Impacts of Odobenidae],
)
#set page(columns: 2)
#set par(justify: true)

#place(
  top + center,
  float: true,
  scope: "parent",
  title(),
)

= About seals in the wild
#lorem(80)
```

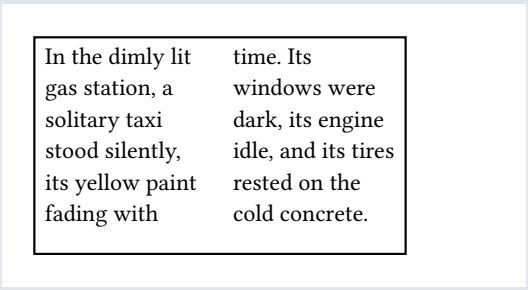


Floating placement refers to elements being pushed to the top or bottom of the column or page, with the remaining content flowing in between. It is also frequently used for figures[3.6.8/5.3].

/1 Use columns anywhere in your document

To create columns within a nested layout, e.g. within a rectangle, you can use the `columns` function[3.10.7] directly. However, it really should only be used within nested layouts. At the page-level, the page set rule is preferable because it has better interactions with things like page-level floats, footnotes, and line numbers.

```
#rect(
  width: 6cm,
  height: 3.5cm,
  columns(2, gutter: 12pt)[
    In the dimly lit gas station,
    a solitary taxi stood silently,
    its yellow paint fading with
    time. Its windows were dark,
    its engine idle, and its tires
    rested on the cold concrete.
  ]
)
```



/2 Balanced columns

If the columns on the last page of a document differ greatly in length, they may create a lopsided and unappealing layout. That's why typographers will often equalize the length of columns on the last page. This effect is called balancing columns. Typst cannot yet balance columns automatically. However, you can balance columns manually by placing `#colbreak()`[3.10.6] at an appropriate spot in your markup, creating the desired column break manually.

4.2.5 One-off modifications

You do not need to override your page settings if you need to insert a single page with a different setup. For example, you may want to insert a page that's flipped to landscape to insert a big table or change the margin and columns for your title page. In this case, you can call `page`[3.10.17] as a function with your content as an argument and the overrides as the other arguments. This will insert enough new pages with your overridden settings to place your content on them. Typst will revert to the page settings from the set rule after the call.

```
#page(flipped: true)[
  = Multiplication table

  #table(
    columns: 5 * (1fr,),
    ..for x in range(1, 10) {
      for y in range(1, 6) {
        (str(x*y),)
      }
    }
  )
]
```

Multiplication table

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25
6	12	18	24	30
7	14	21	28	35
8	16	24	32	40
9	18	27	36	45

4.3 Table Guide

Tables are a great way to present data to your readers in an easily readable, compact, and organized manner. They are not only used for numerical values, but also survey responses, task planning, schedules, and more. Because of this wide set of possible applications, there is no single best way to lay out a table. Instead, think about the data you want to highlight, your document's overarching design, and ultimately how your table can best serve your readers.

Typst can help you with your tables by automating styling, importing data from other applications, and more! This guide takes you through a few of the most common questions you may have when adding a table to your document with Typst. Feel free to skip to the section most relevant to you – we designed this guide to be read out of order.

If you want to look up a detail of how tables work, you should also check out their reference page[3.6.20]. And if you are looking for a table of contents rather than a normal table, the reference page of the `outline` function[3.6.14] is the right place to learn more.

4.3.1 How to create a basic table?

In order to create a table in Typst, use the `table` function[3.6.20]. For a basic table, you need to tell the table function two things:

- The number of columns
- The content for each of the table cells

So, let's say you want to create a table with two columns describing the ingredients for a cookie recipe:

```
#table(
  columns: 2,
  [*Amount*], [*Ingredient*],
  [360g], [Baking flour],
  [250g], [Butter (room temp.)],
  [150g], [Brown sugar],
  [100g], [Cane sugar],
  [100g], [70% cocoa chocolate],
  [100g], [35–40% cocoa chocolate],
  [2], [Eggs],
  [Pinch], [Salt],
  [Drizzle], [Vanilla extract],
)
```

Amount	Ingredient
360g	Baking flour
250g	Butter (room temp.)
150g	Brown sugar
100g	Cane sugar
100g	70% cocoa chocolate
100g	35-40% cocoa chocolate
2	Eggs
Pinch	Salt
Drizzle	Vanilla extract

This example shows how to call, configure, and populate a table. Both the column count and cell contents are passed to the table as arguments. The argument list[3.5.15] is surrounded by round parentheses. In it, we first pass the column count as a named argument. Then, we pass multiple content blocks[3.5.8] as positional arguments. Each content block contains the contents for a single cell.

To make the example more legible, we have placed two content block arguments on each line, mimicking how they would appear in the table. You could also write each cell on its own line. Typst does not care on which line you place the arguments. Instead, Typst will place the content cells from left to right (or right to left, if that is the writing direction of your language) and then from top to bottom. It will automatically add enough rows to your table so that it fits all of your content.

It is best to wrap the header row of your table in the `table.header` function[3.6.20/4.4]. This clarifies your intent and will also allow Typst to make the output more accessible[4.4] to users with a screen reader:

```
#table(
  columns: 2,
  table.header[*Amount*][*Ingredient*],
  [360g], [Baking flour],
 // ... the remaining cells
)
```

Amount	Ingredient
360g	Baking flour
250g	Butter (room temp.)
150g	Brown sugar
100g	Cane sugar
100g	70% cocoa chocolate
100g	35-40% cocoa chocolate
2	Eggs
Pinch	Salt
Drizzle	Vanilla extract

You could also write a show rule that automatically strongly emphasizes[3.6.19] the contents of the first cells for all tables. This quickly becomes useful if your document contains multiple tables!

```
#show table.cell.where(y: 0): strong

#table(
  columns: 2,
  table.header[Amount][Ingredient],
  [360g], [Baking flour],
 // ... the remaining cells
)
```

Amount	Ingredient
360g	Baking flour
250g	Butter (room temp.)
150g	Brown sugar
100g	Cane sugar
100g	70% cocoa chocolate
100g	35-40% cocoa chocolate
2	Eggs
Pinch	Salt
Drizzle	Vanilla extract

We are using a show rule with a selector for cell coordinates here instead of applying our styles directly to `table.header`. This is due to a current limitation of Typst that will be fixed in a future release.

Congratulations, you have created your first table! Now you can proceed to change column sizes[4.3.2], adjust the strokes[4.3.5], add striped rows[4.3.4], and more!

4.3.2 How to change the column sizes?

If you create a table and specify the number of columns, Typst will make each column large enough to fit its largest cell. Often, you want something different, for example, to make a table span the whole width of the page. You can provide a list, specifying how wide you want each column to be, through the `columns` argument. There are a few different ways to specify column widths:

- First, there is `auto`. This is the default behavior and tells Typst to grow the column to fit its contents. If there is not enough space, Typst will try its best to distribute the space among the `auto`-sized columns.
- Lengths[3.10.13] like `6cm`, `0.7in`, or `120pt`. As usual, you can also use the font-dependent `em` unit. This is a multiple of your current font size. It's useful if you want to size your table so that it always fits about the same amount of text, independent of font size.
- A ratio in percent[3.10.20] such as `40%`. This will make the column take up 40% of the total horizontal space available to the table, so either the inner width of the page or the table's container. You can also mix ratios and lengths into relative lengths[3.10.21]. Be mindful that even if you specify a list of column widths that sum up to 100%, your table could still become larger than its container. This is because there can be gutter[3.6.20/3.3] between columns that is not included in the column widths. If you want to make a table fill the page, the next option is often very useful.

– A fractional part of the free space[3.10.9] using the `fr` unit, such as `1fr`. This unit allows you to distribute the available space to columns. It works as follows: First, Typst sums up the lengths of all columns that do not use `fr`s. Then, it determines how much horizontal space is left. This horizontal space then gets distributed to all columns denominated in `fr`s. During this process, a `2fr` column will become twice as wide as a `1fr` column. This is where the name comes from: The width of the column is its fraction of the total fractionally sized columns.

Let's put this to use with a table that contains the dates, numbers, and descriptions of some routine checks. The first two columns are `auto`-sized and the last column is `1fr` wide as to fill the whole page.

```
#table(
  columns: (auto, auto, 1fr),
  table.header[Date][°No][Description],
  [24/01/03], [813], [Filtered participant pool],
  [24/01/03], [477], [Transitioned to sec. regimen],
  [24/01/11], [051], [Cycled treatment substrate],
)
```

Date	°No	Description
24/01/03	813	Filtered participant pool
24/01/03	477	Transitioned to sec. regimen
24/01/11	051	Cycled treatment substrate

Here, we have passed our list of column lengths as an array[3.5.2], enclosed in round parentheses, with its elements separated by commas. The first two columns are automatically sized, so that they take on the size of their content and the third column is sized as `1fr` so that it fills up the remainder of the space on the page. If you wanted to instead change the second column to be a bit more spacious, you could replace its entry in the `columns` array with a value like `6em`.

4.3.3 How to caption and reference my table?

A table is just as valuable as the information your readers draw from it. You can enhance the effectiveness of both your prose and your table by making a clear connection between the two with a cross-reference. Typst can help you with automatic references[3.6.18] and the `figure` function[3.6.8].

Just like with images, wrapping a table in the `figure` function allows you to add a caption and a label, so you can reference the figure elsewhere. Wrapping your table in a figure also lets you use the figure's `placement` parameter to float it to the top or bottom of a page.

Let's take a look at a captioned table and how to reference it in prose:

```
#show table.cell.where(y: 0): set text(weight: "bold")

#figure(
  table(
    columns: 4,
    stroke: none,

    table.header[Test Item][Specification][Test Result][Compliance],
    [Voltage], [220V ± 5%], [218V], [Pass],
    [Current], [5A ± 0.5A], [4.2A], [Fail],
  ),
  caption: [Probe results for design A],
) <probe-a>
```

```
The results from @probe-a show that the design is not yet optimal.
We will show how its performance can be improved in this section.
```

Test Item	Specification	Test Result	Compliance
Voltage	220V ± 5%	218V	Pass
Current	5A ± 0.5A	4.2A	Fail

Table 1: Probe results for design A

The results from Table 1 show that the design is not yet optimal. We will show how its performance can be improved in this section.

The example shows how to wrap a table in a figure, set a caption and a label, and how to reference that label. We start by using the `figure` function. It expects the contents of the figure as a positional argument. We just put the table function call in its argument list, omitting the `#` character because it is only needed when calling a function in markup mode. We also add the caption as a named argument (above or below) the table.

After the figure call, we put a label in angle brackets (`<probe-a>`). This tells Typst to remember this element and make it referenceable under this name throughout your document. We can then reference it in prose by using the at sign and the label name `@probe-a`. Typst will print a nicely formatted reference and automatically update the label if the table's number changes.

4.3.4 How to get a striped table?

Many tables use striped rows or columns instead of strokes to differentiate between rows and columns. This effect is often called *zebra stripes.* Tables with zebra stripes are popular in Business and commercial Data Analytics applications, while academic applications tend to use strokes instead.

To add zebra stripes to a table, we use the `table` function's `fill` argument. It can take three kinds of arguments:

- A single color (this can also be a gradient or a tiling) to fill all cells with. Because we want some cells to have another color, this is not useful if we want to build zebra tables.
- An array with colors which Typst cycles through for each column. We can use an array with two elements to get striped columns.
- A function that takes the horizontal coordinate `x` and the vertical coordinate `y` of a cell and returns its fill. We can use this to create horizontal stripes or checkerboard patterns[3.10.10/6.1].

Let's start with an example of a horizontally striped table:

```
#set text(font: "IBM Plex Sans")

// Medium bold table header.
#show table.cell.where(y: 0): set text(weight: "medium")

// Bold titles.
#show table.cell.where(x: 1): set text(weight: "bold")

// See the strokes section for details on this!
#let frame(stroke) = (x, y) => (
  left: if x > 0 { 0pt } else { stroke },
  right: stroke,
  top: if y < 2 { stroke } else { 0pt },
  bottom: stroke,
)
```

```
#set table(
  fill: (rgb("EAF2F5"), none),
  stroke: frame(1pt + rgb("21222C")),
)

#table(
  columns: (0.4fr, 1fr, 1fr, 1fr),

  table.header[Month][Title][Author][Genre],
  [January], [The Great Gatsby], [F. Scott Fitzgerald], [Classic],
  [February], [To Kill a Mockingbird], [Harper Lee], [Drama],
  [March], [1984], [George Orwell], [Dystopian],
  [April], [The Catcher in the Rye], [J.D. Salinger], [Coming-of-Age],
)
```

Month	Title	Author	Genre
January	**The Great Gatsby**	F. Scott Fitzgerald	Classic
February	**To Kill a Mockingbird**	Harper Lee	Drama
March	**1984**	George Orwell	Dystopian
April	**The Catcher in the Rye**	J.D. Salinger	Coming-of-Age

This example shows a book club reading list. The line `fill: (rgb("EAF2F5"), none)` in `table`'s set rule is all that is needed to add striped columns. It tells Typst to alternate between coloring columns with a light blue (in the rgb[3.11.3/4.5] function call) and nothing (`none`). Note that we extracted all of our styling from the `table` function call itself into set and show rules, so that we can automatically reuse it for multiple tables.

Because setting the stripes itself is easy we also added some other styles to make it look nice. The other code in the example provides a dark blue stroke[4.3.5/3] around the table and below the first line and emboldens the first row and the column with the book title. See the strokes[4.3.5] section for details on how we achieved this stroke configuration.

Let's next take a look at how we can change only the set rule to achieve horizontal stripes instead:

```
#set table(
  fill: (_, y) => if calc.odd(y) { rgb("EAF2F5") },
  stroke: frame(1pt + rgb("21222C")),
)
```

Month	Title	Author	Genre
January	**The Great Gatsby**	F. Scott Fitzgerald	Classic
February	**To Kill a Mockingbird**	Harper Lee	Drama
March	**1984**	George Orwell	Dystopian
April	**The Catcher in the Rye**	J.D. Salinger	Coming-of-Age

We just need to replace the set rule from the previous example with this one and get horizontal stripes instead. Here, we are passing a function to `fill`. It discards the horizontal coordinate with an underscore and then checks if the vertical coordinate y of the cell is odd. If so, the cell gets a light blue fill, otherwise, no fill is returned.

Of course, you can make this function arbitrarily complex. For example, if you want to stripe the rows with a light and darker shade of blue, you could do something like this:

```
#set table(
  fill: (_, y) => (none, rgb("EAF2F5"), rgb("DDEAEF")).at(calc.rem(y, 3)),
  stroke: frame(1pt + rgb("21222C")),
)
```

Month	Title	Author	Genre
January	The Great Gatsby	F. Scott Fitzgerald	Classic
February	To Kill a Mockingbird	Harper Lee	Drama
March	1984	George Orwell	Dystopian
April	The Catcher in the Rye	J.D. Salinger	Coming-of-Age

This example shows an alternative approach to write our fill function. The function uses an array with three colors and then cycles between its values for each row by indexing the array with the remainder of y divided by 3.

Finally, here is a bonus example that uses the *stroke* to achieve striped rows:

```
#set table(
  stroke: (x, y) => (
    y: 1pt,
    left: if x > 0 { 0pt } else if calc.even(y) { 1pt },
    right: if calc.even(y) { 1pt },
  ),
)
```

Month	Title	Author	Genre
January	The Great Gatsby	F. Scott Fitzgerald	Classic
February	To Kill a Mockingbird	Harper Lee	Drama
March	1984	George Orwell	Dystopian
April	The Catcher in the Rye	J.D. Salinger	Coming-of-Age

/1 ## Manually overriding a cell's fill color

Sometimes, the fill of a cell needs not to vary based on its position in the table, but rather based on its contents. We can use the `table.cell` element[3.6.20/4.1] in the `table`'s parameter list to wrap a cell's content and override its fill.

For example, here is a list of all German presidents, with the cell borders colored in the color of their party.

```
#set text(font: "Roboto")

#let cdu(name) = ([CDU], table.cell(fill: black, text(fill: white, name)))
#let spd(name) = ([SPD], table.cell(fill: red, text(fill: white, name)))
#let fdp(name) = ([FDP], table.cell(fill: yellow, name))

#table(
  columns: (auto, auto, 1fr),
  stroke: (x: none),

  table.header[Tenure][Party][President],
  [1949–1959], ..fdp[Theodor Heuss],
```

```
  [1959–1969], ..cdu[Heinrich Lübke],
  [1969–1974], ..spd[Gustav Heinemann],
  [1974–1979], ..fdp[Walter Scheel],
  [1979–1984], ..cdu[Karl Carstens],
  [1984–1994], ..cdu[Richard von Weizsäcker],
  [1994–1999], ..cdu[Roman Herzog],
  [1999–2004], ..spd[Johannes Rau],
  [2004–2010], ..cdu[Horst Köhler],
  [2010–2012], ..cdu[Christian Wulff],
  [2012–2017], [n/a], [Joachim Gauck],
  [2017–],     ..spd[Frank-Walter-Steinmeier],
)
```

Tenure	Party	President
1949-1959	FDP	Theodor Heuss
1959-1969	CDU	Heinrich Lübke
1969-1974	SPD	Gustav Heinemann
1974-1979	FDP	Walter Scheel
1979-1984	CDU	Karl Carstens
1984-1994	CDU	Richard von Weizsäcker
1994-1999	CDU	Roman Herzog
1999-2004	SPD	Johannes Rau
2004-2010	CDU	Horst Köhler
2010-2012	CDU	Christian Wulff
2012-2017	n/a	Joachim Gauck
2017-	SPD	Frank-Walter-Steinmeier

In this example, we make use of variables because there only have been a total of three parties whose members have become president (and one unaffiliated president). Their colors will repeat multiple times, so we store a function that produces an array with their party's name and a table cell with that party's color and the president's name (`cdu`, `spd`, and `fdp`). We then use these functions in the `table` argument list instead of directly adding the name. We use the spread operator[3.5.1/2] `..` to turn the items of the arrays into single cells. We could also write something like `[FDP], table.cell(fill: yellow)[Theodor Heuss]` for each cell directly in the `table`'s argument list, but that becomes unreadable, especially for the parties whose colors are dark so that they require white text. We also delete vertical strokes and set the font to Roboto.

The party column and the cell color in this example communicate redundant information on purpose: Communicating important data using color only is a bad accessibility practice. It disadvantages users with vision impairment and is in violation of universal access standards, such as the WCAG 2.1 Success Criterion 1.4.1[A]. To improve this table, we added a column printing the party name. Alternatively, you could have made sure to choose a color-blindness friendly palette and mark up your cells with an additional label that screen readers can read out loud. The latter feature is not currently supported by Typst, but will be added in a future release. You can check how colors look for color-blind readers with this Chrome extension[B], Photoshop[C], or GIMP[D].

[A] https://www.w3.org/WAI/WCAG21/Understanding/use-of-color.html
[B] https://chromewebstore.google.com/detail/colorblindly/floniaahmccleoclneebhhmnjgdfijgg
[C] https://helpx.adobe.com/photoshop/using/proofing-colors.html
[D] https://docs.gimp.org/2.10/en/gimp-display-filter-dialog.html

4.3.5 How to adjust the lines in a table?

By default, Typst adds strokes between each row and column of a table. You can adjust these strokes in a variety of ways. Which one is the most practical, depends on the modification you want to make and your intent:

- Do you want to style all tables in your document, irrespective of their size and content? Use the `table` function's stroke[3.6.20/3.9] argument in a set rule.
- Do you want to customize all lines in a single table? Use the `table` function's stroke[3.6.20/3.9] argument when calling the table function.
- Do you want to change, add, or remove the stroke around a single cell? Use the `table.cell` element in the argument list of your table call.
- Do you want to change, add, or remove a single horizontal or vertical stroke in a single table? Use the `table.hline`[3.6.20/4.2] and `table.vline`[3.6.20/4.3] elements in the argument list of your table call.

We will go over all of these options with examples next! First, we will tackle the `table` function's stroke[3.6.20/3.9] argument. Here, you can adjust both how the table's lines get drawn and configure which lines are drawn at all.

Let's start by modifying the color and thickness of the stroke:

```
#table(
  columns: 4,
  stroke: 0.5pt + rgb("666675"),
  [*Monday*], [11.5], [13.0], [4.0],
  [*Tuesday*], [8.0], [14.5], [5.0],
  [*Wednesday*], [9.0], [18.5], [13.0],
)
```

Monday	11.5	13.0	4.0
Tuesday	8.0	14.5	5.0
Wednesday	9.0	18.5	13.0

This makes the table lines a bit less wide and uses a bluish gray. You can see that we added a width in point to a color to achieve our customized stroke. This addition yields a value of the stroke type[3.11.12]. Alternatively, you can use the dictionary representation for strokes which allows you to access advanced features such as dashed lines.

The previous example showed how to use the stroke argument in the table function's invocation. Alternatively, you can specify the stroke argument in the `table`'s set rule. This will have exactly the same effect on all subsequent `table` calls as if the stroke argument was specified in the argument list. This is useful if you are writing a template or want to style your whole document.

```
// Renders the exact same as the last example
#set table(stroke: 0.5pt + rgb("666675"))

#table(
  columns: 4,
  [*Monday*], [11.5], [13.0], [4.0],
  [*Tuesday*], [8.0], [14.5], [5.0],
  [*Wednesday*], [9.0], [18.5], [13.0],
)
```

For small tables, you sometimes want to suppress all strokes because they add too much visual noise. To do this, just set the stroke argument to `none`:

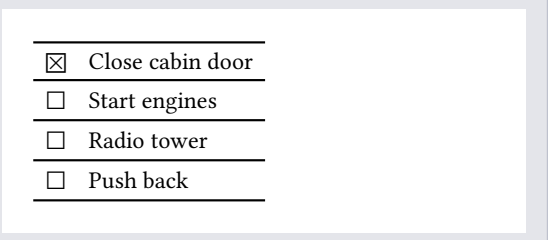
```
#table(
  columns: 4,
  stroke: none,
  [*Monday*], [11.5], [13.0], [4.0],
  [*Tuesday*], [8.0], [14.5], [5.0],
  [*Wednesday*], [9.0], [18.5], [13.0],
)
```

Monday	11.5	13.0	4.0
Tuesday	8.0	14.5	5.0
Wednesday	9.0	18.5	13.0

If you want more fine-grained control of where lines get placed in your table, you can also pass a dictionary with the keys `top`, `left`, `right`, `bottom` (controlling the respective cell sides), `x`, `y` (controlling vertical and

horizontal strokes), and `rest` (covers all strokes not styled by other dictionary entries). All keys are optional; omitted keys will use their previously set value, or the default value if never set. For example, to get a table with only horizontal lines, you can do this:

```
#table(
  columns: 2,
  stroke: (x: none),
  align: horizon,
  [☒], [Close cabin door],
  [☐], [Start engines],
  [☐], [Radio tower],
  [☐], [Push back],
)
```



This turns off all vertical strokes and leaves the horizontal strokes in place. To achieve the reverse effect (only vertical strokes), set the stroke argument to `(y: none)` instead.

Further down in the guide[4.3.5/3], we cover how to use a function in the stroke argument to customize all strokes individually. This is how you achieve more complex stroking patterns.

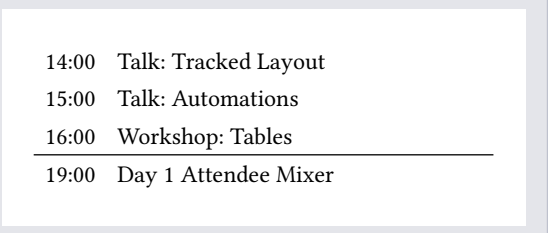
/1 ## Adding individual lines in the table

If you want to add a single horizontal or vertical line in your table, for example to separate a group of rows, you can use the `table.hline`[3.6.20/4.2] and `table.vline`[3.6.20/4.3] elements for horizontal and vertical lines, respectively. Add them to the argument list of the `table` function just like you would add individual cells and a header.

Let's take a look at the following example from the reference:

```
#set table.hline(stroke: 0.6pt)

#table(
  stroke: none,
  columns: (auto, 1fr),
  // Morning schedule abridged.
  [14:00], [Talk: Tracked Layout],
  [15:00], [Talk: Automations],
  [16:00], [Workshop: Tables],
  table.hline(),
  [19:00], [Day 1 Attendee Mixer],
)
```



In this example, you can see that we have placed a call to `table.hline` between the cells, producing a horizontal line at that spot. We also used a set rule on the element to reduce its stroke width to make it fit better with the weight of the font.

By default, Typst places horizontal and vertical lines after the current row or column, depending on their position in the argument list. You can also manually move them to a different position by adding the `y` (for `hline`) or `x` (for `vline`) argument. For example, the code below would produce the same result:

```
#set table.hline(stroke: 0.6pt)

#table(
  stroke: none,
  columns: (auto, 1fr),
  // Morning schedule abridged.
  table.hline(y: 3),
  [14:00], [Talk: Tracked Layout],
  [15:00], [Talk: Automations],
  [16:00], [Workshop: Tables],
  [19:00], [Day 1 Attendee Mixer],
)
```

Let's imagine you are working with a template that shows none of the table strokes except for one between the first and second row. Now, since you have one table that also has labels in the first column, you want to add an extra vertical line to it. However, you do not want this vertical line to cross into the top row. You can achieve this with the `start` argument:

```
// Base template already configured tables, but we need some
// extra configuration for this table.
#{
  set table(align: (x, _) => if x == 0 { left } else { right })
  show table.cell.where(x: 0): smallcaps
  table(
    columns: (auto, 1fr, 1fr, 1fr),
    table.vline(x: 1, start: 1),
    table.header[Trainset][Top Speed][Length][Weight],
    [TGV Réseau], [320 km/h], [200m], [383t],
    [ICE 403], [330 km/h], [201m], [409t],
    [Shinkansen N700], [300 km/h], [405m], [700t],
  )
}
```

Trainset	Top Speed	Length	Weight
TGV Réseau	320 km/h	200m	383t
ICE 403	330 km/h	201m	409t
Shinkansen N700	300 km/h	405m	700t

In this example, we have added `table.vline` at the start of our positional argument list. But because the line is not supposed to go to the left of the first column, we specified the x argument as `1`. We also set the `start` argument to `1` so that the line does only start after the first row.

The example also contains two more things: We use the align argument with a function to right-align the data in all but the first column and use a show rule to make the first column of table cells appear in small capitals. Because these styles are specific to this one table, we put everything into a code block[3.3.2], so that the styling does not affect any further tables.

/2 ## Overriding the strokes of a single cell

Imagine you want to change the stroke around a single cell. Maybe your cell is very important and needs highlighting! For this scenario, there is the `table.cell` function[3.6.20/4.1]. Instead of adding your content directly in the argument list of the table, you wrap it in a `table.cell` call. Now, you can use `table.cell`'s argument list to override the table properties, such as the stroke, for this cell only.

Here's an example with a matrix of two of the Big Five personality factors, with one intersection highlighted.

```
#table(
  columns: 3,
  stroke: (x: none),

  table.header[][*High Neuroticism*][*Low Neuroticism*],

  [*High Agreeableness*],
  table.cell(stroke: orange + 2pt)[
    _Sensitive_ \ Prone to emotional distress but very empathetic.
  ],
  [_Compassionate_ \ Caring and stable, often seen as a supportive figure.],

  [*Low Agreeableness*],
  [_Contentious_ \ Competitive and easily agitated.],
```

```
  [_Detached_ \ Independent and calm, may appear aloof.],
)
```

	High Neuroticism	Low Neuroticism
High Agreeableness	*Sensitive* Prone to emotional distress but very empathetic.	*Compassionate* Caring and stable, often seen as a supportive figure.
Low Agreeableness	*Contentious* Competitive and easily agitated.	*Detached* Independent and calm, may appear aloof.

Above, you can see that we used the `table.cell` element in the table's argument list and passed the cell content to it. We have used its `stroke` argument to set a wider orange stroke. Despite the fact that we disabled vertical strokes on the table, the orange stroke appeared on all sides of the modified cell, showing that the table's stroke configuration is overwritten.

/3 ## Complex document-wide stroke customization

This section explains how to customize all lines at once in one or multiple tables. This allows you to draw only the first horizontal line or omit the outer lines, without knowing how many cells the table has. This is achieved by providing a function to the table's `stroke` parameter. The function should return a stroke given the zero-indexed x and y position of the current cell. You should only need these functions if you are a template author, do not use a template, or need to heavily customize your tables. Otherwise, your template should set appropriate default table strokes.

For example, this is a set rule that draws all horizontal lines except for the very first and last line.

```
#show table.cell.where(x: 0): set text(style: "italic")
#show table.cell.where(y: 0): set text(style: "normal", weight: "bold")
#set table(stroke: (_, y) => if y > 0 { (top: 0.8pt) })

#table(
  columns: 3,
  align: center + horizon,
  table.header[Technique][Advantage][Drawback],
  [Diegetic], [Immersive], [May be contrived],
  [Extradiegetic], [Breaks immersion], [Obtrusive],
  [Omitted], [Fosters engagement], [May fracture audience],
)
```

Technique	Advantage	Drawback
Diegetic	Immersive	May be contrived
Extradiegetic	Breaks immersion	Obtrusive
Omitted	Fosters engagement	May fracture audience

In the set rule, we pass a function that receives two arguments, assigning the vertical coordinate to y and discarding the horizontal coordinate. It then returns a stroke dictionary with a `0.8pt` top stroke for all but

the first line. The cells in the first line instead implicitly receive `none` as the return value. You can easily modify this function to just draw the inner vertical lines instead as `(x, _) => if x > 0 { (left: 0.8pt) }`.

Let's try a few more stroking functions. The next function will only draw a line below the first row:

```
#set table(stroke: (_, y) => if y == 0 { (bottom: 1pt) })

// Table as seen above
```

Technique	Advantage	Drawback
Diegetic	Immersive	May be contrived
Extradiegetic	Breaks immersion	Obtrusive
Omitted	Fosters engagement	May fracture audience

If you understood the first example, it becomes obvious what happens here. We check if we are in the first row. If so, we return a bottom stroke. Otherwise, we'll return `none` implicitly.

The next example shows how to draw all but the outer lines:

```
#set table(stroke: (x, y) => (
  left: if x > 0 { 0.8pt },
  top: if y > 0 { 0.8pt },
))

// Table as seen above
```

Technique	Advantage	Drawback
Diegetic	Immersive	May be contrived
Extradiegetic	Breaks immersion	Obtrusive
Omitted	Fosters engagement	May fracture audience

This example uses both the x and y coordinates. It omits the left stroke in the first column and the top stroke in the first row. The right and bottom lines are not drawn.

Finally, here is a table that draws all lines except for the vertical lines in the first row and horizontal lines in the table body. It looks a bit like a calendar.

```
#set table(stroke: (x, y) => (
  left: if x == 0 or y > 0 { 1pt } else { 0pt },
  right: 1pt,
  top: if y <= 1 { 1pt } else { 0pt },
  bottom: 1pt,
))

// Table as seen above
```

Technique	Advantage	Drawback
Diegetic	Immersive	May be contrived
Extradiegetic	Breaks immersion	Obtrusive
Omitted	Fosters engagement	May fracture audience

This example is a bit more complex. We start by drawing all the strokes on the right of the cells. But this means that we have drawn strokes in the top row, too, and we don't need those! We use the fact that `left` will override `right` and only draw the left line if we are not in the first row or if we are in the first column. In all other cases, we explicitly remove the left line. Finally, we draw the horizontal lines by first setting the bottom line and then for the first two rows with the `top` key, suppressing all other top lines. The last line appears because there is no `top` line that could suppress it.

/4 **How to achieve a double line?**

Typst does not yet have a native way to draw double strokes, but there are multiple ways to emulate them, for example with tilings[3.11.13]. We will show a different workaround in this section: Table gutters.

Tables can space their cells apart using the `gutter` argument. When a gutter is applied, a stroke is drawn on each of the now separated cells. We can selectively add gutter between the rows or columns for which we want to draw a double line. The `row-gutter` and `column-gutter` arguments allow us to do this. They accept arrays of gutter values. Let's take a look at an example:

```
#table(
  columns: 3,
  stroke: (x: none),
  row-gutter: (2.2pt, auto),
  table.header[Date][Exercise Type][Calories Burned],
  [2023-03-15], [Swimming], [400],
  [2023-03-17], [Weightlifting], [250],
  [2023-03-18], [Yoga], [200],
)
```

Date	Exercise Type	Calories Burned
2023-03-15	Swimming	400
2023-03-17	Weightlifting	250
2023-03-18	Yoga	200

We can see that we used an array for `row-gutter` that specifies a `2.2pt` gap between the first and second row. It then continues with `auto` (which is the default, in this case `0pt` gutter) which will be the gutter between all other rows, since it is the last entry in the array.

4.3.6 How to align the contents of the cells in my table?

You can use multiple mechanisms to align the content in your table. You can either use the `table` function's `align` argument to set the alignment for your whole table (or use it in a set rule to set the alignment for

tables throughout your document) or the `align`[3.10.1] function (or `table.cell`'s `align` argument) to override the alignment of a single cell.

When using the `table` function's align argument, you can choose between three methods to specify an alignment[3.10.2]:

- Just specify a single alignment like `right` (aligns in the top-right corner) or `center + horizon` (centers all cell content). This changes the alignment of all cells.
- Provide an array. Typst will cycle through this array for each column.
- Provide a function that is passed the horizontal `x` and vertical `y` coordinate of a cell and returns an alignment.

For example, this travel itinerary right-aligns the day column and left-aligns everything else by providing an array in the `align` argument:

```
#set text(font: "IBM Plex Sans")
#show table.cell.where(y: 0): set text(weight: "bold")

#table(
  columns: 4,
  align: (right, left, left, left),
  fill: (_, y) => if calc.odd(y) { green.lighten(90%) },
  stroke: none,

  table.header[Day][Location][Hotel or Apartment][Activities],
  [1], [Paris, France], [Hôtel de l'Europe], [Arrival, Evening River Cruise],
  [2], [Paris, France], [Hôtel de l'Europe], [Louvre Museum, Eiffel Tower],
  [3], [Lyon, France], [Lyon City Hotel], [City Tour, Local Cuisine Tasting],
  [4], [Geneva, Switzerland], [Lakeview Inn], [Lake Geneva, Red Cross Museum],
  [5], [Zermatt, Switzerland], [Alpine Lodge], [Visit Matterhorn, Skiing],
)
```

Day	Location	Hotel or Apartment	Activities
1	Paris, France	Hôtel de l'Europe	Arrival, Evening River Cruise
2	Paris, France	Hôtel de l'Europe	Louvre Museum, Eiffel Tower
3	Lyon, France	Lyon City Hotel	City Tour, Local Cuisine Tasting
4	Geneva, Switzerland	Lakeview Inn	Lake Geneva, Red Cross Museum
5	Zermatt, Switzerland	Alpine Lodge	Visit Matterhorn, Skiing

However, this example does not yet look perfect — the header cells should be bottom-aligned. Let's use a function instead to do so:

```
#set text(font: "IBM Plex Sans")
#show table.cell.where(y: 0): set text(weight: "bold")

#table(
  columns: 4,
```

```
  align: (x, y) =>
    if x == 0 { right } else { left } +
    if y == 0 { bottom } else { top },
  fill: (_, y) => if calc.odd(y) { green.lighten(90%) },
  stroke: none,

  table.header[Day][Location][Hotel or Apartment][Activities],
  [1], [Paris, France], [Hôtel de l'Europe], [Arrival, Evening River Cruise],
  [2], [Paris, France], [Hôtel de l'Europe], [Louvre Museum, Eiffel Tower],
 // ... remaining days omitted
)
```

Day	Location	Hotel or Apartment	Activities
1	Paris, France	Hôtel de l'Europe	Arrival, Evening River Cruise
2	Paris, France	Hôtel de l'Europe	Louvre Museum, Eiffel Tower
3	Lyon, France	Lyon City Hotel	City Tour, Local Cuisine Tasting
4	Geneva, Switzerland	Lakeview Inn	Lake Geneva, Red Cross Museum
5	Zermatt, Switzerland	Alpine Lodge	Visit Matterhorn, Skiing

In the function, we calculate a horizontal and vertical alignment based on whether we are in the first column (`x == 0`) or the first row (`y == 0`). We then make use of the fact that we can add horizontal and vertical alignments with `+` to receive a single, two-dimensional alignment.

You can find an example of using `table.cell` to change a single cell's alignment on its reference page[3.6.20/4.1].

4.3.7 How to merge cells?

When a table contains logical groupings or the same data in multiple adjacent cells, merging multiple cells into a single, larger cell can be advantageous. Another use case for cell groups are table headers with multiple rows: That way, you can group for example a sales data table by quarter in the first row and by months in the second row.

A merged cell spans multiple rows and/or columns. You can achieve it with the `table.cell`[3.6.20/4.1] function's `rowspan` and `colspan` arguments: Just specify how many rows or columns you want your cell to span.

The example below contains an attendance calendar for an office with in-person and remote days for each team member. To make the table more glanceable, we merge adjacent cells with the same value:

```
#let ofi = [Office]
#let rem = [_Remote_]
#let lea = [*On leave*]

#show table.cell.where(y: 0): set text(
  fill: white,
  weight: "bold",
)
```

```
#table(
  columns: 6 * (1fr,),
  align: (x, y) => if x == 0 or y == 0 { left } else { center },
  stroke: (x, y) => (
    // Separate black cells with white strokes.
    left: if y == 0 and x > 0 { white } else { black },
    rest: black,
  ),
  fill: (_, y) => if y == 0 { black },

  table.header(
    [Team member],
    [Monday],
    [Tuesday],
    [Wednesday],
    [Thursday],
    [Friday]
  ),
  [Evelyn Archer],
    table.cell(colspan: 2, ofi),
    table.cell(colspan: 2, rem),
    ofi,
  [Lila Montgomery],
    table.cell(colspan: 5, lea),
  [Nolan Pearce],
    rem,
    table.cell(colspan: 2, ofi),
    rem,
    ofi,
)
```

Team member	Monday	Tuesday	Wednesday	Thursday	Friday
Evelyn Archer	Office		*Remote*		Office
Lila Montgomery	**On leave**				
Nolan Pearce	*Remote*	Office		*Remote*	Office

In the example, we first define variables with "Office", "Remote", and "On leave" so we don't have to write these labels out every time. We can then use these variables in the table body either directly or in a `table.cell` call if the team member spends multiple consecutive days in office, remote, or on leave.

The example also contains a black header (created with `table`'s `fill` argument) with white strokes (`table`'s `stroke` argument) and white text (set by the `table.cell` set rule). Finally, we align all the content of all table cells in the body in the center. If you want to know more about the functions passed to `align`, `stroke`, and `fill`, you can check out the sections on alignment[3.10.2], strokes[4.3.5/3], and striped tables[4.3.4].

This table would be a great candidate for fully automated generation from an external data source! Check out the section about importing data[4.3.10] to learn more about that.

4.3.8 How to rotate a table?

When tables have many columns, a portrait paper orientation can quickly get cramped. Hence, you'll sometimes want to switch your tables to landscape orientation. There are two ways to accomplish this in Typst:

- If you want to rotate only the table but not the other content of the page and the page itself, use the `rotate` function[3.10.23] with the `reflow` argument set to `true`.
- If you want to rotate the whole page the table is on, you can use the page function[3.10.17] with its `flipped` argument set to `true`. The header, footer, and page number will now also appear on the long edge of the page. This has the advantage that the table will appear right side up when read on a computer, but it also

means that a page in your document has different dimensions than all the others, which can be jarring to your readers.

Below, we will demonstrate both techniques with a student grade book table.

First, we will rotate the table on the page. The example also places some text on the right of the table.

```
#set page("a5", columns: 2, numbering: "— 1 —")
#show table.cell.where(y: 0): set text(weight: "bold")

#rotate(
  -90deg,
  reflow: true,

  table(
    columns: (1fr,) + 5 * (auto,),
    inset: (x: 0.6em,),
    stroke: (_, y) => (
      x: 1pt,
      top: if y <= 1 { 1pt } else { 0pt },
      bottom: 1pt,
    ),
    align: (left, right, right, right, right, left),

    table.header(
      [Student Name],
      [Assignment 1], [Assignment 2],
      [Mid-term], [Final Exam],
      [Total Grade],
    ),
    [Jane Smith], [78%], [82%], [75%], [80%], [B],
    [Alex Johnson], [90%], [95%], [94%], [96%], [A+],
    [John Doe], [85%], [90%], [88%], [92%], [A],
    [Maria Garcia], [88%], [84%], [89%], [85%], [B+],
    [Zhang Wei], [93%], [89%], [90%], [91%], [A-],
    [Marina Musterfrau], [96%], [91%], [74%], [69%], [B-],
  ),
)

#lorem(80)
```

Student Name	Assignment 1	Assignment 2	Mid-term	Final Exam	Total Grade
Jane Smith	78%	82%	75%	80%	B
Alex Johnson	90%	95%	94%	96%	A+
John Doe	85%	90%	88%	92%	A
Maria Garcia	88%	84%	89%	85%	B+
Zhang Wei	93%	89%	90%	91%	A-
Marina Musterfrau	96%	91%	74%	69%	B-

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et.

What we have here is a two-column document on ISO A5 paper with page numbers on the bottom. The table has six columns and contains a few customizations to stroke[4.3.5], alignment and spacing. But the most important part is that the table is wrapped in a call to the `rotate` function with the `reflow` argument being `true`. This will make the table rotate 90 degrees counterclockwise. The reflow argument is needed so that the table's rotation affects the layout. If it was omitted, Typst would lay out the page as if the table was not rotated (`true` might become the default in the future).

The example also shows how to produce many columns of the same size: To the initial `1fr` column, we add an array with five `auto` items that we create by multiplying an array with one `auto` item by five. Note that arrays with just one item need a trailing comma to distinguish them from merely parenthesized expressions.

The second example shows how to rotate the whole page, so that the table stays upright:

```
#set page("a5", numbering: "— 1 —")
#show table.cell.where(y: 0): set text(weight: "bold")

#page(flipped: true)[
  #table(
    columns: (1fr,) + 5 * (auto,),
    inset: (x: 0.6em,),
    stroke: (_, y) => (
      x: 1pt,
      top: if y <= 1 { 1pt } else { 0pt },
      bottom: 1pt,
    ),
    align: (left, right, right, right, right, left),

    table.header(
      [Student Name],
      [Assignment 1], [Assignment 2],
      [Mid-term], [Final Exam],
      [Total Grade],
    ),
    [Jane Smith], [78%], [82%], [75%], [80%], [B],
    [Alex Johnson], [90%], [95%], [94%], [96%], [A+],
    [John Doe], [85%], [90%], [88%], [92%], [A],
    [Maria Garcia], [88%], [84%], [89%], [85%], [B+],
    [Zhang Wei], [93%], [89%], [90%], [91%], [A-],
    [Marina Musterfrau], [96%], [91%], [74%], [69%], [B-],
  )

  #pad(x: 15%, top: 1.5em)[
    = Winter 2023/24 results
    #lorem(80)
  ]
]
```

Student Name	Assignment 1	Assignment 2	Mid-term	Final Exam	Total Grade
Jane Smith	78%	82%	75%	80%	B
Alex Johnson	90%	95%	94%	96%	A+
John Doe	85%	90%	88%	92%	A
Maria Garcia	88%	84%	89%	85%	B+
Zhang Wei	93%	89%	90%	91%	A-
Marina Musterfrau	96%	91%	74%	69%	B-

Winter 2023/24 results

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et.

Here, we take the same table and the other content we want to set with it and put it into a call to the page[3.10.17] function while supplying `true` to the `flipped` argument. This will instruct Typst to create new pages with width and height swapped and place the contents of the function call onto a new page. Notice how the page number is also on the long edge of the paper now. At the bottom of the page, we use the pad[3.10.16] function to constrain the width of the paragraph to achieve a nice and legible line length.

4.3.9 How to break a table across pages?

It is best to contain a table on a single page. However, some tables just have many rows, so breaking them across pages becomes unavoidable. Fortunately, Typst supports breaking tables across pages out of the box. If you are using the `table.header`[3.6.20/4.4] and `table.footer`[3.6.20/4.5] functions, their contents will be repeated on each page as the first and last rows, respectively. If you want to disable this behavior, you can set `repeat` to `false` on either of them.

If you have placed your table inside of a figure[3.6.8], it becomes unable to break across pages by default. However, you can change this behavior. Let's take a look:

```
#set page(width: 9cm, height: 6cm)
#show table.cell.where(y: 0): set text(weight: "bold")
#show figure: set block(breakable: true)

#figure(
  caption: [Training regimen for Marathon],
  table(
    columns: 3,
    fill: (_, y) => if y == 0 { gray.lighten(75%) },

    table.header[Week][Distance (km)][Time (hh:mm:ss)],
    [1], [5],  [00:30:00],
    [2], [7],  [00:45:00],
    [3], [10], [01:00:00],
    [4], [12], [01:10:00],
    [5], [15], [01:25:00],
```

```
    [6], [18], [01:40:00],
    [7], [20], [01:50:00],
    [8], [22], [02:00:00],
    [...], [...], [...],
    table.footer[_Goal_][_42.195_][_02:45:00_],
  )
)
```

Week	Distance (km)	Time (hh:mm:ss)
1	5	00:30:00
2	7	00:45:00
3	10	01:00:00
4	12	01:10:00
5	15	01:25:00
6	18	01:40:00
Goal	*42.195*	*02:45:00*

Week	Distance (km)	Time (hh:mm:ss)
7	20	01:50:00
8	22	02:00:00
…	…	…
Goal	*42.195*	*02:45:00*

Table 1: Training regimen for Marathon

A figure automatically produces a block[3.10.4] which cannot break by default. However, we can reconfigure the block of the figure using a show rule to make it `breakable`. Now, the figure spans multiple pages with the headers and footers repeating.

4.3.10 How to import data into a table?

Often, you need to put data that you obtained elsewhere into a table. Sometimes, this is from Microsoft Excel or Google Sheets, sometimes it is from a dataset on the web or from your experiment. Fortunately, Typst can load many common file formats[3.13], so you can use scripting to include their data in a table.

The most common file format for tabular data is CSV. You can obtain a CSV file from Excel by choosing "Save as" in the *File* menu and choosing the file format "CSV UTF-8 (Comma-delimited) (.csv)". Save the file and, if you are using the web app, upload it to your project.

In our case, we will be building a table about Moore's Law. For this purpose, we are using a statistic with how many transistors the average microprocessor consists of per year from Our World in Data[A]. Let's start by pressing the "Download" button to get a CSV file with the raw data.

Be sure to move the file to your project or somewhere Typst can see it, if you are using the CLI. Once you did that, we can open the file to see how it is structured:

[A] https://ourworldindata.org/grapher/transistors-per-microprocessor

```
Entity,Code,Year,Transistors per microprocessor
World,OWID_WRL,1971,2308.2417
World,OWID_WRL,1972,3554.5222
World,OWID_WRL,1974,6097.5625
```

The file starts with a header and contains four columns: Entity (which is to whom the metric applies), Code, the year, and the number of transistors per microprocessor. Only the last two columns change between each row, so we can disregard "Entity" and "Code".

First, let's start by loading this file with the `csv`[3.13.3] function. It accepts the file name of the file we want to load as a string argument:

```
#let moore = csv("moore.csv")
```

We have loaded our file (assuming we named it `moore.csv`) and bound it[3.3.3] to the new variable `moore`. This will not produce any output, so there's nothing to see yet. If we want to examine what Typst loaded, we can either hover the name of the variable in the web app or print some items from the array:

```
#let moore = csv("moore.csv")

#moore.slice(0, 3)
```

```
(
  (
    "Entity",
    "Code",
    "Year",
    "Transistors per microprocessor",
  ),
  ("World", "OWID_WRL", "1995",
"9646616"),
  ("World", "OWID_WRL", "1996",
"9646616"),
)
```

With the arguments `(0, 3)`, the `slice`[3.5.2/3.9] method returns the first three items in the array (with the indices 0, 1, and 2). We can see that each row is its own array with one item per cell.

Now, let's write a loop that will transform this data into an array of cells that we can use with the table function.

```
#let moore = csv("moore.csv")

#table(
  columns: 2,
  ..for (.., year, count) in moore {
    (year, count)
  }
)
```

Year	Transistors per microprocessor
1995	9646616
1996	9646616
1997	9646616
1998	15261378
1999	21673922
2000	37180264
2001	42550656
2002	220673400
2003	220673400
2004	273842000
2005	305052770
2006	582941600
2007	805842200
2008	805842200
2009	2308241400
2010	2308241400
2011	2600000000
2012	2600000000
2013	5000000000
2014	5700000000
2016	8000000000
2017	19200000000
2018	21100000000
2019	39500000000
2020	39500000000
2021	58200000000

The example above uses a for loop that iterates over the rows in our CSV file and returns an array for each iteration. We use the for loop's destructuring[3.3.3] capability to discard all but the last two items of each row. We then create a new array with just these two. Because Typst will concatenate the array results of all the loop iterations, we get a one-dimensional array in which the year column and the number of transistors alternate. We can then insert the array as cells. For this we use the spread operator[3.5.1/2] (`..`). By prefixing an array, or, in our case an expression that yields an array, with two dots, we tell Typst that the array's items should be used as positional arguments.

Alternatively, we can also use the `map`[3.5.2/3.15], `slice`[3.5.2/3.9], and `flatten`[3.5.2/3.23] array methods to write this in a more functional style:

```
#let moore = csv("moore.csv")

#table(
   columns: 2,
   ..moore.map(m => m.slice(2, 4)).flatten(),
)
```

This example renders the same as the previous one, but we first load the CSV and then transform each row using `map`. The function we pass to `map` is applied to each row of the data and returns a new array that replaces the original row. Here, we use `.slice(2, 4)` to extract only the third and fourth column, since these are the ones we want to keep. Because `moore` is a two-dimensional array (each row is itself an array), the result of mapping is still a nested array. The `flatten` function converts this nested structure into a one-

dimensional array, which is required when spreading the data into the `table` function. Finally, we explicitly specify `columns: 2` because we are keeping exactly two columns from each row.

Now that we have nice code for our table, we should try to also make the table itself nice! The transistor counts go from millions in 1995 to trillions in 2021 and changes are difficult to see with so many digits. We could try to present our data logarithmically to make it more digestible:

```
#let moore = csv("moore.csv")
#let moore-log = moore.slice(1).map(m => {
  let (.., year, count) = m
  let log = calc.log(float(count))
  let rounded = str(calc.round(log, digits: 2))
  (year, rounded)
})

#show table.cell.where(x: 0): strong

#table(
   columns: moore-log.first().len(),
   align: right,
   fill: (_, y) => if calc.odd(y) { rgb("D7D9E0") },
   stroke: none,

   table.header[Year][Transistor count ($log_10$)],
   table.hline(stroke: rgb("4D4C5B")),
   ..moore-log.flatten(),
)
```

Year	Transistor count ($\log_{10}$)
1995	6.98
1996	6.98
1997	6.98
1998	7.18
1999	7.34
2000	7.57
2001	7.63
2002	8.34
2003	8.34
2004	8.44
2005	8.48
2006	8.77
2007	8.91
2008	8.91
2009	9.36
2010	9.36
2011	9.41
2012	9.41
2013	9.7
2014	9.76
2016	9.9
2017	10.28
2018	10.32
2019	10.6
2020	10.6
2021	10.76

In this example, we first drop the header row from the data since we are adding our own. Then, we discard all but the last two columns as above. We do this by destructuring[3.3.3] the array `m`, discarding all but the two last items. We then convert the string in `count` to a floating point number, calculate its logarithm and store it in the variable `log`. Finally, we round it to two digits, convert it to a string, and store it in the variable `rounded`. Then, we return an array with `year` and `rounded` that replaces the original row. In our table, we have added our custom header that tells the reader that we've applied a logarithm to the values. Then, we spread the flattened data as above.

We also styled the table with stripes[4.3.4], a horizontal line[4.3.5/1] below the first row, aligned[4.3.6] everything to the right, and emboldened the first column. Click on the links to go to the relevant guide sections and see how it's done!

4.3.11 What if I need the table function for something that isn't a table?

Tabular layouts of content can be useful not only for matrices of closely related data, like shown in the examples throughout this guide, but also for presentational purposes. Typst differentiates between grids that are for layout and presentational purposes only and tables, in which the arrangement of the cells itself conveys information.

To make this difference clear to other software and allow templates to heavily style tables, Typst has two functions for grid and table layout:

- The `table`[3.6.20] function explained throughout this guide which is intended for tabular data.
- The `grid`[3.10.10] function which is intended for presentational purposes and page layout.

Both elements work the same way and have the same arguments. You can apply everything you have learned about tables in this guide to grids. There are only three differences:

- You'll need to use the `grid.cell`[3.10.10/6.1], `grid.vline`[3.10.10/6.3], and `grid.hline`[3.10.10/6.2] elements instead of `table.cell`[3.6.20/4.1], `table.vline`[3.6.20/4.3], and `table.hline`[3.6.20/4.2].
- The grid has different defaults: It draws no strokes by default and has no spacing (`inset`) inside of its cells.
- Elements like `figure` do not react to grids since they are supposed to have no semantical bearing on the document structure.

4.4 Accessibility Guide

Making a document accessible means that it can be used and understood by everyone. That not only includes people with permanent or temporary disabilities, but also those with different devices or preferences. To underscore why accessibility is important, consider that people might read your document in more contexts than you expected:

- A user may print the document on paper
- A user may read your document on a phone, with reflow in their PDF reader enabled
- A user may have their computer read the document back to them
- A user may ask artificial intelligence to summarize your document for them
- A user may convert your document to another file format like HTML that is more accessible to them

To accommodate all of these people and scenarios, you should design your document for **Universal Access.** Universal Access is a simple but powerful principle: instead of retrofitting a project for accessibility after the fact, design from the beginning to work for the broadest possible range of people and situations. This will improve the experience for all readers!

Typst can help you to create accessible files that read well on screen readers, look good even when reflowed for a different screen size, and pass automated accessibility checkers. However, to create accessible files, you will have to keep some rules in mind. This guide will help you learn what issues impact accessibility, how to design for Universal Access, and what tools Typst gives you to accomplish this. Much of the guidance here applies to all export targets, but the guide focuses on PDF export. Notable differences to HTML export are called out.

4.4.1 Basics of Accessibility

Accessible files allow software to do more with them than to just render them. Instead, your computer can understand what each part of the document is supposed to represent and use this information to present the document to the user.

This information is consumed by different software to provide access. When exporting a PDF from Typst, the *PDF viewer* (sometimes also called a reader) will display the document's pages just as you designed them with Typst's preview. Some people rely on *Assistive Technology* (AT) such as screen readers, braille displays, screen magnifiers, and more for consuming PDF files. In that case, the semantic information in the file is used to adapt the contents of a file into spoken or written text, or into a different visual representation. Other users will make the PDF viewer reflow the file to create a layout similar to a web page: The content will fit the viewport's width and scroll continuously. Finally, some users will repurpose the PDF into another format, for example plain text for ingestion into a Large Language Model (LLM) or HTML. A special form of repurposing is copy and paste where users use the clipboard to extract content from a file to use it in another application.

Accessibility support differs based on viewer and AT. Some combinations work better than others. In our testing, Adobe Acrobat[A] paired with NVDA[B] on Windows and VoiceOver[C] on macOS provided the richest accessibility support. Paired with HTML export, browsers provide a more consistent baseline of accessibility when compared to PDF readers.

Only PDF and HTML export produce accessible files. Neither PNGs nor SVGs are accessible on their own. Both formats can be used in an accessible larger work by providing a textual representation[4.4.7].

4.4.2 Maintaining semantics

To add correct semantic information for AT and repurposing to a file, Typst needs to know what semantic role each part of the file plays. For example, this means that a heading in a compiled PDF should not just be text that is large and bold. Instead, the file should contain the explicit information (known as a *tag*) that a particular text makes up a heading. A screen reader will then announce it as a heading and allow the user to navigate between headings.

[A] https://www.adobe.com/acrobat.html
[B] https://www.nvaccess.org/download/
[C] https://support.apple.com/guide/voiceover/welcome/mac

If you are using Typst idiomatically, using the built-in markup and elements, Typst automatically adds tags with rich semantic information to your files. Let's take a look at two code examples:

```
// ❌ Don't do this
#text(
  size: 16pt,
  weight: "bold",
)[Heading]
```

Heading

```
// ✅ Do this
#show heading: set text(size: 16pt)
= Heading
```

Heading

Both of these examples look the same. They both contain the text "Heading" in boldface, sized at 16 point. However, only the second example is accessible. By using the heading markup, Typst knows that the semantic meaning of this text is that of a heading and can propagate that information to the final PDF. In the first example, it just knows that it should use boldface and larger type on otherwise normal text and cannot make the assumption that you meant that to be a heading and not a stylistic choice or some other element like a quote.

Using semantics is not limited to headings. Here are a few more examples for elements you should use:

- Use underscores / `emph`[3.6.7] instead of the `text`[3.7.12] function to make text emphasized
- Use stars / `strong`[3.6.19] instead of the text function to make text carry strong emphasis
- Use lists (`list`[3.6.3], `enum`[3.6.12], `terms`[3.6.21]) instead of normal text with newlines when working with itemized or ordered content
- Use `quote`[3.6.17] for inline and block quotes
- Use the built-in `bibliography`[3.6.2] and `cite`[3.6.4] functions instead of manually printing a bibliography
- Use labels and `ref`[3.6.18] or `@references` to reference other parts of your documents instead of just typing out a reference
- Use the `caption` argument of the `figure` element[3.6.8/6.1] to provide captions instead of adding them as text below the function call

If you want to style the default appearance of an element, do not replace it with your own custom function. Instead, use set[3.2.1], show-set, and show rules[3.2.2] to customize its appearance. Here is an example on how you can change how strong emphasis looks in your document:

```
// Change how text inside of strong emphasis looks
#show strong: set text(tracking: 0.2em, fill: blue, weight: "black")

When setting up your tents, *never forget* to secure the pegs.
```

When setting up your tents, **n e v e r f o r g e t** to secure the pegs.

The show-set rule completely changes the default appearance of the `strong`[3.6.19] element, but its semantic meaning will be preserved. If you need even more customization, you can provide show rules with fully custom layout code and Typst will still retain the semantic purpose of the element.

4.4.3 Reading order

For AT to read the contents of a document in the right order and for repurposing applications, accessible files must make their reading order explicit. This is because the logical reading order can differ from layout order. Floating figures are a common example for such a difference: A figure may be relevant to a paragraph in the center of a page but appear at the top or bottom edge. In non-accessible files, PDF readers and AT have to assume that layout order equals the logical reading order, often leading to confusion for AT users.

When the reading order is well-defined, screen readers read a footnote or a floating figure immediately where it makes sense.

Fortunately, Typst markup already implies a single reading order. You can assume that Typst documents will read in the order that content has been placed in the markup. For most documents, this is good enough. However, when using the `place`[3.10.19] and `move`[3.10.15] function or floating figures[3.6.8/5.3], you must pay special attention to place the function call at an appropriate spot in the logical reading order in markup, even if this has no consequence on the layout. Just ask yourself where you would want a screen reader to announce the content that you are placing.

4.4.4 Layout containers

Typst provides some layout containers like `grid`[3.10.10], `stack`[3.10.28], `box`[3.10.5], `columns`[3.10.7], and `block`[3.10.4] to visually arrange your content. None of these containers come with any semantic meaning attached. Typst will conserve some of these containers during PDF reflow while other containers will be discarded.

When designing for Universal Access, you need to be aware that AT users often cannot view the visual layout that the container creates. Instead, AT will just read its contents, so it is best to think about these containers as transparent in terms of accessibility. For example, a grid's contents will just be read out flatly, in the order that you have added the cells in the source code. If the layout you created is merely visual and decorative, this is fine. If, however, the layout carries semantic meaning that is apparent to a sighted user viewing the file in a regular PDF reader, it is not accessible. Instead, create an alternative representation of your content that leverages text or wrap your container in the `figure`[3.6.8] element to provide an alternative textual description.

Do not use the grid container to represent tabular data. Instead, use `table`[3.6.20]. Tables are accessible to AT users: their AT will allow them to navigate the table two-dimensionally. Tables are conserved during reflow and repurposing. When creating tables, use the `table.header`[3.6.20/4.4] and `table.footer`[3.6.20/4.5] elements to mark up the semantic roles of individual rows. The table documentation contains an accessibility section[3.6.20/2] with more information on how to make your tables accessible. Keep in mind that while AT users can access tables, it is often cumbersome to them: Tables are optimized for visual consumption. Being read the contents of a set of cells while having to recall their row and column creates additional mental load. Consider making the core takeaway of the table accessible as text or a caption elsewhere.

Likewise, if you use functions like `rotate`[3.10.23], `scale`[3.10.24], and `skew`[3.10.25], take care that this transformation either has no semantic meaning or that the meaning is available to AT users elsewhere, i.e. in figure alt text[4.4.7] or a caption.

4.4.5 Artifacts

Some things on a page have no semantic meaning and are irrelevant to the content of a document. We call these items *artifacts.* Artifacts are hidden from AT and repurposing and will vanish during reflow. Here are some examples for artifacts:

- The hyphens inserted by automatic hyphenation at the end of a line
- The headers and footers on each page
- A purely decorative page background image

In general, every element on a page must either have some way for AT to announce it or be an artifact for a document to be considered accessible.

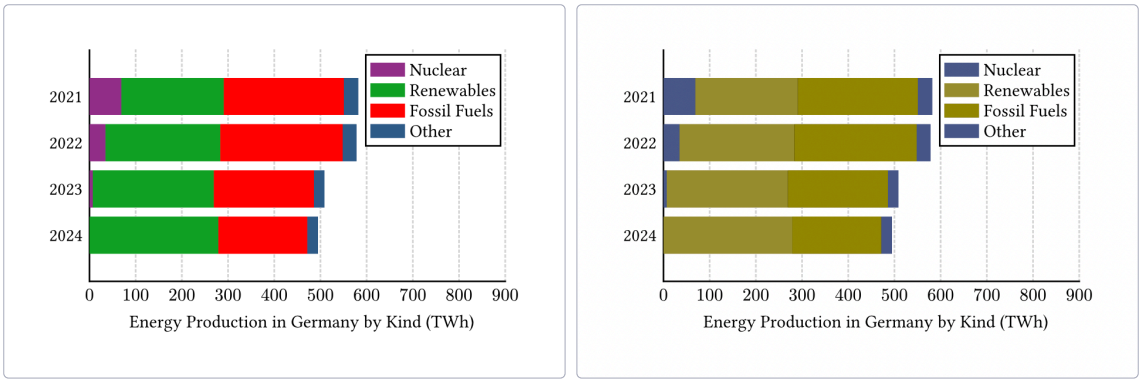
Typst automatically tags many layout artifacts such as headers, footers, page back- and foregrounds, and automatic hyphenation as artifacts. However, if you'd like to add purely decorative content to your document, you can use the `pdf.artifact`[3.14.4] function to mark a piece of content as an artifact. If you are unsure if you should mark an element as an artifact, ask yourself this: Would it be purely annoying if a screen reader announced the element to you? Then, it may be an artifact. If, instead, it could be useful to have it announced, then it is not an artifact.

For technical reasons, once in an artifact, content cannot become semantic again. To stack artifacts and semantic contents, use `place`[3.10.19] to move the contents on top of one another.

Please note that Typst will mark shapes and paths like `square`[3.11.11] and `circle`[3.11.2] as artifacts while their content will remain semantically relevant and accessible to AT. If your shapes have a semantic meaning, please wrap them in the `figure`[3.6.8] element to provide an alternative textual description.

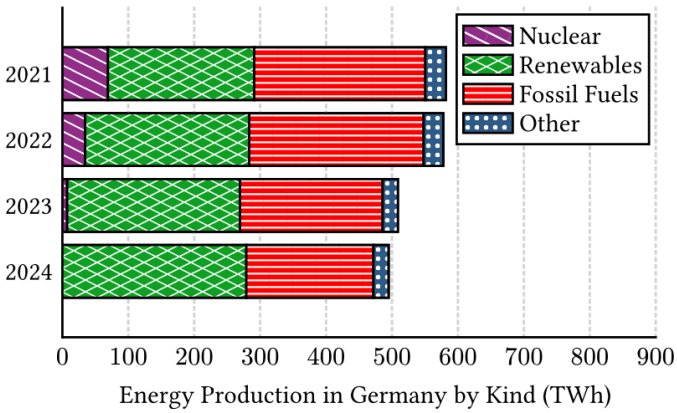
4.4.6 Color use and contrast

Universal Access not only means that your documents works with AT, reflow, and repurposing, but also that visual access is possible to everyone, including people with impaired eyesight. Not only does aging often come with worse sight, a significant portion of people have problems differentiating color: About 8% of men and 0.5% of women are color blind.



This means that color must not be the only way you make information accessible to sighted users in your documents. As an example, consider a stacked bar chart with multiple colored segments per bar. Our example shows a chart of the domestic energy production in Germany by kind[A]. In the picture, you can see the chart as it would normally appear and a simulation of how it would appear to people with deuteranopia-type color blindness. You can see that the two pairs of the first and last segment both look blue and the center pair looks yellow-ish. The first challenge for the colorblind user is thus to make out the boundary of the "Renewable" and "Fossil Fuels" bar. Then, they must keep track of which bar is which by only their order, adding to their mental load. A way to make this chart even less accessible would be to make the order of segments not match their order in the legend.

How can we improve the chart? First, make sure that no information is solely communicated through color use. One possible way to do this by adding a pattern to each bar. Then, we can help the user make out the boundaries of each segment by adding a high-contrast border. Then, our chart could look something like this:

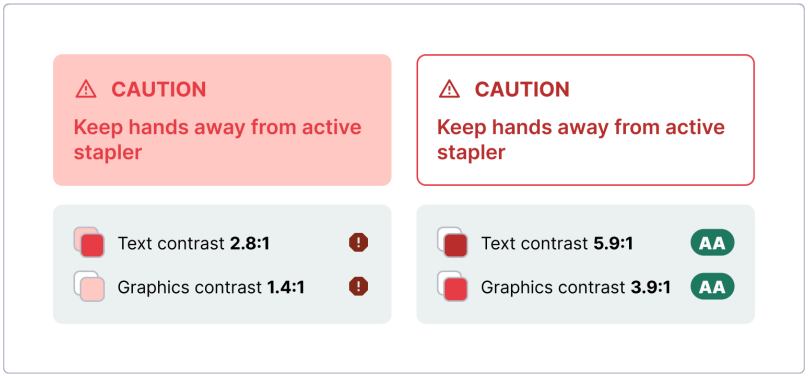


[A]Dataset from the German Federal Statistics Authority (Statistisches Bundesamt, Destatis). "Bruttostromerzeugung nach Energieträgern in Deutschland ab 1990", 2025, available under the *Data licence Germany – attribution – version 2.0*.

This could be further improved by choosing colors that are differentiable to people afflicted by common colorblindness types. You could also iterate on the design by choosing two-tone patterns, aligning them to the bars, or changing font use.

You can check your design in the web app by using the built-in color blindness simulator. To use it, open the "View" menu and select the desired mode in the "Simulate color blindness" menu. You can also use other tools on the web to simulate the color perception of various color blindnesses[A] if you are not using our web app.

Also consider the color contrast between background and foreground. For example, when you are using light gray text for footnotes, they could become hard to read. Another situation that often leads to low contrast is superimposing text on an image.



In our example, we can see two designs for callout boxes. Because these boxes aim to help the user avoid a hazard, it is paramount that they can actually read them. However, in the first box, the background is fairly light, making it hard to make out the box. Worse, the red text is difficult to read on the light red background. The text has a 2.8:1 contrast ratio, failing the bar of 4.5:1 contrast the Web Content Accessibility Guidelines (WCAG) set. Likewise, the box has an 1.4:1 contrast ratio with the white page background, falling short of the 3:1 threshold for graphical objects.

Colors in the second example have been adjusted to meet WCAG AA color contrast thresholds. It should be markedly easier to read the text in the box, even if you have good vision!

There are tools to compare how much contrast a pair of colors has[B] as foreground and background. The most common one is the WCAG color contrast ratio. For a given font size, a color pair may either fail the test, get to the AA level, or reach the higher AAA level. Aim for at least AA contrast for all your color pairings.

Content	AA Ratio	AAA Ratio
Large text (≥18pt, or bold and ≥14pt)	3:1	4.5:1
Small text	4.5:1	7:1
Non-text content	3:1	3:1

Note that common accessibility frameworks like WCAG make an exception for purely decorative text and logos: Due to their graphic character, they can have contrast ratios that fail to achieve AA contrast ratio.

4.4.7 Textual representations

To support AT use and some repurposing workflows, all elements with a semantic meaning must have a textual representation. Think about it in terms of Universal Access: If an item is not an artifact[4.4.5], it has a semantic meaning. If, however, AT cannot ingest the item, the full semantic meaning of a document is not available to AT users. Hence, to provide Universal Access, use the mechanisms built into Typst to provide alternative representations.

When you add an image, be sure to use the `alt` argument of the image function[3.11.7/2.5] to describe what's visible in the image. This alternative description (sometimes known as alt text) should describe the gist of

[A] https://daltonlens.org/colorblindness-simulator
[B] https://webaim.org/resources/contrastchecker/

the image: Think about how you would describe the image to a friend if you called them on the phone. To write good alternative descriptions, consider the context in which the image appears:

```
#image("heron.jpg", alt: "?")

Herons have feet with interdigital
webbing, allowing for good mobility
when swimming, and wings that span
up to 2.3 m.
```



Herons have feet with interdigital webbing, allowing for good mobility when swimming, and wings that span up to 2.3 m.

What could be a good alternative description for this image[A]? Let's consider a few examples for what *not* to do:

- `"Image of a heron"`
  ❌ The screen reader will already announce the image on its own, so saying this is an image is redundant. In this example, the AT user would hear "Image, Image of a heron".
- `"A bird"`
  ❌ The alternative description is not specific enough. For example, it is relevant to a user that the image depicts a heron and both its feet and wings are visible.
- `"Gray heron in flight. Picture by Makasch1966 on Wikimedia Commons, CC Attribution 4.0 International license"`
  ❌ The alternative description should not include details not visible in the image, such as attribution, jokes, or metadata. Keep in mind that it is not accessible to sighted users. That information belongs elsewhere.
- `"Gray heron flying low, heading from the right to left. Its feet are extended and slightly point downwards, touching a blurred horizon where a dark forest becomes visible. The bird's wings are extended and arc upwards. There are out-of-focus branches visible in the lower left corner of the image."`
  ❌ The alternative description is too verbose. Use your discretion and determine how important the image is to the content. Think about how long a sighted user would realistically look at the image; your alt text should take about the same effort to 'consume.' For example, the anatomic description contained above could be appropriate for a longer discussion in a zoology textbook while the compositional information is useful when writing about photography. The context the example image comes with is relatively short, so write a more brief description.

Instead, in the given example, you could use this alternative text:

- `"Heron in flight with feet and wings spread"`
  ✅ This alternative description describes the image, is relevant to the context, and matches its brevity.

There are resources available on the web to learn more about writing good alternative descriptions[B]. The requirement to add alternative text to images applies to all image formats. Typst does not currently retain the tags of a PDF image in the compiled document, even if the PDF image file on its own was accessible.

Do not use images of text; likewise, do not use the path operations to draw text manually. Typst will not be able to process text in any images to make it accessible in the same way that native text is. There is one exception to this rule: Use an image of text when the appearance of the text is essential to the semantic meaning of the document and cannot be reproduced with Typst natively. In that case, you must describe both the textual content and the essential visual characteristics in the alternative description.

[A] https://commons.wikimedia.org/wiki/File:Reiher_im_Flug.jpg
[B] https://webaim.org/techniques/alttext/

Like the image function, the figure function has a `alt` attribute[3.6.8/5.2]. When you use this attribute, many screen readers and other AT will not announce the content inside of the figure and instead just read the alternative description. Your alternative description must be comprehensive enough so that the AT user does not need to access the body of the figure. Only use the alternative description if the content of the figure are not otherwise accessible. For example, do not use the `alt` attribute of a figure if it contains a `table` element, but do use it if you used shapes within that come with a semantic meaning. If you specify both `alt` and `caption`, both will be read by AT. When your figure contains an image, set the alternative description on the image itself[3.11.7/2.5], not on the figure. Do not set both, as the image description would be overridden by the figure description.

```
#figure(
  alt: "Star with a blue outline",
  curve(
    stroke: blue,
    curve.move((25pt, 0pt)),
    curve.line((10pt, 50pt)),
    curve.line((50pt, 20pt)),
    curve.line((0pt, 20pt)),
    curve.line((40pt, 50pt)),
    curve.close(),
  ),
)
```

Finally, you can specify an alternative description on math using `math.equation`[3.8.15]. Describe your formula as if read out loud in natural language. Currently, adding an alternative description is required for accessible math for all export formats. Not adding an alternative description for your formula will result in a failure of PDF/UA-1 export. In the future, Typst will automatically make math accessible in HTML and PDF 2.0 by leveraging MathML technology.

```
#math.equation(
  alt: "a squared plus b squared equals c squared",
  block: true,
  $ a^2 + b^2 = c^2 $,
)
```

Another element that represents itself as text are links. It is best to avoid non-descriptive link texts such as *here* or *go*. These link texts also hurt Search Engine Optimization (SEO) if that is a consideration for your document. Instead, try to have the link contain text about where it is pointing to. Note that, unless you are aiming for the highest level of accessibility, it is also okay if the link itself is not descriptive but its purpose can be understood from the content immediately surrounding it.

4.4.8 Natural Language

In order for screen readers to pronounce your document correctly and translation software to work properly, you must indicate in which natural language your document is written. Use the rule `#set text(lang: "..")`[3.7.12/2.16] at the very start of your document or your template's capability to set a language. If you do not do so, Typst will assume that your content is in English. The natural language you choose not only impacts accessibility, but also how Typst will apply hyphenation, what typesetting conventions are applied, the labels of figures and references, and, in the web app, what language is used for spellcheck.

If you are using a language with significant variation between regions, such as Chinese or English, also use the `region` argument[3.7.12/2.17]. For example, Chinese as it is spoken in Hong Kong would look like this:

```
#set text(lang: "zh", region: "HK")
```

To specify your language, use ISO 639 codes. For regions, use the ISO 3166-1 alpha-2[A] code. ISO 639 contains three variants, one for two-letter language codes like "de" for German (ISO 639-1)[B] and two for

[A] https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
[B] https://en.wikipedia.org/wiki/List_of_ISO_639_language_codes

three-letter language codes like "deu" (ISO 639-2[A] and ISO 639-3[B]). If your language has a two-letter ISO 639-1 code, always prefer using that. ISO 639-2 and 639-3 share most codes, but there are some differences. When your language code differs between the two standards, use ISO 639-2 when exporting to PDF 1.7 (Typst's default) and below and ISO 639-3 for export to PDF 2.0 and HTML.

There are three special language codes defined by both ISO 639-2 and ISO 639-3 that you can use when providing a normal language code is difficult:

- `zxx` for text that is not in a natural language
- `und` for text for which you cannot determine the natural language
- `mis` for text in languages that have not been assigned a language code

If your document contains text in multiple languages, you can use the text function or a scoped text set rule to enclose instances of other languages:

```
This is #text(lang: "fr")[français].

#[
  #set text(lang: "es")
  Este es un fragmento más largo
  del texto en español.
]
```

This is français.

Este es un fragmento más largo del texto en español.

4.4.9 Document Title and Headings

Titling your document makes it easier to retrieve it and to navigate between it and other documents, both for AT users and regular users of PDF viewers. This is why accessibility standards such as WCAG and PDF/UA require you to set a machine-readable title for your document.

To do so in Typst, place this set rule in your document before any content:

```
#set document(title: [GlorboCorp Q1 2023 Revenue Report])
```

This will set the title in the document's metadata[3.6.6/3.3] and in the title bar of the PDF viewer or web browser. If this results in an error when using a template, consider whether your template may provide an alternative way to set the document title.

Most likely, you will also want to include the title visibly in your document. To do so, use the `title`[3.6.22] element. When you add a call to the title element without any arguments, it will print the contents of what you set as the document's title. Alternatively, you can customize the title by passing content as the positional body argument. Do not use the title element more than once in your document.

Never use a heading for your document title; instead, use the title element. Should you have experience with HTML, it is important to remember that the semantics of the heading element in Typst differ from HTML headings. It is encouraged to use multiple first-level headings for section headings in Typst documents. When exporting to HTML, a title[3.6.22] will be serialized as a `h1` tag while a first-level heading[3.6.10/5.1] will be serialized as a `h2` tag. In PDF export, the title and headings will be correctly tagged based on the PDF version targeted.

It is important that the sequence of headings you use is sequential: Never skip a heading level when going deeper. This means that a third-level heading must be followed by a heading of level four or lower, but never a heading of level five or higher.

```
// ❌ Don't do this:
= First level heading
=== Third level heading
```

Note that in order to pass the automated accessibility check in Adobe Acrobat[C], documents with 21 pages or more must contain outlined headings.

[A] https://en.wikipedia.org/wiki/List_of_ISO_639-2_codes
[B] https://en.wikipedia.org/wiki/List_of_ISO_639-3_codes
[C] https://helpx.adobe.com/acrobat/using/create-verify-pdf-accessibility.html#Bookmarks

4.4.10 Accessibility Standards and Legislation

Typst can help you to assert that your document is accessible by checking it against international standards. For PDF export, there are multiple standards for accessible files, most notably the PDF/UA standard. Its first part (PDF/UA-1) is already supported by Typst while support for the second part (PDF/UA-2) is planned for the future. Below, you can find an explanation of all relevant standards:

- **Tagged PDF:** Tagged PDFs contain machine-readable data about the semantic structure of a document that AT can parse. Typst will write Tagged PDFs by default, but keep in mind that Typst can only write appropriate tags if it knows about the semantic structure of your document. Refer to the Section *Maintaining semantics*[4.4.2] to learn how to use Typst's elements to communicate semantics. To provide Universal Access, you are also responsible to provide textual representation of non-text content yourself.
- **PDF/UA-1:** The PDF/UA standard explains how to write a PDF 1.7 file optimized for Universal Access. It implies Tagged PDF, enforces alternative descriptions for images and mathematics, requires a document title, and introduces rules how document contents like tables should be structured. If you are following this guide, you are already avoiding most of the compiler errors that can occur during PDF/UA-1 export.
- **PDF/UA-2:** There is also the more recent part PDF/UA-2 that targets PDF 2.0 files. It improves accessibility for mathematics and some semantic elements. Support for PDF/UA-2 not yet available in Typst, but planned.
- **Well Tagged PDF (WTPDF):** This is an industry standard that is very similar to PDF/UA-2. Like PDF/UA-2, it is not currently supported by Typst. Originally, it was drafted because both parts of the PDF/UA specification were only available at a high cost from the International Standards Organization. Hence, WTPDF[A] was designed so that all conforming files can also declare conformance with PDF/UA-2. By now, both parts of the PDF/UA specification are available free of charge[B], decreasing the relevance of WTPDF.
- **PDF/A-1a:** The PDF/A standard describes how to produce PDF files that are well-suited for archival. Parts one to three of the PDF/A standard feature multiple conformance levels. The strictest conformance level A contains rules for accessibility as only files meeting those rules remain usable to the broadest range of people in the far future. Level A implies conformance with Tagged PDF and forces you to provide alternative descriptions for images. Other PDF/A rules not relating to accessibility, e.g. about transparency, colors, and more also apply. This part of the PDF/A standard is based on the outdated PDF 1.4 specification. Only use it if your venue requires it or if you need a very compatible file. Otherwise, PDF/UA-1 and the second and third part of PDF/A provide better alternatives.
- **PDF/A-2a** and **PDF/A-3a:** Like the first part of PDF/A, these standards focus on creating files suitable for archival and long-term storage. Both of these standards target the newer PDF version 1.7 instead of PDF 1.4. Here too, the strictest conformance level A contains rules for accessibility. In addition to the rules in PDF/A-1a, these standards disallow the use of characters in the Unicode Private Use Area[C] whose meaning is not universally defined. Improvements over PDF/A-1 include the ability to use transparency and better reflow. When choosing between these two parts of the PDF/A standard, choose PDF/A-2a unless you need to attach[3.14.5] other files. Note that conformance level A has been removed from PDF/A-4 in favor of the dedicated PDF/UA standard.

The PDF reference page[3.14.2] contains more information about each supported standard. To enable either PDF/UA, PDF/A-2a, or PDF/A-3a, use the appropriate flag in the CLI[3.14.1/1] or use the export dropdown and click on PDF in the web app. You can combine compatible PDF/A and PDF/UA standards. For accessibility-focused documents, we recommend combining PDF/UA-1 with PDF/A-2a or PDF/A-3a.

When you select one of these standards for PDF export, Typst will detect if you are in violation of their rules and fail the export with a descriptive error message. For the strictest accessibility check currently available, choose PDF/UA-1. Do not disable tagging unless you have a good reason, as tags provide a baseline of accessibility across all documents you export.

Maybe you already noticed that some of the factors that go into Universal Access are hard to check automatically. For example, Typst will currently not automatically check that your color contrasts are sufficient or whether the configured natural language matches the actual natural language (although the amount of

[A] https://pdfa.org/wtpdf/
[B] https://pdfa.org/sponsored-standards/
[C] https://en.wikipedia.org/wiki/Private_Use_Areas

spellcheck errors should provide a hint if you are using the web app). There are two international standards that address some of these human factors in more detail:

- The **Web Content Accessibility Guidelines (WCAG)**[A]: Designed by the W3C, a big international consortium behind the technologies that power the internet, WCAG describes how to make a website accessible. All of these rules are applicable to Typst's HTML output, and many of them apply to its PDF output. WCAG separates its rules into the three levels A, AA, and AAA. It is recommended that normal documents aim for AA. If you have high standards for Universal Access, you can also consider AAA Success Criteria. However, Typst does not yet expose all PDF features needed for AAA compliance, e.g. an AT-accessible way to define expansions for abbreviations.
- The **European Norm EN 301 549**[B]: Its Section 9 describes how to create accessible websites and its Section 10 describes what rules apply to non-web documents, including PDFs created by Typst. It points out which WCAG clauses are also applicable to PDFs. Conformance with this standard is a good start for complying with EU and national accessibility laws.

Keep in mind that in order to conform with EN 301 549 and the relevant WCAG provisions, your document must be tagged. If you aim for conformance, we strongly suggest using PDF/UA-1 for export to automate many of the checks for the success criteria within.

Many territories have accessibility legislation that requires you to create accessible files under some circumstances. Here are only some of them:

- **European Accessibility Act (EAA, EU 2019/882)**[C]: This regulation applies to e-books, consumer banking services, e-commerce services, and more. It requires the files distributed in these applications to be accessible.
- **Americans with Disabilities Act (ADA)**: The Department of Justice will require public sector organizations to provide files[D] in accordance to WCAG under Title II of the ADA by 2026. Likewise, private organizations can be held liable[E] for inaccessible digital services under the ADA and state law.

Using this guide can help you reach compliance with either regulation.

4.4.11 Testing for Accessibility

In order to test whether your PDF document is accessible, you can use automated tools and manual testing. Some standards like PDF/UA and PDF/A can be checked exclusively through automated tools, while some rules in WCAG and other standards require manual checks. Many of the automatable checks are automatically passed by Typst when Tagged PDF is enabled. For many other automatable checks, you can enable PDF/UA-1 export so that Typst will run them instead. Automated tools can only provide a baseline of accessibility. For truly Universal Access, it is best if you try the document yourself with AT.

Here is a list of automated checkers to try to test for conformance:

- **veraPDF**[F]**:** This open-source tool can check if your PDF file conforms to the parts of the PDF/A and PDF/UA standards it declared conformance with. Use this tool if you have chosen one of these standards during export. Failures are considered bugs in Typst and should be reported on GitHub[G].
- **PDF Accessibility Checker (PAC)**[H]**:** The freeware PAC checks whether your document complies with PDF/UA and WCAG rules. When you receive a hard error in the PDF/UA tab, this is considered a bug in Typst and should be reported on GitHub[I]. Warnings in the PDF/UA and Quality tabs may either be bugs, problems in your document, or neither. Check on the Forum[J] or on Discord[K] if you are unsure. Errors and warnings in the WCAG tab indicate problems with your document.

[A] https://www.w3.org/TR/WCAG21/
[B] https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf
[C] https://eur-lex.europa.eu/eli/dir/2019/882/oj
[D] https://www.ada.gov/law-and-regs/regulations/title-ii-2010-regulations/
[E] https://www.boia.org/blog/the-robles-v.-dominos-settlement-and-why-it-matters
[F] https://verapdf.org
[G] https://github.com/typst/typst/issues
[H] https://pac.pdf-accessibility.org/en
[I] https://github.com/typst/typst/issues
[J] https://forum.typst.app/
[K] https://discord.gg/2uDybryKPe
[A] https://helpx.adobe.com/acrobat/using/create-verify-pdf-accessibility.html

– **Accessibility Check in Adobe Acrobat Pro**[A]**:** The accessibility checker in the paid version of Adobe Acrobat checks all PDF documents for problems. Instead of checking compliance with a well-known international or industry standard, Adobe has created their own suite of tests. Because the rules behind these tests sometimes contradict international standards like PDF/UA, some of Acrobat's checks are expected to fail for Typst documents[B]. Other checks, such as the contrast check are useful and indicate problems with your document.

When doing manual checking, you can start with a checklist. If your organization places emphasis on accessibility, they will sometimes have their own list. In absence of one, you can try lists by universities such as Universität Bremen (in English)[C] or governments such as in Canada[D] or by the US Social Security Administration[E]. Although these checklists differ in verbosity, they all cover the most essential manual checks. Many of the technical checks in them can be skipped if you choose PDF/UA-1 export in Typst. If unsure which checklist to use, choose one from an organization culturally similar to yours.

However, to reach the highest standard of accessibility for widely circulated documents, consider checking your document with AT. Although there are many AT products and PDF viewers, it is typically sufficient to test a single combination. Which is best differs depending on your operating system:

– Windows: Test with Adobe Acrobat[F] and NVDA[G]. NVDA is free, open-source software. A free version of Acrobat is available.
– macOS: Test with Adobe Acrobat[H] and VoiceOver[I]. VoiceOver is the screen reader that is built into macOS and other Apple platforms.
– Linux: Test with Evince[J] or Okular[K] and Orca[L]. All three tools are free, open-source software. However, AT support across Linux platforms lags behind what is available on Windows and macOS. Likewise, Evince and Okular have less accessibility support than Acrobat. We strongly suggest testing with Acrobat instead.

When first getting into testing, consider completing the interactive training program your screen reader offers, if any. Building confidence with a screen reader helps you experience your document like a full-time screen reader user. When checking your document, check that it not only makes all the same information accessible that is available to a sighted user, but also that it is easy to navigate. The experience your users will have will vary based on the pairing of PDF viewer and AT they use.

4.4.12 Limits and considerations for export formats

Even when you design your document with accessibility in mind, you should be aware of the limitations of your export format. Fundamentally, AT support for PDF files is more difficult to implement than for other formats such as HTML. PDF was conceived in 1993 to accurately render print documents on a computer. Accessibility features were first added with PDF 1.4 in 2001, and improved in PDF 1.5 (2003) and PDF 2.0 (2017). By contrast, HTML offers a richer semantic model and more flexibility, so AT support in browsers generally surpasses what is possible in PDF viewers.

Also keep in mind that PDF files are mostly static. This allows you to disregard many WCAG and EN 301 549 rules designed for interactive content and multimedia. However, the lack of interactivity also makes it more difficult for users to customize a document's layout to their needs.

For example, WCAG Success Criterion 1.4.12[M] (codified in Clause 10.1.4.12 of EN 301 549) prescribes that a user must be able to increase character, letter, line, and paragraph spacing to very wide values. This benefits users with reduced vision or dyslexia. The Success Criterion does not require you to design your

[B] For example, when using footnotes, the check "Lbl and LBody must be children of LI" in the "List" section is expected to fail.
[C] https://www.uni-bremen.de/fileadmin/user_upload/universitaet/Digitale_Transformation/Projekt_BALLON/Checklisten/2._Auflage_englisch/Checklist_for_accessible_PDF_ENG-US_ver2.pdf
[D] https://a11y.canada.ca/en/pdf-accessibility-checklist/
[E] https://www.ssa.gov/accessibility/checklists/PDF_508_Compliance_Checklist.pdf
[F] https://www.adobe.com/acrobat.html
[G] https://www.nvaccess.org/download/
[H] https://www.adobe.com/acrobat.html
[I] https://support.apple.com/guide/voiceover/welcome/mac
[J] https://wiki.gnome.org/Apps/Evince/
[K] https://okular.kde.org/
[L] https://orca.gnome.org
[M] https://www.w3.org/WAI/WCAG21/Understanding/text-spacing.html

document with these layout parameters. Instead, it only requires a mechanism through which users can increase these parameters when reading the document. For HTML files, it is easy to comply with this Success Criterion because the browser lets the user override these spacing parameters on a page. For PDF, the situation is more nuanced: Theoretically, Typst adds tags and attributes designed for reflow to a file. A PDF reader, when reflowing, could allow its user to increase spacings beyond what is codified in these tags. In practice, we are not aware of a PDF viewer with this feature. Instead, this Success Criterion can be satisfied by repurposing the PDF into a HTML file and opening it in a browser.

In practice, even if your file is technically compliant, you cannot expect your users to know about these workarounds. Therefore, if you are aiming to meet the highest standards of Universal Access, consider distributing an HTML version of your document alongside your PDF. Export this file directly using Typst's HTML export[3.15] (in preview). Even though HTML export will not conserve many aspects of your visual layout, it will produce a file that leverages semantic HTML and technologies like Digital Publishing ARIA[A] to provide Universal Access. It will be of a higher quality than a PDF file repurposed to HTML.

Finally, keep in mind that PDFs are designed for print. Hence, you should not assume that interactive features like links are available to users who chose to print your document.

As mentioned above, files created by PNG and SVG export are not accessible.

[A] https://www.w3.org/TR/dpub-aria-1.1/

Changelog

Learn what has changed in the latest Typst releases and move your documents forward. This section documents all changes to Typst since its initial public release.

INFO

Some changelog items contain references like this one: #5017. These refer to a *pull request* or *issue* on GitHub[A] related to the change. If the reference refers to another repository than `typst/typst`, the repository is explicitly listed like this: typst/hayagriva#350.

Versions

0.15.1 625
0.15.0 626
0.14.2 638
0.14.1 639
0.14.0 641
0.13.1 649
0.13.0 650
0.12.0 656
0.11.1 662
0.11.0 663
0.10.0 668
0.9.0 670
0.8.0 673
0.7.0 675
0.6.0 678
0.5.0 680
0.4.0 681
0.3.0 683
0.2.0 685
0.1.0 686
Earlier 688

[A] https://github.com/typst/typst

Typst 0.15.1 (July 17, 2026)

Library

Text

- Updated New Computer Modern fonts to version 8.1.1, fixing a bug where the regular weight of the math font was still using the old calligraphic letterforms[3.8.26/1.6] that were supposed to live in stylistic set[3.7.12/2.24 6] #8552

Math

- Fixed a regression where alignment points did not work correctly when placed within `lr`[3.8.17/2.1] / matched delimiters #8566
- Fixed a regression where `op`[3.8.24] elements could be vertically misaligned #8546

Layout

- Fixed a bug where gaps could appear in multi-page lists with `number-align`[3.6.12/3.9] / `marker-align`[3.6.3/3.6] set to an alignment with a vertical component #8649

Export

SVG

- Fixed that inline SVGs resulting from `html.frame`[3.15.4] elements did not respect pretty printing #8535

Bundle

- In the watch server, the automatic selection of appropriate `Content-Type` headers based on file extensions was expanded to include more commonly used file types (e.g. `json` files) #8650
- The error for PNG/SVG documents with multiple pages is now only raised if it persists until the final document iteration #8618

Command Line Interface

- The `typst eval` subcommand will now exit with code 1 when the provided expression fails to evaluate #8623

Typst 0.15.0 (June 15, 2026)

This section documents all changes to the Typst language and compiler between Typst 0.14.2 and 0.15.0. If you are migrating an existing document to Typst 0.15, make sure to check out the Migration guide. It walks you through changes you may need to make to your existing documents to ensure compatibility with Typst 0.15.

Highlights

- Typst now supports variable fonts[3.7.12/2.33]
- HTML export now supports equations out of the box via MathML
- With the new, experimental *bundle*[3.18] export target, a single Typst project can output multiple files (e.g. a multi-page website)
- A single document can now contain multiple bibliographies[3.6.2/4.5]
- Typst can now target multiple PDF standards[3.14.2] at once
- The new `within`[3.5.25/3.5] selector simplifies many introspection use cases
- The new `divider`[3.6.5] element represents a thematic break that templates can style
- Spot colors[3.11.3/4.21] enable use of custom pigments in offset printing
- With the new file `path`[3.5.21] type, project-relative paths can be passed to packages
- The new, more general `typst eval` CLI subcommand supersedes `typst query`
- Layout convergence[3.4.4] issues now result in detailed diagnostics
- Two long-standing list layout issues with marker alignment and centering were fixed
- Paragraph handling in HTML export is improved, preventing unexpected paragraphs from appearing
- This documentation now has a print version[A]

Language

Syntax

- File paths[3.5.21] (e.g. in imports or `image`[3.11.7] function calls) may not contain backslashes anymore; instead forward slashes must be used #7688 **(Breaking change)**
- Added hints for invalid characters in code mode #7752
- Added hint when trying to use a unary operator directly in an embedded expression using a hash (e.g. `#-30deg`) #8069
- Fixed potential stack overflow crashes by enforcing a maximum parsing depth #7207
- Fixed incremental parsing of unclosed strings #8067

Styling

- Text show rules now have tracebacks that include the matched text #8088
- Fixed a crash with text show rules that match on multi-character symbols #8011

Scripting

- Extended hint when built-in definitions are shadowed to set and show rules #7131
- Added hint when trying to spread one or multiple dictionaries into an array #7798
- Improved diagnostics for invalid method calls #7865
- Improved hint for unknown variables in math that are available in `std` #7810
- Fixed a misleading error message when trying to assign to a temporary return value #8212

Library

Foundations

- Added file `path`[3.5.21] type that is now accepted in all places where paths were previously only represented as strings #7555
  - A path constructed in one file can be used in another file, but will be resolved relative to its original file
  - Likewise, paths can be passed across package boundaries

[A] https://github.com/typst/typst/releases/download/v0.15.0/typst-documentation.pdf

  - The initial path type is very minimal, but additional features like file existence checks or directory walking are planned
- Collections
  - Added `map` and `filter` functions on dictionaries[3.5.11/3.8] and arguments[3.5.1/4.5] #7284
  - Named arguments on `arguments`[3.5.1] values are now accessible with field access syntax #8407
  - The `slice` functions on strings[3.5.27/4.5] and arrays[3.5.2/3.9] will now error if passing both an `end` and a `count` #7238 **(Minor breaking change)**
  - Added `inclusive` parameter[3.5.2/3.13.3] to `range` function #8345
- Calculation
  - Added `asinh`[3.5.7/1.16], `acosh`[3.5.7/1.17], `atanh`[3.5.7/1.18], and `erf`[3.5.7/1.21] functions to `calc` module #6967 #7655
  - Added `int.min` and `int.max` constants for the minimum and maximum representable integer, respectively #8445
  - Fixed behavior of `quo`[3.5.7/1.40] for negative integers #7392
  - Fixed potential overflows in `norm`[3.5.7/1.41], `abs`[3.5.7/1.1], `gcd`[3.5.7/1.25], and `lcm`[3.5.7/1.26] #8156 #8352
  - Floating-point calculations are now consistently deterministic across platforms #7712
- Date & time handling
  - The `offset` parameter[3.5.9/5.1.1] of `datetime.today` now accepts durations[3.5.12] as an alternative to integers, allowing for sub-hour precision offsets #7691
  - Addition and subtraction of datetimes[3.5.9] and durations[3.5.12] now retains precision instead of clamping to full days in some cases #7843
  - The `datetime`[3.5.9] constructor now emits more precise errors when components are missing #7371
- Conversions
  - Added `base` parameter[3.5.16/2.2] to `int` constructor to configure in which base to parse a string #7386
  - Fixed that the `base` parameter[3.5.27/3.2] of the `str` constructor was accepted for non-integer values if its value was `10` #7237 **(Minor breaking change)**
  - Added hint when trying to construct a string[3.5.27/3] with base 1 #7237
- The `panic`[3.5.20] function now displays strings as-is instead of showing their `repr`[3.5.24], making it more suitable for friendly, user-facing messages #8211
- Changed `repr`[3.5.24] of styles and locations[3.12.4] to be more distinct #7364

Model

- Added `divider`[3.6.5] element representing a thematic break that templates can style #7982
- Bundle-related elements
  - The `document`[3.6.6] element can now be constructed to produce individual documents in bundle export #7964
  - Added `path`[3.6.6/3.1] and `format`[3.6.6/3.2] parameters to `document` element #7964
  - Added experimental `asset`[3.6.1] element #7964
- Bibliography management
  - A single document can now contain multiple bibliographies #8307
  - Added `target` parameter[3.6.2/4.5] to `bibliography` element to configure which citation is picked up by which bibliography #8307
  - Added `group` parameter[3.6.2/4.6] to `bibliography` element to configure how numbers are shared/reset across bibliographies #8307
  - Added support for numeric values for the `month` key in `.bib` files typst/biblatex#104
  - Added support for BibLaTeX name options in `.bib` files typst/biblatex#90
  - Added support for propagating non-numeric `volume` fields in `.bib` files to bibliographies typst/hayagriva#478
  - Improved sorting in bibliographies to take into account language conventions typst/hayagriva#314
  - Improved interoperability with CSL styles; for a full listing of changes, review the Hayagriva 0.10.0 changelog[A]
  - Added support for setting directors on videos without a parent in Hayagriva YAML files typst/hayagriva#450
  - Improved handling of `Anthos` entries in Hayagriva YAML files by treating them as `chapter`s in CSL typst/hayagriva#427
- Footnotes

[A] https://github.com/typst/hayagriva/blob/v0.10.0/CHANGELOG.md

  - The link of a `footnote`[3.6.9] is now within the superscript instead of around it, improving PDF tagging and HTML output #7637
  - The thickness of the default footnote separator[3.6.9/4.1.2] is now specified in font-relative instead of absolute units #7950
- Numbering
  - Added `numbering`[3.6.13] support for Armenian numerals #7529, Arabic Abjad numerals typst/codex#170, and Tibetan numerals typst/codex#163
  - Greek `numbering`[3.6.13] now uses the modern Greek style #7632
- The `par.first-line-indent`[3.6.15/4.6] property will now *fold,* meaning that partial dictionaries across different set rules or `par`[3.6.15] calls are combined #7747
- Added `list.marker-align`[3.6.3/3.6] property for defining how to align list markers #7895
  - When omitted, it will default to the new baseline alignment (vertically), combined with `end` alignment (horizontally)

Text

- Added support for variable fonts #8425
  - The well-known variation axes `ital`, `slnt`, `wght`, `wdth`, and `opsz` are automatically set based on text `weight`[3.7.12/2.4], `stretch`[3.7.12/2.5], `style`[3.7.12/2.3], and `size`[3.7.12/2.6]
  - Custom variations can be configured via the new `variations` parameter[3.7.12/2.33] of the `text` function
  - When using a variable font with Typst, the suffixes "Variable", "Var", and "VF" should be omitted as Typst trims them to unify static and variable fonts into a single family #8444 **(Minor breaking change)**
- Font features
  - The `text.alternates`[3.7.12/2.23] parameter now accepts an integer in addition to a boolean to select stylistic alternates other than `0` and `1` #7423
  - Parsing of tag names in `text.features`[3.7.12/2.32] is now more strict #7423 **(Minor breaking change)**
- Fixed that `context text.font` did not reflect the `covers` field #7354
- Fixed uneven CJK-Latin spacing[3.7.12/2.11] in justified paragraphs #7606
- Fixed a bug where the `lorem`[3.7.3] function would not produce the exact number of requested words #7877
- Improved translations for Swedish #7166, Portuguese #7088, Czech #7318, Latvian #7701, Slovak #7734, Polish #7734, Vietnamese #7774, Finnish #7988 #7989, and Welsh #7811
- Added font exception to avoid *SimSun-ExtB* being incorrectly merged with *SimSun* #8042
- Updated New Computer Modern fonts to version 8.1.0 #7597 #7663 #8164 #8330 #8435
  - This update changes the default look of calligraphic letterforms[3.8.26/1.6] in the math font; the previous style can be restored through `show math.equation: set text(stylistic-set: 6)`
- Updated Unicode components #8406
  - In particular, this fixed an issue with linebreaking of guillemets

Math

- Layout
  - Improved layout of under/over elements like `underbrace`[3.8.25/1.3] #7299
  - Slightly improved spacing around `math.op`[3.8.24] elements #7429
  - In which situations *cramped* styles (with tighter spacing) are applied is now fully consistent with TeX and MathML Core #8082
  - The `size` parameter[3.8.17/2.1.1] of the `lr` function now consistently applies to middle delimiters[3.8.17/2.2] in the same way it does to outer delimiters #7435
  - The `size` parameter[3.8.17/2.1.1] of the `lr` function now resolves relatively to the height of just the inner content; it does not take the delimiters into account anymore #7605 **(Minor breaking change)**
  - Glyph stretching[3.8.22] is now always relative to the base glyph rather than a potentially already scaled version (e.g. due to display sizing) #7435 **(Minor breaking change)**
  - Fixed left/right alignment not being applied correctly due to spacing next to alignment points #7473 #7435
  - The `binom`[3.8.11] element now uses different OpenType constants for layout; though this does not lead to visible changes with most fonts #7685
  - The default length and stroke width of `math.cancel`[3.8.12] lines is now specified in font-relative instead of absolute units #7241
  - Fixed potential misalignment in `cases`[3.8.13] function #7662
- Text handling

  - Improved handling of multi-character symbols in math #6929
  - Fixed that some glyphs did not stretch correctly in script sizes #8171
  - Fraction, root, and under/over lines now respect `text.stroke`[3.7.12/2.8] #7540
  - Accents in math are now always rendered in front of their base if they overlap #7733
- The `class`[3.8.14] function now applies the class only to its direct body rather than recursively #8328 **(Minor breaking change)**
- More delimiter symbols (e.g. `chevron.l`) are now callable to produce an `lr`[3.8.17/2.1] element #7228 **(Minor breaking change)**
- Fixed various bugs with rendering of mathematical expressions that look like function calls but in reality aren't (e.g. `$pi(1, 2)$`, since `pi` is not a function) #7865
- Fixed a bug with ordering of primes and nested attachments #7647

Symbols

- Added many new symbols and variants. View the `codex` 0.3.0 changelog[A] for a full listing.

Layout

- Baseline information is now retained in many more parts of the layout engine #8150 **(Breaking change)**
  - In particular, text contained in a `box`[3.10.5] with an inset is now aligned with the text surrounding the `box`
  - This also fixes a bug where wrapping an inline equation in a `box` would shift its baseline
  - Similarly, using a `block`[3.10.4] in an equation will keep the baseline intact
  - Last but not least, the marker/number and item of a `list`[3.6.3] or `enum`[3.6.12] are now properly baseline-aligned with the first line of the item even if the item is vertically larger than a normal line #7895
- Centering something in a list now centers based on the full available width rather than based on the maximum width of other list content #7895
- Page layout
  - Added `bleed` parameter[3.10.17/4.6] to `page` element to set up bleed margins #6357
  - Fixed the size of `"us-executive"` paper[3.10.17/4.1] #7869
  - Added warning for `show page` rules as they are unsupported #7445
- Paragraph layout
  - Fixed a bug where justified text could accidentally protrude into the margin when it ends with certain kinds of characters (e.g. a zero-width space) #8415
  - Fixed a bug where first-line indent[3.6.15/4.6] was applied at the start of a column even if `all: false` is set #7722
- Added support for spacing that is both weak[3.10.27/2.2] and fractional[3.10.9] #6833

Visualize

- Added support for spot colors[3.11.3/4.21] (also called separation colors) #7629
- Tilings
  - Added `offset` parameter[3.11.13/3.3] for shifting the starting position of a tiling #7506
  - Fixed parent-relative placement for `stack`[3.10.28] and `polygon`[3.11.9] #8324
- Gradients
  - Added `color.map.coolwarm`[3.11.3/3] for use with gradients #7489
  - Fixed interpolation of gradient stops[3.11.6/7.7] in Oklab color space #7326
  - Fixed gradient angle[3.11.6/7.10] handling for negative-size shapes #7826
  - Fixed gradient strokes for lines[3.11.8] and curves[3.11.4] #7863
  - Fixed parent-relative placement for `stack`[3.10.28] and `polygon`[3.11.9] #8324
- Fixed various bugs with rectangle strokes in combination with radii[3.11.10/2.5] #7357 #8081
- Fixed a potential deadlock with font fallback in SVGs #7766
- Various improvements to SVG image handling (see the resvg 0.46 and 0.47 changelogs[B]); in particular:
  - Added support for SVGs without top-level `xmlns` attribute
  - Added support for variable fonts in SVG using the `font-variation-settings` CSS property
- Various improvements to PDF image handling (see the commits between `d0b540f` and `3483462` on hayro[C]); in particular:

[A] https://github.com/typst/codex/blob/v0.3.0/CHANGELOG.md
[B] https://github.com/linebender/resvg/blob/v0.47.0/CHANGELOG.md
[C] https://github.com/LaurenzV/hayro/compare/d0b540fc9ab8e18b4a7a000d1404139af8e9d023...34834627c0b4afa9c83c9b64d4d978b127030c77

  - Added support for JPEG2000 (`JPXDecode`) and JBIG2 (`JBIG2Decode`) images
  - Improved parsing robustness for non-compliant files
  - Added support for blend modes

Introspection

- Layout convergence issues now result in detailed diagnostics that help pin down the cause #7364
- Added `within` selector[3.5.25/3.5] that matches elements that are contained within any elements matching an ancestor selector #8250 #7964 #8114
- Added `at`[3.12.1/11.2.2] parameter to `counter.display` function #6781
- Improved how `counter.display`[3.12.1/11.2] auto-selects the numbering to use #7446

Data Loading

- Added support for namespaces to `xml`[3.13.7] function #7899
- Added hint when trying to read from a path that looks like a URL #7682
- Diagnostics for binary file loading failures now include file paths #8259
- The `json`[3.13.4] function now emits a friendly error when the loaded JSON has a leading UTF-8 BOM #7488

Export

Bundle

- Added new, experimental *bundle* export target[3.18] #7964
  - With bundle export, you can emit multiple output files from a single Typst project
  - Bundles can contain any combination of HTML pages[3.15], PDFs[3.14], PNGs[3.16], SVGs[3.17], and arbitrary assets[3.6.1]

HTML

- Mathematical equations are now automatically exported to MathML *(thanks to @mkorje[A])* #7436
  - MathML defines how to render an equation, but also preserves its semantics
  - If you've previously relied on show rules that use `html.frame`[3.15.4] to render equations to SVG, try MathML output. Switching to it will improve the accessibility of your document (though rendering will be somewhat less consistent across browsers).
- The `box`[3.10.5] and `block`[3.10.4] elements' purpose is now aligned with paged export #8181 **(Breaking change)**
  - `box`[3.10.5] is used to bring block-level content inline
  - `block`[3.10.4] ensures inline-level content becomes block-level
  - Depending on the contained content, this may be achieved by setting the CSS `display` property or by wrapping in an additional `<span>` or `<div>`.
- The rules of how paragraphs are grouped in HTML have been adjusted to fix cases where paragraphs would appear unexpectedly #8181 #7505 **(Breaking change)**
  - The list of HTML elements that can be grouped into paragraphs was tweaked (it now includes all *phrasing content* with the exception of elements that default to `display: none`)
  - This default can be controlled by wrapping an element in a `box`[3.10.5] or `block`[3.10.4] as appropriate
  - HTML elements that cannot be part of paragraphs (like a `<div>`) do not immediately force adjacent inline-level Typst content to be wrapped in a paragraph; rather, they are considered *neutral* for paragraph grouping
  - Paragraph creation is only forced by *block-level Typst elements* (as opposed to HTML elements). Built-in block-level elements like headings or images wrap the HTML elements they create in `block`[3.10.4] elements to force adjacent inline content into paragraphs. Package authors should do the same to ensure paragraph creation is consistent between HTML and paged export.
- The `target`[3.5.30] function can now be used without the `html` feature flag (the rest of HTML export remains feature-flagged) #8248
- DOM structure and built-in show rules
  - The root `<html>` element now receives a `lang` attribute respecting what was configured for `text.lang`[3.7.12/2.16] #7208

[A] https://github.com/mkorje

  - The Typst `image`[3.11.7] element now always emits `width` and `height` attributes on the generated HTML `<img>` element #8118
  - Code in `<pre>` tags will now prefer raw newlines over `<br>` elements to encode line breaks #7675
  - Fixed generated HTML for quote attributions[3.6.17/2.3] #8181
  - Fixed table cell show rules not working in HTML export #7821
- Serialization
  - HTML is now minified by default; use the `--pretty` CLI flag or the checkbox in the web app to pretty-print it #8371
- HTML elements
  - The `attrs` parameter[3.15.3/1.2] on `html.elem` will now *fold,* meaning that partial attributes across different set rules or `html.elem` calls are combined #8182
  - `html.script`[3.15.5/2.83] and `html.style`[3.15.5/2.92] only accept a string and not arbitrary content anymore #7784 **(Breaking change)**
- Whitespace handling
  - HTML `<br>` elements now collapse adjacent Typst spaces #8166
  - Fixed spans being emitted to protect whitespace from collapsing unnecessarily in some cases #8166

PDF

- Typst can now target multiple (compatible) PDF standards at once, e.g. PDF/UA-1 and PDF/A-2a #8294
- PDFs are now a bit more space-optimized at the cost of being harder to inspect with a text editor; use the `--pretty` CLI flag or the checkbox in the web app to pretty-print them #8294 #8371 #8430
- Labelled headings now result in named destinations even if they are not referenced #7964
- Graphics
  - Added support for more compositing features of COLRv1 fonts LaurenzV/krilla#358
  - Fixed rendering of gradients[3.11.6] in LinearRGB, CMYK, and Luma color spaces #8149
  - Fixed excessive sampling of linear gradients #7818
- Tagging
  - Added support for more specific artifact kinds in `pdf.artifact`[3.14.4]; these are now internally used when appropriate #8416
  - Fixed "invalid document structure" errors with complex list markers[3.6.3/3.2] #7789
  - Fixed wrong PDF tagging order for inline content outside of paragraphs #7861
  - Fixed bounding box computations for stroked shapes in tagged PDFs #8322
- Standards compliance
  - Fixed potentially incompliant PDF files by emitting an error when PDF/UA-1 is requested but complying with PDF/UA-1 would require newer features than available in the current PDF version LaurenzV/krilla#278
  - Fixed potentially incompliant PDF 1.4 files by emitting errors when implementation limits are exceeded LaurenzV/krilla#348

SVG

- SVGs are now minified by default; use the `--pretty` CLI flag or the checkbox in the web app to pretty-print them #8371
- Somewhat reduced the size of generated SVGs #7476 #7680 #7857
- SVG elements do not have the classes `typst-frame`, `typst-doc`, `typst-group`, `typst-shape`, and `typst-text` anymore #7680 **(Minor breaking change)**
- Graphics
  - Fixed a bug where tilings[3.11.13] could be incorrectly reused in multiple places #7837
  - Fixed rendering of gradients[3.11.6] in LinearRGB, CMYK, and Luma color spaces #8149
  - Fixed handling of conic gradient angles[3.11.6/7.3.2] #7952
  - Fixed excessive sampling of linear gradients #7818
- Fixed positioning and sizing of color bitmap glyphs #7679 #8440
- Fixed sources of non-determinism in SVG export #7680

PNG

- Fixed handling of conic gradient angles[3.11.6/7.3.2] #7952
- Fixed positioning and sizing of color bitmap glyphs #8440
- Fixed that negatively scaled[3.10.24] text with equal x and y scale would turn invisible in PNG export #8111

Command Line Interface

- Added new `typst eval` subcommand to evaluate a Typst code expression from the CLI; this command supersedes `typst query` #7362
- Tracebacks for diagnostics are now more compact and readable #8000
- Added `--pretty` flag for producing human-readable output; output is otherwise minified by default (applies to HTML, SVG, and PDF, but not PNG) #8371
- Fonts
  - The output of `typst fonts --variants` is now more readable and informative; in particular, it also displays the paths of font files and, for variable fonts, variation axes #7490 #8425
  - The CLI now discovers fonts lazily, saving time on operations that do not need fonts (like HTML export without frames[3.15.4]) #7380
  - Adobe Creative Cloud fonts are now discovered as system fonts #7716
- Dependency output
  - When writing to stdout while using `--deps-format make` (which is incompatible and thus fails), no empty Make dependency file will be emitted anymore #7246
  - The JSON dependency format now includes information about outputs in addition to inputs #7209
- Path handling fixes on Windows #8508
  - Reading from a path that contains a drive letter now results in an explicit error instead of the drive letter being silently ignored
  - Attempting to read from a path that contains a reserved filename (like `COM1`) now results in an explicit error
- Non-Unicode input paths are not supported anymore #7688 **(Breaking change)**
- The experimental `--timings` argument now requires an explicit file name instead of defaulting to `record-{n}.json` #8119 **(Breaking change)**
- Added colors to `--help/-h` and `typst info` output #7443 #7500
- Fixed inconsistency in environment variable handling between `typst info` and `typst compile` #8030
- Fixed a bug with the injection of a live reload script when using `typst watch` with HTML export #7770
- Fixed a bug where local timezone information was taken into account even if a fixed date is set via `--creation-timestamp` or `SOURCE_DATE_EPOCH`, leading to irreproducible results #7856

Tooling

- Syntax highlighting
  - In math, parentheses used for grouping are now highlighted differently than ones intended for display #7894
- Autocomplete and tooltips
  - Autocompletion and tooltips are now aware of parameters of user-defined functions #7808
  - Autocomplete descriptions and tooltips for font families are now more detailed and, for variable fonts, include variation axes #8425
  - Function autocompletions in math mode now always prefer round parentheses over square brackets (which are not supported in math) #8417
  - Fixed autocompletion and tooltips not working with argument lists in math mode #8116

Deprecations

- Certain unclear/ambiguous ways to write a raw language tag; these will now emit a warning in anticipation of an upcoming change to how they are parsed[3.7.6/3.2] #8257
- Fallback to Arabic numerals for `numbering`[3.6.13] systems that do not support the number zero (e.g. for `"⓪"`) #7936
- Various symbols were renamed, see the deprecation section in the dedicated changelog[A] for a full listing
- Some citation styles that were renamed or superseded typst/hayagriva#404 typst/hayagriva#424 typst/hayagriva#453
  - `council-of-science-editors` is now called `cse-citation-sequence-brackets-8th-edition`
  - `council-of-science-editors-author-date` is now called `cse-name-year`
  - `modern-language-association-8` / `mla-8` is superseded by `modern-language-association` / `mla`
  - `vancouver` is now called `nlm-citation-sequence`

[A] https://github.com/typst/codex/blob/v0.3.0/CHANGELOG.md#deprecations-in-sym

  - `vancouver-superscript` is now called `nlm-citation-sequence-superscript`
- Undocumented array forms of `enum`[3.6.12] and `terms`[3.6.21] items #7484

Removals

- The `path` element, use `curve`[3.11.4] instead #7554 **(Breaking change)**
- The `pattern` type, use `tiling`[3.11.13] instead #8252 **(Breaking change)**
- The `pdf.embed` element, use `pdf.attach`[3.14.5] instead #8252 **(Breaking change)**
- The scoped functions `cbor.decode`, `csv.decode`, `json.decode`, `toml.decode`, `xml.decode`, `yaml.decode`, and `image.decode`; directly pass `bytes`[3.5.6] to the top-level functions instead #8252 **(Breaking change)**
- Various previously deprecated symbols, see the removals section in the dedicated changelog[A] for a full listing **(Breaking change)**

Development

- The `typst-kit` crate was completely reworked to make it easier to create a Typst `World` implementation #7710 #8026
- Diagnostic hints can now have spans (though typically they will be *detached,* which just means there isn't a span) #7364
- Increased minimum supported Rust version to 1.92 #8236
- Moved Nix flake from typst/typst[B] to typst/typst-flake[C], where it is now maintained by the community as a best effort #7512

Migration guide

This section walks you through changes you may need to make to your existing documents to ensure compatibility with Typst 0.15.

Typst 0.15 removes definitions from the Typst standard library that were already deprecated in previous Typst versions. If you are still relying on those, also refer to the Typst 0.14[D] and Typst 0.13[E] migration guides.

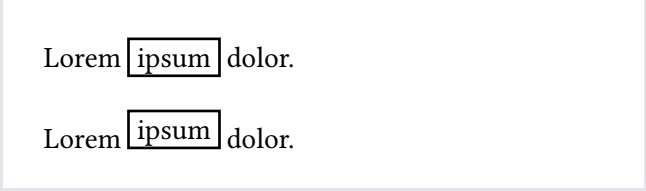
Baselines

Typst now retains baseline information in more parts of the layout engine (most importantly for boxes[3.10.5], blocks[3.10.4], list items[3.6.3/4.1], and equations[3.8.15]). This may lead to silent layout shifts. These are generally improvements, but if you've previously counter-acted bad-looking output through explicit adjustments, you may need to revisit those.

In particular, Typst will now set the baseline of a `box`[3.10.5] to the baseline of its first line of content and use it to align the `box` with its surrounding text. This yields better-looking output out of the box.

```
// Typst 0.15+
Lorem #box(inset: 0.3em, stroke: 1pt)[ipsum] dolor.

// Typst 0.14 and below
Lorem #box(inset: 0.3em, stroke: 1pt)[ipsum] dolor.
```



[A] https://github.com/typst/codex/blob/v0.3.0/CHANGELOG.md#removals-in-sym-breaking-change
[B] https://github.com/typst/typst
[C] https://github.com/typst/typst-flake
[D] https://typst.app/blog/2025/typst-0.14#migrating
[E] https://typst.app/blog/2025/typst-0.13#migrating

File paths

If you are using Typst on Windows, you may have used backslashes in your file paths. This syntax is not supported anymore. Consistent use of forward slashes ensures your document compiles on all supported platforms.

```
// Typst 0.15+ ✅
#include "chapters/introduction.typ"

// Typst 0.14 and below ❌
#include "chapters\\introduction.typ"
```

Moreover, the Typst CLI does not support non-Unicode input paths anymore. Such paths are exceedingly rare today.

Math glyph stretching

In mathematical typesetting, glyphs frequently need to be stretched to the size of some part of an equation. This primarily applies to brackets, but also accents, arrows, and more. In Typst, this happens implicitly through matched delimiters or explicitly through the `lr`[3.8.17/2.1] and `stretch`[3.8.22] functions.

Typst 0.15 makes some adjustments to how ratios passed to these functions are resolved. Previously, a size of `200%` passed to `lr.size`[3.8.17/2.1.1] or `stretch.size`[3.8.22/1.2] would be interpreted relative to a glyph size that was potentially already affected by display sizing[3.8.21/1.1]. Now, ratios are always resolved relative to the base glyph size. In practice, this means that, to retain the same appearance, you may need to increase the target sizes you've configured for glyphs that are already larger out of the box in display style (e.g. integrals).

Math delimiters as functions

In Typst's math mode, various symbols defined in General Symbols[3.9.2] can be called like a function, either to be applied as an accent or to delimit body content. Typst 0.15 extends this handling to additional delimiters like `chevron.l`. This can change the output of existing calls, which would previously fall back to displaying the parentheses (which would rarely have produced desirable output in the case of delimiters).

```
// Typst 0.15+ ✅
$ chevron.l(x) $

// Typst 0.14 and below ⚠️
$ chevron.l(x) $
```

$$\langle x \rangle$$

$$\langle (x) $$

Math classes

The `class`[3.8.14] function defines how part of an equation should be laid out (primarily the spacing around it). In Typst 0.14 and below, the `class` function applied recursively: If applied to a larger piece of content, all items in it would receive the class. In Typst 0.15, the class is only applied to the directly wrapped content. This is mostly a bug fix, but can lead to subtle layout changes.

Calligraphic letterforms in the default math font

With the update to *New Computer Modern Math* 8.1.0, the default calligraphic letterforms[3.8.26/1.6] were changed. The previous letterforms can still be accessed by selecting stylistic set[3.7.12/2.24] 6.

```
// Typst 0.15, default style.
$ R != cal(R) $

// Typst 0.15, with stylistic set 6.
// Reproduces the default style from Typst 0.14 and below.
#show math.equation: set text(stylistic-set: 6)
$ R != cal(R) $
```

$$R \neq \mathcal{R}$$

$$R \neq \mathcal{R}$$

Paragraphs, boxes, and blocks in HTML

Typst automatically collects inline-level content into paragraphs[3.6.15/2]. In Typst 0.14 and below, the exact same rules were used to collect inline-level HTML elements (e.g. a `<span>`) into Typst paragraphs, which in turn result in `<p>` elements. This had the unfortunate effect that even use of the low-level typed HTML API[3.15.5] could result in `<p>` elements appearing automatically. Consider the example below:

```
#html.div({
  html.span[Hello]
  html.div[World]
})
```

Where Typst 0.14 would insert an unexpected additional `<p>` element, Typst 0.15 yields the expected result:

```
<!-- Typst 0.14 and below ⚠️ -->
<div>
  <p><span>Hello</span></p>
  <div>World</div>
</div>

<!-- Typst 0.15+ ✅ -->
<div>
  <span>Hello</span>
  <div>World</div>
</div>
```

This change in output is achieved through a few new rules for how paragraphs are collected in HTML export. Previously, Typst would always categorize elements into either inline-level or block-level. It would then force inline-level elements into a paragraph whenever at least one block-level element was present in the same flow of content. In Typst 0.15, elements can instead be considered inline-level, block-level, or *neutral:*

- Text, Typst `box`[3.10.5] elements and HTML *phrasing content* (elements that are allowed as children of `<p>` elements) are considered inline-level[A]
- All other HTML elements are considered neutral
- Typst `block`[3.10.4] elements are considered block-level

Neutral elements don't force adjacent inline-level content into paragraphs. They can co-exist with inline-level content in a *mixed flow.* This ensures that usage of the typed HTML API does not result in extraneous paragraphs.

Meanwhile, Typst elements that are block-level out of the box (e.g. `heading`[3.6.10]) now explicitly use a `block`[3.10.4] in their default show rule to ensure that they still force adjacent inline-level elements into paragraphs. This ensures consistency in paragraph grouping across paged and HTML export. Package authors should also explicitly use blocks[3.10.4] when creating HTML components that Typst should consider block-level.

To make this change possible, the way boxes[3.10.5] and blocks[3.10.4] behave in HTML export has been adjusted: Instead of unconditionally wrapping their contents in a `<span>` or `<div>`, they now do just what is necessary to ensure their contents are considered inline- or block-level by browsers. If they contain multiple children, they still create a wrapper element, but for just a single child, they instead configure the CSS `display` property. And if the single child already has the appropriate `display` property by default, it is fully omitted. Then, the effect of the `box` or `block` is limited to influencing Typst's paragraph grouping.

[A] There are a few exceptions: Some elements like `<script>` are considered phrasing content by the HTML specifications, but don't make sense in paragraphs as they default to `display: none`.

HTML `script` and `style` elements

The typed HTML functions for the HTML `script`[3.15.5/2.83] and `style`[3.15.5/2.92] element previously accepted arbitrary body content and then only failed during HTML serialization if the content resulted in non-textual elements. In Typst 0.15, they instead only accept strings in the first place. Note that you can also use them with `raw`[3.7.6] syntax by accessing the `.text` field.

```
// Typst 0.15+ ✅
#html.style("a { color: red }")
#html.style(
  ```css
  a { color: red }
  ```.text
)

// Typst 0.14 ❌
#html.style[
  a { color: red }
]
```

Classes in SVG export

Typst 0.14 and below would generate the `class` attributes `typst-frame`, `typst-doc`, `typst-group`, `typst-shape`, and `typst-text` on SVG elements in SVG export. These are no longer emitted. If you are using the `typst-frame` class to style `html.frame`[3.15.4] elements in HTML export, you'll need to adapt your style sheet.

Variable font family names

Typst aims to unify different fonts from the same family under a single family name. To that effect, it automatically trims common style suffixes like "Bold" or "Condensed" from font family names. Instead of selecting these through the name, they should be accessed through Typst's built-in mechanisms (such as the `weight`[3.7.12/2.4] and `stretch`[3.7.12/2.5] parameters).

In Typst 0.15, the additional suffixes "Variable", "Var", and "VF" are trimmed to unify static and variable fonts into a single family. If you've previously used a font with any of these suffixes in its family name, you should now omit the suffix when specifying the `font`[3.7.12/2.1].

Upcoming raw language tag changes

In the next Typst version, we will make a change to how raw language tags are parsed[3.7.6/3.2]. Should this upcoming change have an effect on your document, the compiler will already warn you today and suggest how to adjust your document to future-proof it against the change.

Numbering fallback for the number zero

Not all `numbering`[3.6.13] systems can express the number zero. In Typst 0.14 and below, those would silently fall back to Arabic numerals for `0`. This behavior is deprecated in Typst 0.15. Using zero with these systems will become a hard error in the future.

Tightened validation in some functions

The `array.slice`[3.5.2/3.9] function, `str`[3.5.27] constructor, and `text.features`[3.7.12/2.32] parameter now perform stricter validation on their inputs. If you've previously passed invalid or nonsensical input, you will now receive an error.

Renamed citation styles

Typst 0.15 renames some citation styles to stay aligned with upstream CSL changes. If you are relying on any of these, the compiler will warn you and suggest the new name.

Renamed symbols

Typst 0.15 also renames[A] a few symbols in General Symbols[3.9.2]. If you are using any of these, the compiler will warn you and suggest the new name.

[A] https://github.com/typst/codex/blob/v0.3.0/CHANGELOG.md#deprecations-in-sym

Typst 0.14.2 (December 12, 2025)

Security

– Updated the WebAssembly runtime used for executing plugins[3.5.22]. The version used in Typst 0.14.0 and 0.14.1 suffers from a memory handling vulnerability. Based on our investigation, the vulnerability would be very hard to exploit in practice, but an exploit could theoretically be feasible. In any case, **we recommend upgrading to Typst 0.14.2.** This holds in particular for local users. In the web app, the bug is not critical as the browser offers an extra layer of protection.

  Typst 0.13.1 and below are not affected by this vulnerability.

  Technical details: The `wasmi`[A] WebAssembly runtime versions used in 0.14.0 and 0.14.1 have a *use-after-free* memory handling bug in certain memory growth situations. Specifically, the bug occurs when the plugin tries to grow its memory, but allocating the requested amount of memory fails. Based on our investigation, the bug is hard to trigger in practice as the WebAssembly linear memory is always limited to 4GB on a technical level and modern operating systems rarely fail to serve a 4GB memory allocation request (typically not even under RAM pressure). Once the bug is triggered, it would also still be very challenging to turn it into an actual exploit. Regardless, we recommend upgrading to Typst 0.14.2 for protection against a potential exploit.

Diagnostics

– Added a hint when `array.sorted`[3.5.2/3.30] fails because a pair of elements could not be compared. This hint aids with fixing bugs in user code that were surfaced by a change in internal sorting behavior in Typst 0.14.1.

[A] https://github.com/wasmi-labs/wasmi/

Typst 0.14.1 (December 03, 2025)

PDF export

- Fixed regression where links to labelled elements would sometimes not work correctly LaurenzV/krilla#304
- Fixed bug where PDF text attributes could be written incorrectly #7478
- Fixed crash in link handling #7471
- Fixed crash for zero-sized pages #7454
- Fixed crash when a table `vline`[3.6.20/4.3] or `hline`[3.6.20/4.2] has an out-of-bounds index #7448
- Fixed crash in formatting of font-related PDF export errors #7460
- Fixed crash when a `footnote`[3.6.9] or `place`[3.10.19] element was queried and reinserted into the document #7216
- Fixed crash for PNGs with invalid metadata LaurenzV/krilla#286, LaurenzV/krilla#287
- Fixed bug where text in SVGs with `fill-and-stroke` paint order could be exported incorrectly LaurenzV/krilla#292
- Fixed bug with layer isolation in SVGs where blending/masking is used LaurenzV/krilla#295
- Fixed that table headers could be tagged incorrectly in some scenarios LaurenzV/krilla#289
- Fixed issues where generated PDFs could differ between 32-bit and 64-bit systems LaurenzV/krilla#317, LaurenzV/krilla#316, LaurenzV/krilla#312
- Upgraded JPEG decoder used during PDF export for improved compatibility, fixing a case where a valid JPEG was rejected LaurenzV/krilla#288
- A PDF document information dictionary that would be empty is now fully omitted instead LaurenzV/krilla#280
- A rare crash in PDF tagging was turned into a compiler error #7450

HTML export

- Fixed export of table headers[3.6.20/4.4] and footers[3.6.20/4.5] with gutter #7332
- A `page`[3.10.17] set rule in HTML export is now a warning instead of a hard error, in line with how unsupported elements are generally treated #7513

Math

- Fixed regression where `arrow.l.r` could not be used as an accent[3.8.9] anymore #7481
- Fixed that single-letter strings did not react to spaces around them like multi-letter strings do #7276
- Fixed that spacing around `mat`[3.8.18] and `vec`[3.8.27] with a fence delimiter was whitespace-dependent #7309
- Fixed height calculation for horizontally stretched glyphs #7327

Model

- Fixed regression where Typst would error in heading numbering functions that don't handle the counter state `(0,)`. This can occur in the first layout iteration. Such errors are usually automatically caught by Typst, which was not the case here. #7459

Text

- Fixed regression where Typst would synthesize superscripts for some fonts even when typographic[3.7.11/2.1] glyphs were available #7462
- Fixed regression where some oblique fonts would be classified as italic #7483

Scripting

- Fixed crash due to violated invariants in `array.sorted`[3.5.2/3.30] #7520
- Fixed crashes due to overflow in `calc.rem`[3.5.7/1.37], `calc.rem-euclid`[3.5.7/1.39], `calc.div-euclid`[3.5.7/1.38], `calc.quo`[3.5.7/1.40], and `calc.gcd`[3.5.7/1.25] #7419
- Upgraded WebAssembly runtime, fixing a bug that plugins[3.5.22] could run into #7438

Command Line Interface

- Compiling to standard output in combination with `--deps --deps-format=make` (which results in an error) will not produce an empty Make dependency file as a side effect anymore #7246

Development

- Increased minimum supported Rust version to 1.89 #7363

Typst 0.14.0 (October 24, 2025)

Highlights

- Typst now produces *accessible* PDFs[4.4] out of the box, with opt-in support for stricter checks and conformance to PDF/UA-1
- Typst now supports all PDF/A standards[3.14.2/3]
- PDFs can now be used as images[3.11.7/2.2] (thanks to @LaurenzV[A])
- Added support for character-level justification[3.6.15/4.4] (can significantly improve the appearance of justified text)
- Added support for many more built-in elements in HTML export
- Added typed HTML API (e.g. `html.div`[3.15.5/2.29]) with individually typed attributes
- Added support for multiple headers[3.6.20/4.4] and subheaders in tables
- Added `title`[3.6.22] element for displaying the document title
- Added `frac.style`[3.8.16/3.3] property for producing skewed and inline fractions

PDF export

PDF export was fully rewritten to use the new `krilla`[B] library, fixing various bugs and enabling many improvements. Known fixes are listed below, but there will likely be other changes in how the output behaves. If you spot any regressions, please report them on GitHub[C]. *(Thanks to @LaurenzV[D] for creating krilla!)* #5420

- Typst now produces *accessible* PDFs out of the box. Such documents are suitable for consumption in a wide range of circumstances. That not only includes consumption by people with permanent or temporary disabilities, but also by those with different devices or preferences.
  - Typst PDFs are now *tagged* by default. *Tags* are rich metadata that PDF viewers can use to make the document consumable in other ways than visually (e.g., through a screen reader).
  - In addition, Typst can now emit documents conforming to the PDF/UA-1 standard. (PDF/UA-2 is not yet supported, but planned.)
  - There is an increasing amount of existing and upcoming legislation requiring documents to be accessible, for instance, the European Accessibility Act and the Americans with Disabilities Act.
  - For more details on all of this, read the new Accessibility Guide[4.4].
- Typst now supports all PDF/A standards: PDF/A-1b, PDF/A-1a, PDF/A-2b, PDF/A-2u, PDF/A-2a, PDF/A-3b, PDF/A-3u, PDF/A-3a, PDF/A-4, PDF/A-4f, and PDF/A-4e. See the expanded PDF/A documentation[3.14.2/3] for guidance on how to select a suitable standard. #5420 #7038
- Typst now supports the PDF versions 1.4, 1.5, 1.6, and 2.0 in addition to PDF 1.7. See the relevant section of the PDF documentation[3.14.2/1] for details. #5420
- Added `pdf.artifact`[3.14.4] function for marking content as not semantically meaningful #6619
- Added experimental `pdf.header-cell`[3.14.7], `pdf.data-cell`[3.14.6], and `pdf.table-summary`[3.14.8] functions for enhancing accessibility of documents with complex tables. These functions are guarded by the `a11y-extras` feature. They do not have a final interface and will be removed in the future, either through integration into table functions or through full removal. #6619
- PDF heading bookmarks now contain the heading's numbering #6622
- Attachments[3.14.5]
  - Renamed `pdf.embed` to `pdf.attach`[3.14.5] (the old name will remain as a deprecated alias until Typst 0.15) #6705
  - The `mime`[3.14.5/3.4] property of `pdf.attach` is now checked for syntactical correctness (**Minor breaking change**)
  - Fixed parsing of the `data`[3.14.5/3.2] argument of `pdf.attach` #6435
  - Attachments now smartly determine whether they should be compressed #6256
- Text extraction (i.e. copy paste)
  - Now works correctly even when multiple different characters result in the same glyph #5420

[A] https://github.com/LaurenzV
[B] https://github.com/LaurenzV/krilla
[C] https://github.com/typst/typst/issues
[D] https://github.com/LaurenzV

  - Spaces between words at which a natural line break occurred are now correctly retained for text extraction #6866
  - Fixed mapping of hyphenation artifacts to Unicode text #6799
- Images
  - CMYK images now work properly in PDF export #5420
  - Improved export of text in SVG images with a filter #5420
  - Improved compatibility of SVG images with Quartz rendering engine (the engine used in Apple Preview) #5420
  - Improved handling of SVG images with high group nesting depth #5420
- Fixed a bug with text in patterns #5420
- Fixed gradients with transparency #5420

HTML export

- Added support for many more built-in elements (the *Model* category[3.6] is now fully covered)
  - The `image`[3.11.7] element #6578
  - The `footnote`[3.6.9] and `footnote.entry`[3.6.9/4.1] element #6917
  - The `outline`[3.6.14] and `outline.entry`[3.6.14/5.1] element #6606
  - The `bibliography`[3.6.2] element #6952
  - The `smartquote`[3.7.8] element #6710
  - The `sub`[3.7.10] and `super`[3.7.11] elements #6422
  - The `underline`[3.7.13], `overline`[3.7.5], `strike`[3.7.9], and `highlight`[3.7.1] elements #6510
  - The `smallcaps`[3.7.7] element #6600
  - The `lower`[3.7.4] and `upper`[3.7.14] functions #6585
- Added typed HTML API (e.g. `html.div`[3.15.5/2.29]) with individually typed attributes #6476
  - For example, to generate a `video` element you can now write `#html.video(width: 400, src: "sunrise.mp4")` instead of `#html.elem("video", attrs: (width: "400", src: "sunrise.mp4"))`. Note how the `width` attribute takes an integer instead of a string.
- Added support for intra-doc `link`[3.6.11] targets #6602
- The `raw`[3.7.6] element
  - Added syntax highlighting support #6691
  - Block-level `raw` elements now emit both a `<code>` and a `<pre>` tag #6701
  - The language tag[3.7.6/4.3] of `raw` elements is now preserved as a `data-lang` attribute on the `<code>` tag #6702
- The `authors`[3.6.6/3.4] and `keywords`[3.6.6/3.6] properties of the `document` function now yield corresponding HTML `<meta>` tags #6134
- The `html.elem`[3.15.3] function now supports custom HTML element names #6676
- Improved encoding of `html.frame`[3.15.4] #6605
- Empty attributes are now encoded with shorthand syntax (e.g. `<div hidden></div>`) #6479
- Zero-sized horizontal weak spacing (`h(0pt, weak: true)`) does not cause a "was ignored during HTML export" warning anymore, so it can be used to destruct surrounding spaces without producing any output, as in paged export #6917
- Fixed encoding of `<pre>` and `<textarea>` elements that start with a newline #6487 #6497
- Fixed encoding of raw text elements[A] #6487 #6720
- Fixed sizing of `html.frame`[3.15.4] #6505
- Fixed `measure`[3.10.14] in HTML export #7186
- Fixed nested `html.frames`[3.15.4] #6509
- Fixed that a `box`[3.10.5] without a body was ignored in HTML export #6709
- Fixed encoding of whitespace in HTML #6750

SVG export

- Added support for COLR-flavored color glyphs #6693
- Reduced amount of `<g>` grouping elements that are generated #6247

[A] https://html.spec.whatwg.org/#raw-text-elements

PNG export

- Fixed crash when text size[3.7.12/2.6] is negative #7004

Visualize

- Added support for using PDFs as images[3.11.7] using the new `hayro`[A] library. PDFs will be embedded directly in PDF export, rasterized in PNG export, and turned into SVGs in SVG and HTML export. *(Thanks to @LaurenzV[B] for creating hayro!)* #6623
- Added support for WebP images #6311
- Various minor improvements for SVG images (see the resvg 0.44 and 0.45 changelogs[C])
- SVG images can now refer to external image files #6794
- Clip paths are now properly anti-aliased #6570
- Fixed gradients on curves where the last segment is `curve.line`[3.11.4/3.2] #6647
- Fixed stroke cap handling of shapes with partial strokes #5688
- Fixed corner radius handling of shapes with partial strokes #6976
- Fixed crash when sampling across two coinciding gradient stops #6166

Layout

- Added opt-in support for character-level justification in addition to word-level justification, configured via the new `par.justification-limits`[3.6.15/4.4] property. This is an impactful microtypographical technique that can significantly improve the appearance of justified text. #6161
- Fixed wrong linebreak opportunities related to object replacement characters #6251
- Fixed an issue where a breakable block would still produce an empty segment even if nothing fit into the first segment, leading to various undesirable behaviors in combination with fills, strokes, and stickiness[3.10.4/2.13] #6335
- Fixed crash with set rule for column or rowspan on a grid cell #6401
- Fixed CJK-Latin-spacing[3.7.12/2.11] at manual line breaks #6700 and at sub- and superscript boundaries #7175

Math

- Added `frac.style`[3.8.16/3.3] property with new options for skewed and inline fractions #6672
- Added `math.equation.alt`[3.8.15/3.5] property for setting an alternative description for an equation #6619
- Text handling
  - A single equation can now use multiple fonts #6365
  - Glyph layout in math now uses proper text shaping, leading to better handling of more complex Unicode features #6336
  - Generated characters in an equation (e.g. the ∫ produced by `sqrt`) can now be targeted by text show rules #6365
  - Added `scr`[3.8.26/1.7] function for roundhand script font style #6309
  - Added `dotless` parameter to `accent`[3.8.9] (typically for rendering a dotless accented i or j) #5939
  - Script-style glyphs are now preferred at reduced math sizes #6320
  - Fixed `text.stroke`[3.7.12/2.8] in math #6243
  - Broken glyph assemblies are now prevented even when font data is incorrect #6688
- Layout
  - Fixed a bug with vertical accent positioning #5941
  - Fixed positioning of bottom accents #6187
  - Fixed a bug with layout of roots #6021
  - Improved layout of `vec`[3.8.27] and `cases`[3.8.13], making it consistent with `mat`[3.8.18] #5934
  - Removed linebreak opportunity before closing bracket in inline math #6216
- An `augment`[3.8.18/2.3] line can now exist at the start and end of a matrix, not only in between columns and rows #5806
- Shorthands and multi-character numbers do not bind more tightly than fractions anymore in cases like `$x>=(y)/z$` #5925 #5996 **(Minor breaking change)**

[A] https://github.com/LaurenzV/hayro
[B] https://github.com/LaurenzV
[C] https://github.com/linebender/resvg/blob/v0.45.1/CHANGELOG.md#0450—2025-02-26

- Named arguments passed to symbols used as function now raise an error instead of being silently ignored #6192 **(Minor breaking change)**
- The `mid`[3.8.17/2.2] element does not force the `"large"` math class[3.8.14] upon its contents anymore and instead defaults to `"relation"` #5980
- Fixed error in math parsing when `..` isn't followed by anything #7105
- Fixed the default math class of ϑ, ∫, ∱, ∳, ∫, ∀, and ∢ #5949 #6537

Thanks to @mkorje[A] for his work on math!

Model

- Tables
  - Added support for multiple headers[3.6.20/4.4] and subheaders in tables #6168
  - Table headers now force a rowbreak, i.e. an incomplete row before a header will not be filled with cells after the header #6687
  - Fixed a bug where headers[3.6.20/4.4] and footers could accidentally expand to contain non-header cells #5919
- Added `title`[3.6.22] element for displaying the document title #5618
- Added `figure.alt`[3.6.8/5.2] property for setting an alternative description for a figure #6619
- Link[3.6.11] hit boxes for text are now vertically a bit larger to avoid issues with automatic link detection in PDF viewers #6252
- The `link`[3.6.11] function will now produce an error when passed an empty string as a URL #7049 **(Minor breaking change)**
- The value of the `number`[3.6.12/4.1.1] argument of `enum.item` now takes `auto` instead of `none` for automatic numbering #6609 **(Minor breaking change)**
- Improved spacing of nested tight lists #6242
- Fixed that `quotes: false` was ignored for inline-level quotes[3.6.17/2.1] #5991
- Fixed hanging indent[3.6.10/5.8] for centered, numbered headings #6839
- Fixed `footnote.entry`[3.6.9/4.1] show rules breaking links from footnote to entry #6912
- Hebrew numbering does not add Geresh and Gershayim anymore #6122

Bibliography

- Built-in styles
  - Updated styles to their latest upstream CSL versions typst/hayagriva#350
  - Renamed `"chicago-fullnotes"` to `"chicago-notes"` (the old name remains as a deprecated alias) #6920
  - Renamed `"modern-humanities-research-association"` to `"modern-humanities-research-association-notes"` (the old name remains as a deprecated alias) #6994
  - Added support for locator/supplement in alphanumeric style typst/hayagriva#307
- Hayagriva format
  - Added `chapter` field[B] corresponding to CSL `chapter-number` and BibLaTeX `chapter` typst/hayagriva#383
- BibLaTeX format
  - Fixed parsing of alphanumeric page ranges typst/biblatex#86
  - Added support for `%` comment syntax typst/biblatex#80
  - Fixed parsing of space-separated single character commands typst/biblatex#71
  - Added "primaryclass" alias for "eprintclass" field typst/biblatex#75
  - Added support for BibLaTeX `language` field typst/hayagriva#317
  - Improved translation of BibLaTeX fields to `genre` and `serial-number` typst/hayagriva#296 typst/hayagriva#369
- CSL handling
  - The bibliography rendering now uses `strong`[3.6.19], `emph`[3.6.7], and `smallcaps`[3.7.7] to express CSL font styling instead of directly adjusting the `text`[3.7.12] style, making styling easier #6984
  - Added support for date seasons, which are displayed when the month is missing typst/hayagriva#391
  - Terms for "AD" and "BC" are now correctly used from the chosen locale typst/hayagriva#364

[A] https://github.com/mkorje
[B] https://github.com/typst/hayagriva/blob/v0.9.1/docs/file-format.md#chapter

  - Fixed how subsequent citations with differing supplements translate into CSL `ibid` and `ibid-with-locator` positions #6171
  - Fixed handling of `ibid` and `ibid-with-locator` positions in styles typst/hayagriva#301
  - Fixed the `location` conditional in CSL styles for citations with no locator typst/hayagriva#399
  - Fixed accesses of the year suffix resulting in wrong CSL renders typst/hayagriva#400
  - Fixed regression where page variables were no longer supported in styles' `<number>` elements typst/hayagriva#289
  - Fixed sorting and formatting of name parts typst/hayagriva#287 typst/hayagriva#313
  - Fixed year suffix collapsing typst/hayagriva#367
  - Fixed delimiters in locale-specific date formatting typst/hayagriva#385
  - Fixed rendering of date ordinals typst/hayagriva#366
  - Fixed rendering and sorting of dates with BC years typst/hayagriva#334 typst/hayagriva#368
  - Fixed sorting for empty sort values typst/hayagriva#390

Text

- The `sub`[3.7.10] and `super`[3.7.11] functions now use the `subs` and `sups` OpenType font features instead of special Unicode characters for typographic scripts, fixing semantical, sizing, and positioning issues #5777
- The `raw`[3.7.6] element
  - Tweaked default syntax-highlighting color scheme of `raw`[3.7.6] text to make the colors more accessible #6754
  - JSON keys and string values now use different colors in the default `raw` syntax highlighting theme[3.7.6/4.6] #6873
  - Fixed a crash when a `raw` syntax[3.7.6/4.5] contains an unescaped trailing backslash #6883
  - Fixed tab indentation in `raw`[3.7.6] text with CRLF line terminators #6961
- Translations
  - Added term translations (a term being the "Section" in "Section 1") for many new languages by importing pre-existing translations from LaTeX packages #6852
  - Added term translations for Indonesian #6108, Latvian #6348, Croatian #6413, Lithuanian #6587, French #7010, French (Canada) #7098, Galician #7019, Dutch #7026, Danish #7031, Slovenian #7032, Italian #7050, Chinese #7044, Spanish #7051, and Irish (Gaeilge) #7024
  - Improved term translations for Czech #6101, Swedish #6519, Galician #7019, Dutch #7026, Chinese #7044, Spanish #7051, Irish (Gaeilge) #7024, and German #7082
  - Improved smart quotes for French #5976, Ukrainian #6372, Russian #6331, and Arabic #6626
- An empty font list is not allowed anymore in `text.font`[3.7.12/2.1] #6049 **(Minor breaking change)**
- Added a warning when using a variable font as those are not currently supported #6425
- Fixed usage of the same font with different coverage[3.7.12/2.1] settings #6604
- Fixed hyphens not showing up when hyphenating at specific positions (where invisible metadata exists) #6807
- Fixed styling of repeated hyphens in languages with hyphen repetition #6798
- Last resort font fallback does not consider default ignorable characters anymore during font selection #6805
- Updated New Computer Modern fonts to version 7.0.4 #6376
- Updated data and shaper to Unicode 16.0.0 #5407

Scripting

- The `in` operator can now be used to check whether a definition is present in a module #6498
- Added `default` parameter to `array.first`[3.5.2/3.2], `array.last`[3.5.2/3.3] #5970, `array.join`[3.5.2/3.26] #6932, `str.first`[3.5.27/4.2], and `str.last`[3.5.27/4.3] #6554 methods
- Added `by` parameter to `array.sorted`[3.5.2/3.30] for sorting with a comparison function #5627
- Added `str.normalize`[3.5.27/4.10] function for Unicode normalization #5631
- Added `from`[3.10.8/1.1], `to`[3.10.8/1.2], and `sign`[3.10.8/1.4] methods to `direction` type #5893
- Labels[3.5.17] cannot be empty anymore #6332 **(Minor breaking change)**
- The WebAssembly runtime used by the `plugin`[3.5.22] system was updated and now supports SIMD #6997
- Numberings and counters now use 64-bit numbers instead of platform-dependent numbers for consistency across platforms #6026
- Improved consistency of how large numbers are handled in data loading functions #6836

- The `toml`[3.13.6] function is now guaranteed to return a dictionary[3.5.11] and `toml.encode`[3.13.6/4.1] must receive a dictionary (it already errored before when passed something else, but the new function signature makes the error clearer) #6743
- Serialization of `bytes`[3.5.6] in human-readable formats now uses `repr`[3.5.24] #6743
- Fixed slicing of last N elements of an array using the `count`[3.5.2/3.9.3] parameter #6838
- Fixed crash when the expression wrapped in a `context` expression is an anonymous function #6975
- Fixed equality check between `raw.line`[3.7.6/5.1] elements #6560
- Fixed `repr`[3.5.24] of labels[3.5.17] being potentially syntactically invalid #6456
- Fixed `repr`[3.5.24] of functions modified through `with`[3.5.15/8.1] #6773

Introspection

- The following elements are newly locatable (i.e. they can be discovered with `query`[3.12.6], `locate`[3.12.3], etc. without having a label): `par`, `table`, `enum`, `list`, `terms`, `title`, `figure.caption`, `footnote.entry`, `outline.entry`, `image`, `emph`, `strong`, `link`, `cite`, `raw`, `underline`, `overline`, `strike`, and `highlight` #6619
- Fixed issues with logical order in bidirectional text #5887 #6796
- Fixed logical order of cells in right-to-left `grid`[3.10.10] #6232
- Fixed logical order of elements in `grid`[3.10.10] cells that span multiple pages #7198
- Fixed logical order of metadata at the starts and ends of paragraphs #6881 #6909
- Fixed introspection positions[3.12.4/2.2] of inline elements at the very start of a paragraph (previously the Y position of an element at the very start would differ from one in the middle of the first line) #6881

Styling

- Fixed rare infinite loop in show rule application #6259

Performance

- Optimized incremental compilation with a new algorithm that, in particular, eliminates cases of very slow compilation with heavy context usage #6683

Command Line Interface

- Added `typst info` subcommand for emitting build and environment information #6761
- Added `typst completions` subcommand for retrieving shell completions #6568
- Added `TYPST_IGNORE_SYSTEM_FONTS` environment variable #6092
- Added `--ignore-embedded-fonts` flag and `TYPST_IGNORE_EMBEDDED_FONTS` environment variable for disabling the use of fonts embedded into the Typst binary #7037
- Added `--no-pdf-tags` flag for disabling the automatic generation of accessibility tags. These are emitted when no particular standard like PDF/UA-1 is selected to provide a baseline of accessibility. #6619
- Added `--target` parameter to `typst query` #6405
- Added `--deps` and `--deps-format` parameters for emitting a list of files compilation depends on. Supports the three formats `json`, `zero`, and `make`. #7022
- Deprecated the `--make-deps` flag in favor of `--deps` with `--deps-format make` #7022
- On Linux, the font search will now fall back to known font directories if none were loaded via Fontconfig. RazrFalcon/fontdb#71
- The CLI will now warn when trying to watch the standard input #6381
- Fixed race condition when two separate Typst CLI processes concurrently try to install a package #5979
- Fixed incremental SVG export not writing output SVGs on changes that only affect the page (e.g. changing `page.fill`) #6810
- Fixed a rare potential crash when stack space couldn't be grown as expected #6969

Tooling and Diagnostics

- Errors in many kinds of external text files (e.g., bibliographies, JSON files, etc.) are now annotated within these files instead of at the positions where the files are loaded from a Typst file #6308
- Warnings originating from within `eval`[3.5.13] are now correctly emitted #6100
- Diagnostic messages and hints

  - Improved error messages related to parsing of numbers #5969
  - The error message for an unsuitable CSL style now mentions the name of the style #6306
  - Added hints to various deprecated items with the removal timeline #6617
  - Added hint for the error message when an X/Y pair is expected #6476
  - Added hint for a label that appears in both the document and the bibliography #6457
  - Added additional hint for show rule recursion depth error #5856
  - Fixed inconsistent formatting of code points and strings in error messages #6487
- Autocompletion
  - Labels will now be deduplicated in completions #6516
  - Math font autocompletions are now based on the presence of an OpenType MATH table instead of the word "Math" in the name #6316
  - Autocompletion immediately after a comma in a parameter list is now supported for explicitly triggered completions (e.g. via Ctrl/Cmd+Space) #6550
  - Citation style aliases are now displayed as autocompletions #6696
  - Fixed autocompletion false positives with cursor after parameter list #6475
  - Fixed autocompletion after partial identifier in math #6415
  - Fixed which definitions are suggested in math #6415
  - Fixed inapplicable method autocompletions being listed #5824
- Tooltips
  - Fixed tooltip for scoped functions (e.g. `calc.round`) #6234
  - Fixed tooltip and details for figure references #6580
  - Expression tooltips now use × instead of `x` to indicate a repeated value #6163
- Fixed jump from click (jumping to the source panel with a click in the preview) in presence of transformations and clipping #6037

Symbols

- Added many new symbols and variants; many more than could be listed here. View the dedicated changelog[A] for a full listing.
- Code points that have a symbol and emoji presentation now have the correct variation selector attached depending on whether they appear in `sym` or `emoji`. That said, they still don't render consistently in Typst due to how font fallback works. typst/codex#114
- The `symbol`[3.5.28] function can now be used to create symbols that comprise not just one character, but one full grapheme cluster #6489

Deprecations

- The name `pdf.embed` in favor of the new name `pdf.attach`[3.14.5] #6705
- The `"chicago-fullnotes"` bibliography style in favor of `"chicago-notes"` #6920
- The `"modern-humanities-research-association"` bibliography style in favor of `"modern-humanities-research-association-notes"` #6994
- The `--make-deps` CLI flag in favor of `--deps` with `--deps-format make` #7022
- Various symbols, see the deprecation section in the dedicated changelog[B] for a full listing

Removals

- Removed compatibility behavior of type/str comparisons (e.g. `int == "integer"`), which was temporarily introduced in Typst 0.8 and deprecated in Typst 0.13 **(Breaking change)**

Development

- The `Default` impl for `Library` had to be removed for crate splitting and trait coherence reasons, but you can get a drop-in replacement via `use typst::LibraryExt` #6576
- The `PdfOptions` struct has a new `tagged` field, which defaults to `true` #6619 #7046
- Fixed a potential panic in `World::font` implementations. Downstream `World` implementations might need to apply the same fix[C]. #6117

[A] https://github.com/typst/codex/blob/v0.2.0/CHANGELOG.md#version-020-october-7-2025
[B] https://github.com/typst/codex/blob/v0.2.0/CHANGELOG.md#deprecated

- Increased minimum supported Rust version to 1.88 #6637
- The Docker container now has the optional non-root user `typst` #7058

[C]https://github.com/typst/typst/pull/6117

Typst 0.13.1 (March 07, 2025)

Command Line Interface

- Fixed high CPU usage for `typst watch` on Linux. Depending on the project size, CPU usage would spike for varying amounts of time. This bug appeared with 0.13.0 due to a behavioral change in the inotify file watching backend.

HTML export

- Fixed export of tables with gutters[3.6.20/3.3]
- Fixed usage of `<html>` and `<body>` element within context[3.4]
- Fixed querying of metadata[3.12.5] next to `<html>` and `<body>` element

Visualization

- Fixed curves[3.11.4] with multiple non-closed components

Introspection

- Fixed a regression where labelled symbols[3.5.28] could not be queried[3.12.6] by label

Deprecations

- Fixed false positives in deprecation warnings for type/str comparisons

Typst 0.13.0 (February 19, 2025)

Highlights

- There is now a distinction between proper paragraphs[3.6.15] and just inline-level content. This is important for future work on accessibility and means that first line indent[3.6.15/4.6] can now be enabled for all paragraphs instead of just consecutive ones.
- The `outline`[3.6.14] has a better out-of-the-box look and is more customizable
- The new `curve`[3.11.4] function (that supersedes the `path` function) provides a simpler and more flexible interface for creating Bézier curves
- The `image` function now supports raw pixel raster formats[3.11.7/2.2] for generating images from within Typst
- Functions that accept file paths now also accept raw bytes[3.5.6], for full flexibility
- WebAssembly plugins[3.5.22] are more flexible and automatically run multi-threaded
- Fixed a long-standing bug where single-letter strings in math (`$"a"$`) would be displayed in italics
- You can now specify which charset should be covered[3.7.12/2.1] by which font family
- The `pdf.embed`[3.14.5] function lets you embed arbitrary files in the exported PDF
- HTML export is currently under active development. The feature is still *very* incomplete, but already available for experimentation behind a feature flag.

Model

- There is now a distinction between proper paragraphs[3.6.15] and just inline-level content **(Breaking change)**
  - All text at the root of a document is wrapped in paragraphs. Meanwhile, text in a container (like a block) is only wrapped in a paragraph if the container holds any block-level content. If all of the content is inline-level, no paragraph is created.
  - In the laid-out document, it's not immediately visible whether text became part of a paragraph. However, it is still important for accessibility, HTML export, and for properties like `first-line-indent`.
  - Show rules on `par` now only affect proper paragraphs
  - The `first-line-indent` and `hanging-indent` properties also only affect proper paragraphs
  - Creating a `par[..]` with body content that is not fully inline-level will result in a warning
  - The default show rules of various built-in elements like lists, quotes, etc. were adjusted to ensure they produce/don't produce paragraphs as appropriate
  - Removed support for booleans and content in `outline.indent`[3.6.14/4.4]
- The `outline`[3.6.14] function was fully reworked to improve its out-of-the-box behavior **(Breaking change)**
  - Outline entries[3.6.14/5.1] are now blocks[3.10.4] and are thus affected by block spacing
  - The `auto` indentation mode now aligns numberings and titles outline-wide for a grid-like look
  - Automatic indentation now also indents entries without a numbering
  - Titles wrapping over multiple lines now have hanging indent
  - The page number won't appear alone on its own line anymore
  - The link now spans the full entry instead of just the title and page number
  - The default spacing between outline leader dots was increased
  - The `fill`[3.6.14/5.1.3] parameter was moved from `outline` to `outline.entry` and can thus be configured through show-set rules
  - Removed `body` and `page` fields from outline entry
  - Added `indented`, `prefix`, `inner`, `body`, and `page` methods on outline entries to simplify writing of show rules
- Added configuration to `par.first-line-indent`[3.6.15/4.6] for indenting all paragraphs instead of just consecutive ones
- Added `form`[3.6.18/4.3] parameter to `ref` function. Setting the form to `"page"` will produce a page reference instead of a textual one.
- Added `document.description`[3.6.6/3.5] field, which results in corresponding PDF and HTML metadata
- Added `enum.reversed`[3.6.12/3.5] parameter
- Added support for Greek numbering[3.6.13]
- When the `link`[3.6.11] function wraps around a container like a block[3.10.4], it will now generate only one link for the whole block instead of individual links for all the visible leaf elements. This significantly reduces PDF file sizes when combining `link` and `repeat`[3.10.22].
- The `link`[3.6.11] function will now only strip one prefix (like `mailto:` or `tel:`) instead of multiple

- The link function now suppresses hyphenation via a built-in show-set rule rather than through its default show rule
- Displaying the page counter without a specified numbering will now take the page numbering into account

Visualization

- Added new `curve`[3.11.4] function that supersedes the `path` function and provides a simpler and more flexible interface. The `path` function is now deprecated.
- The `image` function now supports raw pixel raster formats[3.11.7/2.2]. This can be used to generate images from within Typst without the need for encoding in an image exchange format.
- Added `image.scaling`[3.11.7/2.8] parameter for configuring how an image is scaled by PNG export and PDF viewers (smooth or pixelated)
- Added `image.icc`[3.11.7/2.9] parameter for providing or overriding the ICC profile of an image
- Renamed `pattern` to `tiling`[3.11.13]. The name `pattern` remains as a deprecated alias.
- Added `gradient.center`[3.11.6/7.11], `gradient.radius`[3.11.6/7.12], `gradient.focal-center`[3.11.6/7.13], and `gradient.focal-radius`[3.11.6/7.14] methods
- Fixed interaction of clipping and outset on `box`[3.10.5] and `block`[3.10.4]
- Fixed panic with `path` of infinite length
- Fixed non-solid (e.g. tiling) text fills in clipped blocks
- Fixed a crash for images with a DPI value of zero
- Fixed floating-point error in `gradient.repeat`[3.11.6/7.5]
- Auto-detection of image formats from a raw buffer now has support for SVGs

Scripting

- Functions that accept file paths now also accept raw bytes[3.5.6]
  - `image`[3.11.7], `cbor`[3.13.2], `csv`[3.13.3], `json`[3.13.4], `toml`[3.13.6], `xml`[3.13.7], and `yaml`[3.13.8] now support a path string or bytes and their `.decode` variants are deprecated
  - `plugin`[3.5.22], `bibliography`[3.6.2], `bibliography.style`[3.6.2/4.4], `cite.style`[3.6.4/3.4], `raw.theme`[3.7.6/4.6], and `raw.syntaxes`[3.7.6/4.5] now accept bytes in addition to path strings. These did not have `.decode` variants, so this adds new flexibility.
  - The `path` argument/field of `image`[3.11.7] and `bibliography`[3.6.2] was renamed to `source` and `sources`, respectively **(Minor breaking change)**
- Improved WebAssembly plugins[3.5.22]
  - The `plugin` type is replaced by a `plugin` function[3.5.22] that returns a module[3.5.18] containing normal Typst functions. This module can be used with import syntax. **(Breaking change)**
  - Plugins now automatically run in multiple threads without any changes by plugin authors
  - A new `plugin.transition`[3.5.22/8.1] API is introduced which allows plugins to run impure initialization in a way that doesn't break Typst's purity guarantees
- The variable name bound by a bare import (no renaming, no import list) is now determined statically and dynamic imports without `as` renaming (e.g. `import "ot" + "her.typ"`) are a hard error **(Breaking change)**
- Values of the `arguments`[3.5.1] type can now be added with `+` and joined[3.3.2] in curly-braced code blocks
- Functions in an element function's scope can now be called with method syntax, bringing elements and types closer (in anticipation of a future full unification of the two). Currently, this is only useful for `outline.entry`[3.6.14/5.1] as no other element function defines methods.
- Added `calc.norm`[3.5.7/1.41] function
- Added support for 32-bit floats in `float.from-bytes`[3.5.14/3.4] and `float.to-bytes`[3.5.14/3.5]
- The `decimal`[3.5.10] constructor now also accepts decimal values
- Improved `repr` of symbols[3.5.28], arguments[3.5.1], and types[3.5.31]
- Duplicate symbol[3.5.28] variants and modifiers are now a hard error **(Breaking change)**

Math

- Fixed a bug where single letter strings in math (`$"a"$`) would be displayed in italics
- Math function calls can now have hyphenated named arguments and support argument spreading[3.5.1/2]
- Better looking accents thanks to support for the `flac` (Flattened Accent Forms) and `dtls` (Dotless Forms) OpenType features
- Added `lcm` text operator[3.8.24]

- The `bold`[3.8.23/1.3] function now works with 𝖿 and 𝖥
- The `italic`[3.8.23/1.2] function now works with ℏ
- Fixed a bug where the extent of a math equation was wrongly affected by internal metadata
- Fixed interaction of `lr`[3.8.17/2.1] and context[3.4] expressions
- Fixed weak spacing being unconditionally ignored in `lr`[3.8.17/2.1]
- Fixed sub/superscripts sometimes being in the wrong position with `lr`[3.8.17/2.1]
- Fixed multi-line annotations (e.g. overbrace) changing the math baseline
- Fixed merging of attachments when the base is a nested equation
- Fixed resolving of contextual (em-based) text sizes within math
- Fixed spacing around up tacks (⊥)

Bibliography

- Prose and author-only citations now use editor names if the author names are unavailable
- Some non-standard but widely used BibLaTeX `editortypes` like `producer`, `writer`, `scriptwriter`, and `none` (defined by widespread style `biblatex-chicago` to mean performers within `music` and `video` entries) are now recognized
- CSL styles can now render affixes around the bibliography
- For BibTeX entries with `eprinttype = {pubmed}`, the PubMed ID will now be correctly processed
- Whitespace handling for strings delimiting initialized names has been improved
- Uppercase spelling after apostrophes used as quotation marks is now possible
- Fixed bugs around the handling of CSL delimiting characters
- Fixed a problem with parsing multibyte characters in page ranges that could prevent Hayagriva from parsing some BibTeX page ranges
- Updated CSL APA style
- Updated CSL locales for Finnish, Swiss German, Austrian German, German, and Arabic

Text

- Added support for specifying which charset should be covered[3.7.12/2.1] by which font family
- Added `all`[3.7.7/4.1] parameter to `smallcaps` function that also enables small capitals on uppercase letters
- Added basic i18n for Basque and Bulgarian
- Justification[3.6.15/4.3] does not affect raw[3.7.6] blocks anymore
- CJK-Latin-spacing[3.7.12/2.11] does not affect raw[3.7.6] text anymore
- Fixed wrong language codes being used for Greek and Ukrainian
- Fixed default quotes for Croatian and Bulgarian
- Fixed crash in RTL text handling
- Added support for `raw`[3.7.6] syntax highlighting for a few new languages: CFML, NSIS, and WGSL
- New font metadata exception for New Computer Modern Sans Math
- Updated bundled New Computer Modern fonts to version 7.0.1

Layout

- Fixed various bugs with footnotes
  - Fixed footnotes getting lost when multiple footnotes were nested within another footnote
  - Fixed endless loops with empty and overlarge footnotes
  - Fixed crash with overlarge footnotes within a floating placement
- Fixed sizing of quadratic shapes (`square`[3.11.11] and `circle`[3.11.2])
- Fixed `block.sticky`[3.10.4/2.13] not working properly at the top of a container
- Fixed crash due to consecutive weak spacing
- Fixed crash when a block[3.10.4] or text have negative sizes
- Fixed unnecessary hyphenations occurring in rare scenarios due to a bad interaction between padding and paragraph optimization
- Fixed lone citations[3.6.4] in `align`[3.10.1] not becoming their own paragraph

Syntax

- Top-level closing square brackets that do not have a matching opening square bracket are now a hard error **(Minor breaking change)**
- Adding a space between the identifier and the parentheses in a set rule is not allowed anymore **(Minor breaking change)**
- Numbers with a unit cannot have a base prefix anymore, e.g. `0b100000pt` is not allowed anymore. Previously, it was syntactically allowed but always resolved to a value of zero. **(Minor breaking change)**
- Using `is` as an identifier will now warn as it might become a keyword in the future
- Fixed minor whitespace handling bugs
  - in math mode argument lists
  - at the end of headings
  - between a term list's term and description
- Fixed parsing of empty single line raw blocks with 3+ backticks and a language tag
- Fixed minor bug with parentheses parsing in math
- Markup that can only appear at the start of the line (headings, lists) can now also appear at the start of a list item
- A shebang `#!` at the very start of a file is now ignored

PDF export

- Added `pdf.embed`[3.14.5] function for embedding arbitrary files in the exported PDF
- Added support for PDF/A-3b export
- The PDF timestamp will now contain the timezone by default

HTML export

Note: HTML export is currently under active development. The feature is still *very* incomplete, but already available for experimentation behind a feature flag.

- Added HTML output support for some (but not all) of the built-in elements
- Added `html.elem`[3.15.3] function for outputting an arbitrary HTML element
- Added `html.frame`[3.15.4] function for integrating content that requires layout into HTML (by embedding an SVG)
- Added `target`[3.5.30] function which returns either `"paged"` or `"html"` depending on the export target

Tooling and Diagnostics

- Autocompletion improvements
  - Added autocompletion for file paths
  - Smarter autocompletion of variables: Completing `rect(fill: |)` will now only show variables which contain a valid fill (either directly or nested, e.g. a dictionary containing a valid fill)
  - Different functions will now autocomplete with different brackets (round vs square) depending on which kind is more useful
  - Positional parameters which are already provided aren't autocompleted again anymore
  - Fixed variable autocompletion not considering parameters
  - Added autocompletion snippets for common figure usages
  - Fixed autocompletion after half-completed import item
  - Fixed autocompletion for `cite` function
- Added warning when an unconditional return in a code block discards joined content
- Fixed error message when accessing non-existent label
- Fixed handling of nested imports in IDE functionality

Command Line Interface

- Added `--features` argument and `TYPST_FEATURES` environment variable for opting into experimental features. The only feature so far is `html`.
- Added a live reloading HTTP server to `typst watch` when targeting HTML
- Fixed self-update not being aware about certain target architectures

- Fixed crash when piping `typst fonts` output to another command
- Fixed handling of relative paths in `--make-deps` output
- Fixed handling of multipage SVG and PNG export in `--make-deps` output
- Colons in filenames are now correctly escaped in `--make-deps` output

Symbols

- New
  - `inter`, `inter.and`, `inter.big`, `inter.dot`, `inter.double`, `inter.sq`, `inter.sq.big`, `inter.sq.double`, `integral.inter`
  - `asymp`, `asymp.not`
  - `mapsto`, `mapsto.long`
  - `divides.not.rev`, `divides.struck`
  - `interleave`, `interleave.big`, `interleave.struck`
  - `eq.triple.not`, `eq.dots`, `eq.dots.down`, `eq.dots.up`
  - `smt`, `smt.eq`, `lat`, `lat.eq`
  - `colon.tri`, `colon.tri.op`
  - `dagger.triple`, `dagger.l`, `dagger.r`, `dagger.inv`
  - `hourglass.stroked`, `hourglass.filled`
  - `die.six`, `die.five`, `die.four`, `die.three`, `die.two`, `die.one`
  - `errorbar.square.stroked`, `errorbar.square.filled`, `errorbar.diamond.stroked`, `errorbar.diamond.filled`, `errorbar.circle.stroked`, `errorbar.circle.filled`
  - `numero`
- Renamed **(Breaking change)**
  - `ohm.inv` to `Omega.inv`
- Changed codepoint **(Breaking change)**
  - `angle.l.double` from 《 to ⟪
  - `angle.r.double` from 》 to ⟫
  - `angstrom` from U+212B (Å) to U+00C5 (Å)
- Deprecated
  - `sect` and all its variants in favor of `inter`
  - `integral.sect` in favor of `integral.inter`
- Removed **(Breaking change)**
  - `degree.c` in favor of `°C` (`$upright(°C)$` or `$upright(degree C)$` in math)
  - `degree.f` in favor of `°F` (`$upright(°F)$` or `$upright(degree F)$` in math)
  - `kelvin` in favor of just K (`$upright(K)$` in math)
  - `ohm` in favor of `Omega`

Deprecations

- The `path` function in favor of the `curve`[3.11.4] function
- The name `pattern` for tiling patterns in favor of the new name `tiling`[3.11.13]
- `image.decode`, `cbor.decode`, `csv.decode`, `json.decode`, `toml.decode`, `xml.decode`, `yaml.decode` in favor of the top-level functions directly accepting both paths and bytes
- The `sect` and its variants in favor of `inter`, and `integral.sect` in favor of `integral.inter`
- The compatibility behavior of type/str comparisons (e.g. `int == "integer"`) which was temporarily introduced in Typst 0.8 now emits warnings. It will be removed in Typst 0.14.

Removals

- Removed `style` function and `styles` argument of `measure`[3.10.14], use a `context`[3.4] expression instead **(Breaking change)**
- Removed `state.display` function, use `state.get`[3.12.7/6.1] instead **(Breaking change)**
- Removed `location` argument of `state.at`[3.12.7/6.2], `counter.at`[3.12.1/11.3], and `query`[3.12.6] **(Breaking change)**
- Removed compatibility behavior where `counter.display`[3.12.1/11.2] worked without `context`[3.4] **(Breaking change)**
- Removed compatibility behavior of `locate`[3.12.3] **(Breaking change)**

Development

- The `typst::compile` function is now generic and can return either a `PagedDocument` or an `HtmlDocument`
- `typst-timing` now supports WebAssembly targets via `web-sys` when the `wasm` feature is enabled
- Increased minimum supported Rust version to 1.80
- Fixed linux/arm64 Docker image

Typst 0.12.0 (October 18, 2024)

Highlights

- Added support for multi-column floating placement[3.10.19/4.2] and figures[3.6.8/5.4]
- Added support for automatic line numbering[3.6.15/5.1] (often used in academic papers)
- Typst's layout engine is now multithreaded. Typical speedups are 2-3x for larger documents. The multi-threading operates on page break boundaries, so explicit page breaks are necessary for it to kick in.
- Paragraph justification was optimized with a new two-pass algorithm. Speedups are larger for shorter paragraphs and go up to 6x.
- Highly reduced PDF file sizes due to better font subsetting (thanks to @LaurenzV[A])
- Emoji are now exported properly in PDF
- Added initial support for PDF/A. For now, only the PDF/A-2b profile is supported, but more is planned for the future.
- Added various options for configuring the CLI's environment (fonts, package paths, etc.)
- Text show rules now match across multiple text elements
- Block-level equations can now optionally break over multiple pages
- Fixed a bug where some fonts would not print correctly on professional printers
- Fixed a long-standing bug which could cause headings to be orphaned at the bottom of the page

Layout

- Added support for multi-column floating placement and figures via `place.scope`[3.10.19/4.2] and `figure.scope`[3.6.8/5.4]. Two-column documents should now prefer `set page(columns: 2)` over `show: column.with(2)` (see the Page Setup Guide[4.2.4]).
- Added support for automatic line numbering[3.6.15/5.1] (often used in academic papers)
- Added `par.spacing`[3.6.15/4.2] property for configuring paragraph spacing. This should now be used instead of `show par: set block(spacing: ..)` **(Breaking change)**
- Block-level elements like lists, grids, and stacks now show themselves as blocks and are thus affected by all block properties (e.g. `stroke`) rather than just `spacing` **(Breaking change)**
- Added `block.sticky`[3.10.4/2.13] property which prevents a page break after a block
- Added `place.flush`[3.10.19/5.1] function which forces all floating figures to be placed before any further content
- Added `skew`[3.10.25] function
- Added `auto` option for `page.header`[3.10.17/4.13] and `page.footer`[3.10.17/4.15] which results in an automatic header/footer based on the numbering (which was previously inaccessible after a change)
- Added `gap` and `justify` parameters to `repeat`[3.10.22] function
- Added `width` and `height` parameters to the `measure`[3.10.14] function to define the space in which the content should be measured. Especially useful in combination with `layout`[3.10.12].
- The height of a `block`, `image`, `rect`, `square`, `ellipse`, or `circle` can now be specified in fractional units[3.10.9]
- The `scale`[3.10.24] function now supports absolute lengths for `x`, `y`, `factor`. This way an element of unknown size can be scaled to a fixed size.
- The values of `block.above` and `block.below` can now be retrieved in context expressions.
- Increased accuracy of conversions between absolute units (pt, mm, cm, in)
- Fixed a bug which could cause headings to be orphaned at the bottom of the page
- Fixed footnotes within breakable blocks appearing on the page where the breakable block ends instead of at the page where the footnote marker is
- Fixed numbering of nested footnotes and footnotes in floats
- Fixed empty pages appearing when a context[3.4] expression wraps whole pages
- Fixed `set block(spacing: x)` behaving differently from `set block(above: x, below: x)`
- Fixed behavior of `rotate`[3.10.23] and `scale`[3.10.24] with `reflow: true`
- Fixed interaction of `align(horizon)` and `v(1fr)`
- Fixed various bugs where floating placement would yield overlapping results
- Fixed a bug where widow/orphan prevention would unnecessarily move text into the next column
- Fixed weak spacing[3.10.26/4.2] not being trimmed at the start and end of lines in a paragraph (only at the start and end of paragraphs)

[A] https://github.com/LaurenzV

- Fixed interaction of weak page break and `pagebreak.to`[3.10.18/2.2]
- Fixed compilation output of a single weak page break
- Fixed crash when padding[3.10.16] by `100%`

Text

- Tuned hyphenation: It is less eager by default and hyphenations close to the edges of words are now discouraged more strongly **(May lead to larger layout reflows)**
- New default font: Libertinus Serif. This is the maintained successor to the old default font Linux Libertine. **(May lead to smaller reflows)**
- Setting the font to an unavailable family will now result in a warning
- Implemented a new smart quote algorithm, fixing various bugs where smart quotes weren't all that smart
- Added `text.costs`[3.7.12/2.21] parameter for tweaking various parameters that affect the choices of the layout engine during text layout
- Added `typm` highlighting mode for math in raw blocks[3.7.6/4.3]
- Added basic i18n for Galician, Catalan, Latin, Icelandic, Hebrew
- Implemented hyphenation duplication for Czech, Croatian, Lower Sorbian, Polish, Portuguese, Slovak, and Spanish.
- The `smallcaps`[3.7.7] function is now an element function and can thereby be used in show(-set) rules.
- The `raw.theme`[3.7.6/4.6] parameter can now be set to `none` to disable highlighting even in the presence of a language tag, and to `auto` to reset to the default
- Multiple stylistic sets[3.7.12/2.24] can now be enabled at once
- Fixed the Chinese translation for "Equation"
- Fixed that hyphenation could occur outside of words
- Fixed incorrect layout of bidirectional text in edge cases
- Fixed layout of paragraphs with explicit trailing whitespace
- Fixed bugs related to empty paragraphs created via `#""`
- Fixed accidental trailing spaces for line breaks immediately preceding an inline equation
- Fixed `text.historical-ligatures`[3.7.12/2.27] not working correctly
- Fixed accidental repetition of Thai characters around line breaks in some circumstances
- Fixed smart quotes[3.7.8] for Swiss French
- New font metadata exceptions for Archivo, Kaiti SC, and Kaiti TC
- Updated bundled New Computer Modern fonts to version 6.0

Math

- Block-level equations can now break over multiple pages if enabled via `show math.equation: set block(breakable: true)`.
- Matrix and vector sizing is now more consistent across different cell contents
- Added `stretch`[3.8.22] function for manually or automatically stretching characters like arrows or parentheses horizontally or vertically
- Improved layout of attachments on parenthesized as well as under- or overlined expressions
- Improved layout of nested attachments resulting from code like `#let a0 = $a_0$; $a0^1$`
- Improved layout of primes close to superscripts
- Improved layout of fractions
- Typst now makes use of math-specific height-dependent kerning information in some fonts for better attachment layout
- The `floor` and `ceil` functions in math are now callable symbols, such that `$ floor(x) = lr(floor.l x floor.r) $`
- The `mat.delim`[3.8.18/2.1], `vec.delim`[3.8.27/2.1], and `cases.delim`[3.8.13/2.1] parameters now allow any character that is considered a delimiter or "fence" (e.g. |) by Unicode. The `delim: "||"` notation is *not* supported anymore and should be replaced by `delim: bar.double` **(Minor breaking change)**
- Added `vec.align`[3.8.27/2.2] and `mat.align`[3.8.18/2.2] parameters
- Added `underparen`[3.8.25/1.7], `overparen`[3.8.25/1.8], `undershell`[3.8.25/1.9], and `overshell`[3.8.25/1.10]
- Added ~ shorthand for `tilde.op` in math mode **(Minor breaking change)**
- Fixed baseline alignment of equation numbers
- Fixed positioning of corner brackets (⌜, ⌝, ⌞, ⌟)
- Fixed baseline of large roots

- Fixed multiple minor layout bugs with attachments
- Fixed that alignment points could affect line height in math
- Fixed that spaces could show up between text and invisible elements like `metadata`[3.12.5] in math
- Fixed a crash with recursive show rules in math
- Fixed `lr.size`[3.8.17/2.1.1] not affecting characters enclosed in `mid`[3.8.17/2.2] in some cases
- Fixed resolving of em units in sub- and superscripts
- Fixed bounding box of inline equations when a text edge[3.7.12/2.14] is set to `"bounds"`

Introspection

- Implemented a new system by which Typst tracks where elements end up on the pages. This may lead to subtly different behavior in introspections. **(Breaking change)**
- Fixed various bugs with wrong counter behavior in complex layout situations, through a new, more principled implementation
- Counter updates can now be before the first, in between, and after the last page when isolated by weak page breaks. This allows, for instance, updating a counter before the first page header and background.
- Fixed logical ordering of introspections within footnotes and figures
- Fixed incorrect `here().position()`[3.12.2] when `place`[3.10.19] was used in a context expression
- Fixed resolved positions of elements (in particular, headings) whose show rule emits an invisible element (like a state update) before a page break
- Fixed behavior of stepping a counter at a deeper level than its current state has
- Fixed citation formatting not working in table headers and a few other places
- Displaying the footnote counter will now respect the footnote numbering style

Model

- Document set rules do not need to be at the very start of the document anymore. The only restriction is that they must not occur inside of layout containers.
- The `spacing` property of lists[3.6.3/3.5], enumerations[3.6.12/3.8], and term lists[3.6.21/3.5] is now also respected for tight lists
- Tight lists now only attach (with tighter spacing) to preceding paragraphs, not arbitrary blocks
- The `quote`[3.6.17] element is now locatable (can be used in queries)
- The bibliography heading now uses `depth` instead of `level` so that its level can still be configured via a show-set rule
- Added support for more numbering[3.6.13] formats: Devanagari, Eastern Arabic, Bengali, and circled numbers
- Added `hanging-indent`[3.6.10/5.8] parameter to heading function to tweak the appearance of multi-line headings and improved default appearance of multi-line headings
- Improved handling of bidirectional text in outline entry
- Fixed document set rules being ignored in an otherwise empty document
- Fixed document set rules not being usable in context expressions
- Fixed bad interaction between `set document` and `set page`
- Fixed `show figure: set align(..)`. Since the default figure alignment is now a show-set rule, it is not revoked by `show figure: it => it.body` anymore. **(Minor breaking change)**
- Fixed numbering of footnote references
- Fixed spacing after bibliography heading

Bibliography

- The Hayagriva YAML `publisher` field can now accept a dictionary with a `location` key. The top-level `location` key is now primarily intended for event and item locations.
- Multiple page ranges with prefixes and suffixes are now allowed
- Added `director` and catch-all editor types to BibLaTeX parsing
- Added support for disambiguation to alphanumeric citation style
- The year 0 will now render as 1BC
- Fixes for sorting of bibliography entries
- Fixed pluralization of page range labels
- Fixed sorting of citations by their number
- Fixed how citation number ranges collapse

- Fixed when the short form of a title is used
- Fixed parsing of unbalanced dollars in BibLaTeX `url` field
- Updated built-in citation styles

Visualization

- Added `fill-rule` parameter to `path` and `polygon`[3.11.9/2.2] functions
- Fixed color mixing and gradients for Luma colors[3.11.3/4.1]
- Fixed conversion from Luma to CMYK colors
- Fixed offset gradient strokes in PNG export
- Fixed unintended cropping of some SVGs
- SVGs with foreign objects now produce a warning as they will likely not render correctly in Typst

Syntax

- Added support for nested imports like `import "file.typ": module.item`
- Added support for parenthesized imports like `import "file.typ": (a, b, c)`. With those, the import list can break over multiple lines.
- Fixed edge case in parsing of reference syntax
- Fixed edge case in parsing of heading, list, enum, and term markers immediately followed by comments
- Fixed rare crash in parsing of parenthesized expressions

Scripting

- Added new fixed-point `decimal`[3.5.10] number type for highly precise arithmetic on numbers in base 10, as needed for finance
- Added `std` module for accessing standard library definitions even when a variable with the same name shadows/overwrites it
- Added `array.to-dict`[3.5.2/3.32], `array.reduce`[3.5.2/3.33], `array.windows`[3.5.2/3.29] methods
- Added `exact` argument to `array.zip`[3.5.2/3.17]
- Added `arguments.at`[3.5.1/4.2] method
- Added `int.from-bytes`[3.5.16/3.8], `int.to-bytes`[3.5.16/3.9], `float.from-bytes`[3.5.14/3.4], and `float.to-bytes`[3.5.14/3.5]
- Added proper support for negative values of the `digits` parameter of `calc.round`[3.5.7/1.31] (the behaviour existed before but was subtly broken)
- Conversions from `int`[3.5.16] to `float`[3.5.14] will now error instead of saturating if the float is too large **(Minor breaking change)**
- Added `float.nan` and `float.inf`, removed `calc.nan` **(Minor breaking change)**
- Certain symbols are now generally callable like functions and not only specifically in math. Examples are accents or `floor`[3.8.17/2.5] and `ceil`[3.8.17/2.6].
- Improved `repr`[3.5.24] of relative values, sequences, infinities, NaN, `type(none)` and `type(auto)`
- Fixed crash on whole packages (rather than just files) cyclically importing each other
- Fixed return type of `calc.round`[3.5.7/1.31] on integers when a non-zero value is provided for `digits`

Styling

- Text show rules now match across multiple text elements
- The string `"` in a text show rule now matches smart quotes
- Fixed a long-standing styling bug where the header and footer would incorrectly inherit styles from a lone element on the page (e.g. a heading)
- Fixed `set page` not working directly after a counter/state update
- Page fields configured via an explicit `page(..)[..]` call can now be properly retrieved in context expressions

Export

- Highly reduced PDF file sizes due to better font subsetting
- Emoji are now exported properly in PDF

- Added initial support for PDF/A. For now, only the standard PDF/A-2b is supported, but more is planned for the future. Enabled via `--pdf-standard a-2b` in the CLI and via the UI in File > Export as > PDF in the web app.
- Setting `page.fill`[3.10.17/4.9] to `none` will now lead to transparent pages instead of white ones in PNG and SVG. The new default of `auto` means transparent for PDF and white for PNG and SVG.
- Improved text copy-paste from PDF in complex scenarios
- Exported SVGs now contain the `data-typst-label` attribute on groups resulting from labelled boxes[3.10.5] and blocks[3.10.4]
- Fixed a bug where some fonts would not print correctly on professional printers
- Fixed a bug where transparency could leak from one PDF object to another
- Fixed a bug with CMYK gradients in PDF
- Fixed various bugs with export of Oklab gradients in PDF
- Fixed crashes related to rendering of non-outline glyphs
- Two small fixes for PDF standard conformance

Performance

- Typst's layout engine is now multithreaded. Typical speedups are 2-3x for larger documents. The multi-threading operates on page break boundaries, so explicit page breaks are necessary for it to kick in.
- Paragraph justification was optimized with a new two-pass algorithm. Speedups are larger for shorter paragraphs and range from 1-6x.

Command Line Interface

- Added `--pages` option to select specific page ranges to export
- Added `--package-path` and `--package-cache-path` as well as `TYPST_PACKAGE_PATH` and `TYPST_PACKAGE_CACHE_PATH` environment variables for configuring where packages are loaded from and cached in, respectively
- Added `--ignore-system-fonts` flag to disable system fonts fully for better reproducibility
- Added `--make-deps` argument for outputting the dependencies of the current compilation as a Makefile
- Added `--pretty` option to `typst query`, with the default now being to minify (only applies to JSON format)
- Added `--backup-path` to `typst update` to configure where the previous version is backed up
- Added useful links to help output
- The CLI will now greet users who invoke just `typst` for the first time
- The document can now be written to stdout by passing `-` as the output filename (for PDF or single-page image export)
- Typst will now emit a proper error message instead of failing silently when the certificate specified by `--cert` or `TYPST_CERT` could not be loaded
- The CLI now respects the `SOURCE_DATE_EPOCH` environment variable for better reproducibility
- When exporting multiple images, you can now use `t` (total pages), `p` (current page), and `0p` (zero-padded current page, same as current `n`) in the output path
- The input and output paths now allow non-UTF-8 values
- Times are now formatted more consistently across the CLI
- Fixed a bug related to the `--open` flag
- Fixed path completions for `typst` not working in zsh

Tooling and Diagnostics

- The "compiler" field for specifying the minimum Typst version required by a package now supports imprecise bounds like 0.11 instead of 0.11.0
- Added warning when a label is ignored by Typst because no preceding labellable element exists
- Added hint when trying to apply labels in code mode
- Added hint when trying to call a standard library function that has been shadowed/overwritten by a local definition
- Added hint when trying to set both the language and the region in the `lang` parameter
- Added hints when trying to compile non-Typst files (e.g. after having typed `typst c file.pdf` by accident)
- Added hint when a string is used where a label is expected
- Added hint when a stray end of a block comment (`*/`) is encountered

- Added hints when destructuring arrays with the wrong number of elements
- Improved error message when trying to use a keyword as an identifier in a let binding
- Improved error messages when accessing nonexistent fields
- Improved error message when a package exists, but not the specified version
- Improved hints for unknown variables
- Improved hint when trying to convert a length with non-zero em component to an absolute unit
- Fixed a crash that could be triggered by certain hover tooltips
- Fixed an off-by-one error in to-source jumps when first-line-indent is enabled
- Fixed suggestions for `.` after the end of an inline code expressions
- Fixed autocompletions being duplicated in a specific case

Symbols

- New: `parallelogram`, `original`, `image`, `crossmark`, `rest`, `natural`, `flat`, `sharp`, `tiny`, `miny`, `copyleft`, `trademark`, `emoji.beet`, `emoji.fingerprint`, `emoji.harp`, `emoji.shovel`, `emoji.splatter`, `emoji.tree.leafless`,
- New variants: `club.stroked`, `diamond.stroked`, `heart.stroked`, `spade.stroked`, `gt.neq`, `lt.neq`, `checkmark.heavy`, `paren.double`, `brace.double`, `shell.double`, `arrow.turn`, `plus.double`, `plus.triple`, `infinity.bar`, `infinity.incomplete`, `infinity.tie`, `multimap.double`, `ballot.check`, `ballot.check.heavy`, `emptyset.bar`, `emptyset.circle`, `emptyset.arrow.l`, `emptyset.arrow.r`, `parallel.struck`, `parallel.eq`, `parallel.equiv`, `parallel.slanted`, `parallel.tilde`, `angle.l.curly`, `angle.l.dot`, `angle.r.curly`, `angle.r.dot`, `angle.oblique`, `angle.s`, `em.two`, `em.three`
- Renamed: `turtle` to `shell`, `notes` to `note`, `ballot.x` to `ballot.cross`, `succ.eq` to `succ.curly.eq`, `prec.eq` to `prec.curly.eq`, `servicemark` to `trademark.service`, `emoji.face.tired` to `emoji.face.distress` **(Breaking change)**
- Changed codepoint: `prec.eq`, `prec.neq`, `succ.eq`, `succ.neq`, `triangle` from ▷ to △, `emoji.face.tired` **(Breaking change)**
- Removed: `lt.curly` in favor of `prec`, `gt.curly` in favor of `succ` **(Breaking change)**

Deprecations

- `counter.display`[3.12.1/11.2] without an established context
- `counter.final`[3.12.1/11.4] with a location
- `state.final`[3.12.7/6.3] with a location
- `state.display`
- `query`[3.12.6] with a location as the second argument
- `locate`[3.12.3] with a callback function
- `measure`[3.10.14] with styles
- `style`

Development

- Added `typst-kit` crate which provides useful APIs for `World` implementors
- Added go-to-definition API in `typst-ide`
- Added package manifest parsing APIs to `typst-syntax`
- As the compiler is now capable of multithreading, `World` implementations must satisfy `Send` and `Sync`
- Changed signature of `World::main` to allow for the scenario where the main file could not be loaded
- Removed `Tracer` in favor of `Warned<T>` and `typst::trace` function
- The `xz2` dependency used by the self-updater is now statically linked
- The Dockerfile now has an `ENTRYPOINT` directive

Typst 0.11.1 (May 17, 2024)

Security

- Fixed a vulnerability where image files at known paths could be embedded into the PDF even if they were outside of the project directory

Bibliography

- Fixed et-al handling in subsequent citations
- Fixed suppression of title for citations and bibliography references with no author
- Fixed handling of initials in citation styles without a delimiter
- Fixed bug with citations in footnotes

Text and Layout

- Fixed interaction of `first-line-indent`[3.6.15/4.6] and `outline`[3.6.14]
- Fixed compression of CJK punctuation marks at line start and end
- Fixed handling of rectangles[3.11.10] with negative dimensions
- Fixed layout of `path` in explicitly sized container
- Fixed broken `raw`[3.7.6] text in right-to-left paragraphs
- Fixed tab rendering in `raw` text with language `typ` or `typc`
- Fixed highlighting of multi-line `raw` text enclosed by single backticks
- Fixed indentation of overflowing lines in `raw` blocks
- Fixed extra space when `raw` text ends with a backtick

Math

- Fixed broken equations[3.8.15] in right-to-left paragraphs
- Fixed missing blackboard bold[3.8.26/1.5] letters
- Fixed error on empty arguments in 2D math argument list
- Fixed stretching via `mid`[3.8.17/2.2] for various characters
- Fixed that alignment points in equations were affected by `set align(..)`

Export

- Fixed smart quotes[3.7.8] in PDF outline
- Fixed patterns[3.11.13] with spacing in PDF
- Fixed wrong PDF page labels when page numbering[3.10.17/4.10] was disabled after being previously enabled

Scripting

- Fixed overflow for large numbers in external data files (by converting to floats instead)
- Fixed `str.trim(regex, at: end)`[3.5.27/4.19] when the whole string is matched

Miscellaneous

- Fixed deformed strokes for specific shapes and thicknesses
- Fixed newline handling in code mode: There can now be comments within chained method calls and between an `if` branch and the `else` keyword
- Fixed inefficiency with incremental reparsing
- Fixed autocompletions for relative file imports
- Fixed crash in autocompletion handler
- Fixed a bug where the path and entrypoint printed by `typst init` were not properly escaped
- Fixed various documentation errors

Typst 0.11.0 (March 15, 2024)

Tables

- Tables are now *much* more flexible, read the new Table Guide[4.3] to get started
- Added `table.cell`[3.6.20/4.1] element for per-cell configuration
- Cells can now span multiple columns[3.6.20/4.1.4] or rows[3.6.20/4.1.5]
- The stroke[3.6.20/4.1.9] of individual cells can now be customized
- The `align`[3.6.20/3.7] and `inset`[3.6.20/3.6] arguments of the table function now also take `(x, y) => ..` functions
- Added `table.hline`[3.6.20/4.2] and `table.vline`[3.6.20/4.3] for convenient line customization
- Added `table.header`[3.6.20/4.4] element for table headers that repeat on every page
- Added `table.footer`[3.6.20/4.5] element for table footers that repeat on every page
- All the new table functionality is also available for grids[3.10.10]
- Fixed gutter-related bugs

Thanks to @PgBiel[A] for his work on tables!

Templates

- You can now use template packages to get started with new projects. Click *Start from template* on the web app's dashboard and choose your preferred template or run the `typst init <template>` command in the CLI. You can browse the available templates here[B].
- Switching templates after the fact has become easier. You can just import a styling function from a different template package.
- Package authors can now submit their own templates to the package repository[C]. Share a template for a paper, your institution, or an original work to help the community get a head start on their projects.
- Templates and packages are now organized by category and discipline. Filter packages by either taxonomy in the *Start from template* wizard. If you are a package author, take a look at the new documentation for categories[D] and disciplines[E].

Context

- Added *context expressions:* Read the chapter on context[3.4] to get started
- With context, you can access settable properties, e.g. `context text.lang` to access the language set via `set text(lang: "..")`
- The following existing functions have been made contextual: `query`[3.12.6], `locate`[3.12.3], `measure`[3.10.14], `counter.display`[3.12.1/11.2], `counter.at`[3.12.1/11.3], `counter.final`[3.12.1/11.4], `state.at`[3.12.7/6.2], and `state.final`[3.12.7/6.3]
- Added contextual methods `counter.get`[3.12.1/11.1] and `state.get`[3.12.7/6.1] to retrieve the value of a counter or state in the current context
- Added contextual function `here`[3.12.2] to retrieve the location[3.12.4] of the current context
- The `locate`[3.12.3] function now returns the location of a selector's unique match. Its old behavior has been replaced by context expressions and only remains temporarily available for compatibility.
- The `counter.at`[3.12.1/11.3] and `state.at`[3.12.7/6.2] methods are now more flexible: They directly accept any kind of locatable[3.12.4/1] selector with a unique match (e.g. a label) instead of just locations
- When context is available, `counter.display`[3.12.1/11.2] now directly returns the result of applying the numbering instead of yielding opaque content. It should not be used anymore without context. (Deprecation planned)
- The `state.display` function should not be used anymore, use `state.get`[3.12.7/6.1] instead (Deprecation planned)
- The `location` argument of `query`[3.12.6], `counter.final`[3.12.1/11.4], and `state.final`[3.12.7/6.3] should not be used anymore (Deprecation planned)
- The `styles` argument of the `measure` function should not be used anymore (Deprecation planned)
- The `style` function should not be used anymore, use context instead (Deprecation planned)

[A] https://github.com/PgBiel
[B] https://typst.app/universe/search/?kind=templates
[C] https://github.com/typst/packages
[D] https://github.com/typst/packages/blob/main/docs/CATEGORIES.md
[E] https://github.com/typst/packages/blob/main/docs/DISCIPLINES.md

- The correct context is now also provided in various other places where it is available, e.g. in show rules, layout callbacks, and numbering functions in the outline

Styling

- Fixed priority of multiple show-set rules[3.2.2]: They now apply in the same order as normal set rules would
- Show-set rules on the same element (e.g. `show heading.where(level: 1): set heading(numbering: "1.")`) now work properly
- Setting properties on an element within a transformational show rule (e.g. `show heading: it => { set heading(..); it }`) is **not** supported anymore (previously it also only worked sometimes); use show-set rules instead **(Breaking change)**
- Text show rules that match their own output now work properly (e.g. `show "cmd": \`cmd\``)
- The elements passed to show rules and returned by queries now contain all fields of their respective element functions rather than just specific ones
- All settable properties can now be used in where[3.5.15/8.2] selectors
- And[3.5.25/3.2] and or[3.5.25/3.1] selectors can now be used with show rules
- Errors within show rules and context expressions are now ignored in all but the last introspection iteration, in line with the behavior of the old `locate`[3.12.3]
- Fixed a bug where document set rules were allowed after content

Layout

- Added `reflow` argument to `rotate`[3.10.23] and `scale`[3.10.24] which lets them affect the layout
- Fixed a bug where floating placement[3.10.19/4.3] or floating figures[3.6.8/5.3] could end up out of order
- Fixed overlap of text and figure for full-page floating figures
- Fixed various cases where the `hide`[3.10.11] function didn't hide its contents properly
- Fixed usage of h[3.10.26] and v[3.10.27] in stacks[3.10.28]
- Invisible content like a counter update will no longer force a visible block for just itself
- Fixed a bug with horizontal spacing followed by invisible content (like a counter update) directly at the start of a paragraph

Text

- Added `stroke`[3.7.12/2.8] property for text
- Added basic i18n for Serbian and Catalan
- Added support for contemporary Japanese numbering[3.6.13] method
- Added patches for various wrong metadata in specific fonts
- The text direction[3.7.12/2.19] can now be overridden within a paragraph
- Fixed Danish smart quotes[3.7.8]
- Fixed font fallback next to a line break
- Fixed width adjustment of JIS-style Japanese punctuation
- Fixed Finnish translation of "Listing"
- Fixed Z-ordering of multiple text decorations (underlines, etc.)
- Fixed a bug due to which text features[3.7.12/2.32] could not be overridden in consecutive set rules

Model

- Added `depth`[3.6.10/5.2] and `offset`[3.6.10/5.3] arguments to heading to increase or decrease the heading level for a bunch of content; the heading syntax now sets `depth` rather than `level` **(Breaking change)**
- List markers[3.6.3/3.2] now cycle by default
- The `quote`[3.6.17] function now more robustly selects the correct quotes based on language and nesting
- Fixed indent bugs related to the default show rule of terms[3.6.21]

Math

- Inline equations now automatically linebreak at appropriate places
- Added `number-align`[3.8.15/3.3] argument to equations
- Added support for adjusting the `size`[3.8.9/2.3] of accents relative to their base

- Improved positioning of accents
- Primes[3.8.19] are now always attached as scripts[3.8.10/3.2] by default
- Exposed `math.primes`[3.8.19] element which backs the `$f'$` syntax in math
- Math mode is not affected by `strong`[3.6.19] and `emph`[3.6.7] anymore
- Fixed `attach`[3.8.10/3.1] under fractions[3.8.16]
- Fixed that `math.class`[3.8.14] did not affect smart limit placement
- Fixed weak spacing in `lr`[3.8.17/2.1] groups
- Fixed layout of large operators for Cambria Math font
- Fixed math styling of Hebrew symbol codepoints

Symbols

- Added `gradient` as an alias for `nabla`
- Added `partial` as an alias for `diff`, `diff` will be deprecated in the future
- Added `colon.double`, `gt.approx`, `gt.napprox`, `lt.approx`, and `lt.napprox`
- Added `arrow.r.tilde` and `arrow.l.tilde`
- Added `tilde.dot`
- Added `forces` and `forces.not`
- Added `space.nobreak.narrow`
- Added `lrm` (Left-to-Right Mark) and `rlm` (Right-to-Left Mark)
- Fixed `star.stroked` symbol (which previously had the wrong codepoint)

Scripting

- Arrays can now be compared lexicographically
- Added contextual method `to-absolute`[3.10.13/3.5] to lengths
- Added `calc.root`[3.5.7/1.5]
- Added `int.signum`[3.5.16/3.1] and `float.signum`[3.5.14/3.3] methods
- Added `float.is-nan`[3.5.14/3.1] and `float.is-infinite`[3.5.14/3.2] methods
- Added `int.bit-not`[3.5.16/3.2], `int.bit-and`[3.5.16/3.3], `int.bit-or`[3.5.16/3.4], `int.bit-xor`[3.5.16/3.5], `int.bit-lshift`[3.5.16/3.6], and `int.bit-rshift`[3.5.16/3.7] methods
- Added `array.chunks`[3.5.2/3.28] method
- A module can now be converted to a dictionary with the dictionary constructor[3.5.11/2] to access its contents dynamically
- Added `row-type`[3.13.3/2.3] argument to `csv` function to configure how rows will be represented
- XML parsing[3.13.7] now allows DTDs (document type definitions)
- Improved formatting of negative numbers with `str`[3.5.27] and `repr`[3.5.24]
- For loops can now iterate over bytes[3.5.6]
- Fixed a bug with pattern matching in for loops
- Fixed a bug with labels not being part of `.fields()`[3.5.8/3.4] dictionaries
- Fixed a bug where unnamed argument sinks wouldn't capture excess arguments
- Fixed typo in `repr` output of strokes

Syntax

- Added support for nested destructuring patterns[3.3.3]
- Special spaces (like thin or non-breaking spaces) are now parsed literally instead of being collapsed into normal spaces **(Breaking change)**
- Korean text can now use emphasis syntax without adding spaces **(Breaking change)**
- The token `context`[3.4] is now a keyword and cannot be used as an identifier anymore **(Breaking change)**
- Nested line comments aren't allowed anymore in block comments **(Breaking change)**
- Fixed a bug where `x.)` would be treated as a field access
- Text elements can now span across curly braces in markup
- Fixed silently wrong parsing when function name is parenthesized
- Fixed various bugs with parsing of destructuring patterns, arrays, and dictionaries

Tooling & Diagnostics

- Click-to-jump now works properly within `raw`[3.7.6] text
- Added suggestion for accessing a field if a method doesn't exist
- Improved hint for calling a function stored in a dictionary
- Improved errors for mutable accessor functions on arrays and dictionaries
- Fixed error message when calling constructor of type that doesn't have one
- Fixed confusing error message with nested dictionaries for strokes on different sides
- Fixed autocompletion for multiple packages with the same name from different namespaces

Visualization

- The `image`[3.11.7] function doesn't upscale images beyond their natural size anymore
- The `image`[3.11.7] function now respects rotation stored in EXIF metadata
- Added support for SVG filters
- Added alpha component to `luma`[3.11.3/4.1] colors
- Added `color.transparentize`[3.11.3/4.19] and `color.opacify`[3.11.3/4.20] methods
- Improved `color.negate`[3.11.3/4.16] function
- Added `stroke`[3.7.1/2.2] and `radius`[3.7.1/2.6] arguments to `highlight` function
- Changed default `highlight`[3.7.1] color to be transparent
- CMYK to RGB conversion is now color-managed
- Fixed crash with gradients in Oklch color space
- Fixed color-mixing for hue-based spaces
- Fixed bugs with color conversion
- SVG sizes are not rounded anymore, preventing slightly wrong aspect ratios
- Fixed a few other SVG-related bugs
- `color.components`[3.11.3/4.9] doesn't round anything anymore

Export

- PDFs now contain named destinations for headings derived from their labels
- The internal PDF structure was changed to make it easier for external tools to extract or modify individual pages, avoiding a bug with Typst PDFs in Apple Preview
- PDFs produced by Typst should now be byte-by-byte reproducible when `set document(date: none)` is set
- Added missing flag to PDF annotation
- Fixed multiple bugs with gradients in PDF export
- Fixed a bug with patterns in PDF export
- Fixed a bug with embedding of grayscale images in PDF export
- Fixed a bug with To-Unicode mapping of CFF fonts in PDF export
- Fixed a bug with the generation of the PDF outline
- Fixed a sorting bug in PDF export leading to non-reproducible output
- Fixed a bug with transparent text in PNG export
- Exported SVG files now include units in their top-level `width` and `height`

Command line interface

- Added support for passing inputs[3.5.29] via a CLI flag
- When passing the filename `-`, Typst will now read input from stdin
- Now uses the system-native TLS implementation for network fetching which should be generally more robust
- Watch mode will now properly detect when a previously missing file is created
- Added `--color` flag to configure whether to print colored output
- Fixed user agent with which packages are downloaded
- Updated bundled fonts to the newest versions

Development

- Added `--vendor-openssl` to CLI to configure whether to link OpenSSL statically instead of dynamically (not applicable to Windows and Apple platforms)
- Removed old tracing (and its verbosity) flag from the CLI
- Added new `--timings` flag which supersedes the old flamegraph profiling in the CLI
- Added minimal CLI to `typst-docs` crate for extracting the language and standard library documentation as JSON
- The `typst_pdf::export` function's `ident` argument switched from `Option` to `Smart`. It should only be set to `Smart::Custom` if you can provide a stable identifier (like the web app can). The CLI sets `Smart::Auto`.

Typst 0.10.0 (December 04, 2023)

Bibliography management

- Added support for citation collapsing (e.g. `[1]–[3]` instead of `[1], [2], [3]`) if requested by a CSL style
- Fixed bug where an additional space would appear after a group of citations
- Fixed link show rules for links in the bibliography
- Fixed show-set rules on citations
- Fixed bibliography-related crashes that happened on some systems
- Corrected name of the GB/T 7714 family of styles from 7114 to 7714
- Fixed missing title in some bibliography styles
- Fixed printing of volumes in some styles
- Fixed delimiter order for contributors in some styles (e.g. APA)
- Fixed behavior of alphanumeric style
- Fixed multiple bugs with GB/T 7714 style
- Fixed escaping in Hayagriva values
- Fixed crashes with empty dates in Hayagriva files
- Fixed bug with spacing around math blocks
- Fixed title case formatting after verbatim text and apostrophes
- Page ranges in `.bib` files can now be arbitrary strings
- Multi-line values in `.bib` files are now parsed correctly
- Entry keys in `.bib` files now allow more characters
- Fixed error message for empty dates in `.bib` files
- Added support for years of lengths other than 4 without leading zeros in `.bib` files
- More LaTeX commands (e.g. for quotes) are now respected in `.bib` files

Visualization

- Added support for patterns[3.11.13] as fills and strokes
- The `alpha` parameter of the `components`[3.11.3/4.9] function on colors is now a named parameter (**Breaking change**)
- Added support for the Oklch[3.11.3/4.3] color space
- Improved conversions between colors in different color spaces
- Removed restrictions on Oklab[3.11.3/4.2] chroma component
- Fixed clipping[3.10.4/2.12] on blocks and boxes without a stroke
- Fixed bug with gradients[3.11.6] on math
- Fixed bug with gradient rotation on text
- Fixed bug with gradient colors in PDF
- Fixed relative base of Oklab chroma ratios
- Fixed Oklab color negation

Text and Layout

- CJK text can now be emphasized with the `*` and `_` syntax even when there are no spaces
- Added basic i18n for Greek and Estonian
- Improved default figure caption separator[3.6.8/6.1.2] for Chinese, French, and Russian
- Changed default figure supplement[3.6.8/5.7] for Russian to short form
- Fixed CJK-Latin-spacing[3.7.12/2.11] before line breaks and in `locate`[3.12.3] calls
- Fixed line breaking at the end of links

Math

- Added `mid`[3.8.17/2.2] function for scaling a delimiter up to the height of the surrounding `lr`[3.8.17/2.1] group
- The `op`[3.8.24] function can now take any content, not just strings
- Improved documentation for math alignment[3.8.5]
- Fixed swallowing of trailing comma when a symbol is used in a function-like way (e.g. `pi(a,b,)`)

Scripting

- Any non-identifier dictionary key is now interpreted as an expression: For instance, `((key): value)` will create a dictionary with a dynamic key
- The `stroke`[3.11.12] type now has a constructor that converts a value to a stroke or creates one from its parts
- Added constructor for `arguments`[3.5.1] type
- Added `calc.div-euclid`[3.5.7/1.38] and `calc.rem-euclid`[3.5.7/1.39] functions
- Fixed equality of `arguments`[3.5.1]
- Fixed `repr`[3.5.24] of `cmyk`[3.11.3/4.6] colors
- Fixed crashes with provided elements like figure captions, outline entries, and footnote entries

Tooling and Diagnostics

- Show rules that match on their own output now produce an appropriate error message instead of a crash (this is a first step, in the future they will just work)
- Too highly or infinitely nested layouts now produce error messages instead of crashes
- Added hints for invalid identifiers
- Added hint when trying to use a manually constructed footnote or outline entry
- Added missing details to autocompletions for types
- Improved error message when passing a named argument where a positional one is expected
- Jump from click now works on raw blocks

Export

- PDF compilation output is now again fully byte-by-byte reproducible if the document's `date`[3.6.6/3.7] is set manually
- Fixed color export in SVG
- Fixed PDF metadata encoding of multiple authors[3.6.6/3.4]

Command line interface

- Fixed a major bug where `typst watch` would confuse files and fail to pick up updates
- Fetching of the release metadata in `typst update` now respects proxies
- Fixed bug with `--open` flag on Windows when the path contains a space
- The `TYPST_FONT_PATHS` environment variable can now contain multiple paths (separated by `;` on Windows and `:` elsewhere)
- Updated embedded New Computer Modern fonts to version 4.7
- The watching process doesn't stop anymore when the main file contains invalid UTF-8

Miscellaneous Improvements

- Parallelized image encoding in PDF export
- Improved the internal representation of content for improved performance
- Optimized introspection (query, counter, etc.) performance
- The document title[3.6.6/3.3] can now be arbitrary content instead of just a string
- The `number-align`[3.6.12/3.9] parameter on numbered lists now also accepts vertical alignments
- Fixed selectors on quote[3.6.17] elements
- Fixed parsing of `#return` expression in markup
- Fixed bug where inline equations were displayed in equation outlines
- Fixed potential CRLF issue in `raw`[3.7.6] blocks
- Fixed a bug where Chinese numbering couldn't exceed the number 255

Development

- Merged `typst` and `typst-library` and extracted `typst-pdf`, `typst-svg`, and `typst-render` into separate crates
- The Nix flake now includes the git revision when running `typst --version`

Typst 0.9.0 (October 31, 2023)

Bibliography management

- New bibliography engine based on CSL[A] (Citation Style Language). Ships with about 100 commonly used citation styles and can load custom `.csl` files.
- Added new `form`[3.6.4/3.3] argument to the `cite` function to produce different forms of citations (e.g. for producing a citation suitable for inclusion in prose)
- The `cite`[3.6.4] function now takes only a single label/key instead of allowing multiple. Adjacent citations are merged and formatted according to the citation style's rules automatically. This works both with the reference syntax and explicit calls to the `cite` function. **(Breaking change)**
- The `cite` function now takes a label[3.5.17] instead of a string **(Breaking change)**
- Added `full`[3.6.2/4.3] argument to bibliography function to print the full bibliography even if not all works were cited
- Bibliography entries can now contain Typst equations (wrapped in `$..$` just like in markup), this works both for `.yaml` and `.bib` bibliographies
- The hayagriva YAML format was improved. See its changelog[B] for more details. **(Breaking change)**
- A few bugs with `.bib` file parsing were fixed
- Removed `brackets` argument of `cite` function in favor of `form`

Visualization

- Gradients and colors (thanks to @Dherse[C])
  - Added support for gradients[3.11.6] on shapes and text
  - Supports linear, radial, and conic gradients
  - Added support for defining colors in more color spaces, including Oklab[3.11.3/4.2], Linear RGB(A)[3.11.3/4.4], HSL[3.11.3/4.7], and HSV[3.11.3/4.8]
  - Added `saturate`[3.11.3/4.14], `desaturate`[3.11.3/4.15], and `rotate`[3.11.3/4.17] functions on colors
  - Added `color.map`[3.11.3/3] module with predefined color maps that can be used with gradients
  - Rename `kind` function on colors to `space`[3.11.3/4.10]
  - Removed `to-rgba`, `to-cmyk`, and `to-luma` functions in favor of a new `components`[3.11.3/4.9] function
- Improved rendering of rectangles[3.11.10] with corner radius and varying stroke widths
- Added support for properly clipping boxes[3.10.5/2.9] and blocks[3.10.4/2.12] with a border radius
- Added `background` parameter to `overline`[3.7.5], `underline`[3.7.13], and `strike`[3.7.9] functions
- Fixed inaccurate color embedding in PDFs
- Fixed ICC profile handling for images embedded in PDFs

Text and Layout

- Added support for automatically adding proper spacing[3.7.12/2.11] between CJK and Latin text (enabled by default)
- Added support for automatic adjustment of more CJK punctuation
- Added `quote`[3.6.17] element for inserting inline and block quotes with optional attributions
- Added `raw.line`[3.7.6/5.1] element for customizing the display of individual lines of raw text, e.g. to add line numbers while keeping proper syntax highlighting
- Added support for per-side inset[3.6.20/3.6] customization to table function
- Added Hungarian and Romanian translations
- Added support for Czech hyphenation
- Added support for setting custom smart quotes[3.7.8]
- The default figure separator[3.6.8/6.1.2] now reacts to the currently set language and region
- Improved line breaking of links / URLs (especially helpful for bibliographies with many URLs)
- Improved handling of consecutive hyphens in justification algorithm
- Fixed interaction of justification and hanging indent
- Fixed a bug with line breaking of short lines without spaces when justification is enabled
- Fixed font fallback for hyphen generated by hyphenation

[A] https://citationstyles.org/
[B] https://github.com/typst/hayagriva/blob/main/CHANGELOG.md
[C] https://github.com/Dherse

- Fixed handling of word joiner and other no-break characters during hyphenation
- Fixed crash when hyphenating after an empty line
- Fixed line breaking of composite emoji like 🏳️‍🌈
- Fixed missing text in some SVGs
- Fixed font fallback in SVGs
- Fixed behavior of `to`[3.10.18/2.2] argument on `pagebreak` function
- Fixed `set align(..)` for equations
- Fixed spacing around placed[3.10.19] elements
- Fixed coalescing of `above`[3.10.4/2.10] and `below`[3.10.4/2.11] spacing if given in em units and the font sizes differ
- Fixed handling of `extent` parameter of `underline`[3.7.13], `overline`[3.7.5], and `strike`[3.7.9] functions
- Fixed crash for floating placed elements[3.10.19/4.3] with no specified vertical alignment
- Partially fixed a bug with citations in footnotes

Math

- Added `gap` argument for `vec`[3.8.27/2.3], `mat`[3.8.18/2.4], and `cases`[3.8.13/2.3] function
- Added `size` argument for `abs`[3.8.17/2.3], `norm`[3.8.17/2.4], `floor`[3.8.17/2.5], `ceil`[3.8.17/2.6], and `round`[3.8.17/2.7] functions
- Added `reverse`[3.8.13/2.2] parameter to cases function
- Added support for multinomial coefficients to `binom`[3.8.11] function
- Removed `rotation` argument on `cancel`[3.8.12] function in favor of a new and more flexible `angle` argument (**Breaking change**)
- Added `wide` constant, which inserts twice the spacing of `quad`
- Added `csch` and `sech` operators[3.8.24]
- ↼, ⇀, ↮, and ↔ can now be used as accents[3.8.9]
- Added `integral.dash`, `integral.dash.double`, and `integral.slash` symbols[3.9.2]
- Added support for specifying negative indices for augmentation[3.8.18/2.3] lines to position the line from the back
- Fixed default color of matrix augmentation[3.8.18/2.3] lines
- Fixed attachment of primes to inline expressions
- Math content now respects the text baseline[3.7.12/2.12] setting

Performance

- Fixed a bug related to show rules in templates which would effectively disable incremental compilation in affected documents
- Micro-optimized code in several hot paths, which brings substantial performance gains, in particular in incremental compilations
- Improved incremental parsing, which affects the whole incremental compilation pipeline
- Added support for incremental parsing in the CLI
- Added support for incremental SVG encoding during PDF export, which greatly improves export performance for documents with many SVG

Tooling and Diagnostics

- Improved autocompletion for variables that are in-scope
- Added autocompletion for package imports
- Added autocompletion for labels[3.5.17]
- Added tooltip that shows which variables a function captures (when hovering over the equals sign or arrow of the function)
- Diagnostics are now deduplicated
- Improved diagnostics when trying to apply unary + or − to types that only support binary + and −
- Error messages now state which label or citation key isn't present in the document or its bibliography
- Fixed a bug where function argument parsing errors were shadowed by function execution errors (e.g. when trying to call `array.sorted`[3.5.2/3.30] and passing the key function as a positional argument instead of a named one).

Export

- Added support for configuring the document's creation `date`[3.6.6/3.7]. If the `date` is set to `auto` (the default), the PDF's creation date will be set to the current date and time.
- Added support for configuring document `keywords`[3.6.6/3.6]
- Generated PDFs now contain PDF document IDs
- The PDF creator tool metadata now includes the Typst version

Web app

- Added version picker to pin a project to an older compiler version (with support for Typst 0.6.0+)
- Fixed desyncs between editor and compiler and improved overall stability
- The app now continues to highlight the document when typing while the document is being compiled

Command line interface

- Added support for discovering fonts through fontconfig
- Now clears the screen instead of resetting the terminal
- Now automatically picks correct file extension for selected output format
- Now only regenerates images for changed pages when using `typst watch` with PNG or SVG export

Miscellaneous Improvements

- Added `version`[3.5.32] type and `sys.version` constant specifying the current compiler version. Can be used to gracefully support multiple versions.
- The U+2212 MINUS SIGN is now used when displaying a numeric value, in the `repr`[3.5.24] of any numeric value and to replace a normal hyphen in text mode when before a digit. This improves, in particular, how negative integer values are displayed in math mode.
- Added support for specifying a default value instead of failing for `remove` function in array[3.5.2/3.8] and dictionary[3.5.11/3.4]
- Simplified Page Setup Guide examples
- Switched the documentation from using the word "hashtag" to the word "hash" where appropriate
- Added support for `array.zip`[3.5.2/3.17] without any further arguments
- Fixed crash when a plugin tried to read out of bounds memory
- Fixed crashes when handling infinite lengths[3.10.13]
- Fixed introspection (mostly bibliography) bugs due to weak page break close to the end of the document

Development

- Extracted `typst::ide` into separate `typst_ide` crate
- Removed a few remaining `'static` bounds on `&dyn World`
- Removed unnecessary dependency, which reduces the binary size
- Fixed compilation of `typst` by itself (without `typst-library`)
- Fixed warnings with Nix flake when using `lib.getExe`

Typst 0.8.0 (September 13, 2023)

Scripting

- Plugins (thanks to @astrale-sharp[A] and @arnaudgolfouse[B])
  - Typst can now load plugins[3.5.22] that are compiled to WebAssembly
  - Anything that can be compiled to WebAssembly can thus be loaded as a plugin
  - These plugins are fully encapsulated (no access to file system or network)
  - Plugins can be shipped as part of packages[3.3.9]
  - Plugins work just the same in the web app
- Types are now first-class values **(Breaking change)**
  - A type[3.5.31] is now itself a value
  - Some types can be called like functions (those that have a constructor), e.g. `int`[3.5.16] and `str`[3.5.27]
  - Type checks are now of the form `type(10) == int` instead of the old `type(10) == "integer"`. Compatibility with the old way will remain for a while to give package authors time to upgrade, but it will be removed at some point.
  - Methods are now syntax sugar for calling a function scoped to a type, meaning that `"hello".len()` is equivalent to `str.len("hello")`
- Added support for `import`[3.3.8] renaming with `as`
- Added a `duration`[3.5.12] type
- Added support for CBOR[3.13.2] encoding and decoding
- Added encoding and decoding functions from and to bytes for data formats: `json.decode`, `json.encode`[3.13.4/4.1], and similar functions for other formats
- Added `array.intersperse`[3.5.2/3.27] function
- Added `str.rev`[3.5.27/4.21] function
- Added `calc.tau` constant
- Made bytes[3.5.6] joinable and addable
- Made `array.zip`[3.5.2/3.17] function variadic
- Fixed bug with `eval`[3.5.13] when the `mode` was set to `"math"`
- Fixed bug with `ends-with`[3.5.27/4.13] function on strings
- Fixed bug with destructuring in combination with break, continue, and return
- Fixed argument types of hyperbolic functions[3.5.7/1.14], they don't allow angles anymore **(Breaking change)**
- Renamed some color methods: `rgba` becomes `to-rgba`, `cmyk` becomes `to-cmyk`, and `luma` becomes `to-luma` **(Breaking change)**

Export

- Added SVG export (thanks to @Enter-tainer[C])
- Fixed bugs with PDF font embedding
- Added support for page labels that reflect the page numbering[3.10.17/4.10] style in the PDF

Text and Layout

- Added `highlight`[3.7.1] function for highlighting text with a background color
- Added `polygon.regular`[3.11.9/3.1] function for drawing a regular polygon
- Added support for tabs in `raw`[3.7.6] elements alongside `tab-width`[3.7.6/4.7] parameter
- The layout engine now tries to prevent "runts" (final lines consisting of just a single word)
- Added Finnish translations
- Added hyphenation support for Polish
- Improved handling of consecutive smart quotes of different kinds
- Fixed vertical alignments for `number-align`[3.10.17/4.12] argument on page function **(Breaking change)**
- Fixed weak pagebreaks after counter updates
- Fixed missing text in SVG when the text font is set to "New Computer Modern"
- Fixed translations for Chinese
- Fixed crash for empty text in show rule

[A] https://github.com/astrale-sharp
[B] https://github.com/arnaudgolfouse
[C] https://github.com/Enter-tainer

- Fixed leading spaces when there's a linebreak after a number and a comma
- Fixed placement of floating elements in columns and other containers
- Fixed sizing of block containing just a single box

Math

- Added support for augmented matrices[3.8.18/2.3]
- Removed support for automatic matching of fences like `|` and `||` as there were too many false positives. You can use functions like `abs`[3.8.17/2.3] or `norm`[3.8.17/2.4] or an explicit `lr`[3.8.17/2.1] call instead. **(Breaking change)**
- Fixed spacing after number with decimal point in math
- Fixed bug with primes in subscript
- Fixed weak spacing
- Fixed crash when text within math contains a newline

Tooling and Diagnostics

- Added hints when trying to call a function stored in a dictionary without extra parentheses
- Fixed hint when referencing an equation without numbering
- Added more details to some diagnostics (e.g. when SVG decoding fails)

Command line interface

- Added `typst update` command for self-updating the CLI (thanks to @jimvdl[A])
- Added download progress indicator for packages and updates
- Added `--format` argument to explicitly specify the output format
- The CLI now respects proxy configuration through environment variables and has a new `--cert` option for setting a custom CA certificate
- Fixed crash when field wasn't present and `--one` is passed to `typst query`

Miscellaneous Improvements

- Added Page Setup Guide[4.2]
- Added `figure.caption`[3.6.8/6.1] function that can be used for simpler figure customization (**Breaking change** because `it.caption` now renders the full caption with supplement in figure show rules and manual outlines)
- Moved `caption-pos` argument to `figure.caption` function and renamed it to `position` **(Breaking change)**
- Added `separator`[3.6.8/6.1.2] argument to `figure.caption` function
- Added support for combination of and/or and before/after selectors[3.5.25]
- Packages can now specify a minimum compiler version[B] they require to work
- Fixed parser bug where method calls could be moved onto their own line for `#let` expressions in markup **(Breaking change)**
- Fixed bugs in sentence and title case conversion for bibliographies
- Fixed supplements for alphanumeric and author-title bibliography styles
- Fixed off-by-one error in APA bibliography style

Development

- Made `Span` and `FileId` more type-safe so that all error conditions must be handled by `World` implementors

[A] https://github.com/jimvdl
[B] https://github.com/typst/packages#package-format

Typst 0.7.0 (August 07, 2023)

Text and Layout

- Added support for floating figures through the `placement`[3.6.8/5.3] argument on the figure function
- Added support for arbitrary floating content through the `float`[3.10.19/4.3] argument on the place function
- Added support for loading `.sublime-syntax` files as highlighting syntaxes[3.7.6/4.5] for raw blocks
- Added support for loading `.tmTheme` files as highlighting themes[3.7.6/4.6] for raw blocks
- Added *bounds* option to `top-edge`[3.7.12/2.14] and `bottom-edge`[3.7.12/2.15] arguments of text function for tight bounding boxes
- Removed nonsensical top- and bottom-edge options, e.g. *ascender* for the bottom edge **(Breaking change)**
- Added `script`[3.7.12/2.18] argument to text function
- Added `alternative`[3.7.8/3.3] argument to smart quote function
- Added basic i18n for Japanese
- Added hyphenation support for `nb` and `nn` language codes in addition to `no`
- Fixed positioning of placed elements[3.10.19] in containers
- Fixed overflowing containers due to optimized line breaks

Export

- Greatly improved export of SVG images to PDF. Many thanks to @LaurenzV[A] for their work on this.
- Added support for the alpha channel of RGBA colors in PDF export
- Fixed a bug with PPI (pixels per inch) for PNG export

Math

- Improved layout of primes (e.g. in `$a'_1$`)
- Improved display of multi-primes (e.g. in `$a''$`)
- Improved layout of roots[3.8.20/2.1]
- Changed relations to show attachments as limits[3.8.10/3.3] by default (e.g. in `$a ->^x b$`)
- Large operators and delimiters are now always vertically centered
- Boxes[3.10.5] in equations now sit on the baseline instead of being vertically centered by default. Notably, this does not affect blocks[3.10.4] because they are not inline elements.
- Added support for weak spacing[3.10.26/4.2]
- Added support for OpenType character variants
- Added support for customizing the math class[3.8.14] of content
- Fixed spacing around `.`, `\/`, and `...`
- Fixed spacing between closing delimiters and large operators
- Fixed a bug with math font weight selection
- Symbols and Operators **(Breaking changes)**
  - Added `id`, `im`, and `tr` text operators[3.8.24]
  - Renamed `ident` to `equiv` with alias `eq.triple` and removed `ident.strict` in favor of `eq.quad`
  - Renamed `ast.sq` to `ast.square` and `integral.sq` to `integral.square`
  - Renamed `.eqq` modifier to `.equiv` (and `.neqq` to `.nequiv`) for `tilde`, `gt`, `lt`, `prec`, and `succ`
  - Added `emptyset` as alias for `nothing`
  - Added `lt.curly` and `gt.curly` as aliases for `prec` and `succ`
  - Added `aleph`, `beth`, and `gimmel` as alias for `alef`, `bet`, and `gimel`

Scripting

- Fields
  - Added `abs` and `em` field to lengths[3.10.13]
  - Added `ratio` and `length` field to relative lengths[3.10.21]
  - Added `x` and `y` field to 2d alignments[3.10.1/5.1]
  - Added `paint`, `thickness`, `cap`, `join`, `dash`, and `miter-limit` field to strokes[3.11.12]
- Accessor and utility methods

[A] https://github.com/LaurenzV

  - Added dedup[3.5.2/3.31] method to arrays
  - Added `pt`, `mm`, `cm`, and `inches` method to lengths[3.10.13]
  - Added `deg` and `rad` method to angles[3.10.3]
  - Added `kind`, `hex`, `rgba`, `cmyk`, and `luma` method to colors[3.11.3]
  - Added `axis`, `start`, `end`, and `inv` method to directions[3.10.28/3.1]
  - Added `axis` and `inv` method to alignments[3.10.1/5.1]
  - Added `inv` method to 2d alignments[3.10.1/5.1]
  - Added `start` argument to `enumerate`[3.5.2/3.16] method on arrays
- Added `color.mix`[3.11.3/4.18] function
- Added `mode` and `scope` arguments to `eval`[3.5.13] function
- Added `bytes`[3.5.6] type for holding large byte buffers
  - Added `encoding`[3.13.5/2.2] argument to read function to read a file as bytes instead of a string
  - Added `image.decode` function for decoding an image directly from a string or bytes
  - Added `bytes`[3.5.6] function for converting a string or an array of integers to bytes
  - Added `array`[3.5.2] function for converting bytes to an array of integers
  - Added support for converting bytes to a string with the `str`[3.5.27] function

Tooling and Diagnostics

- Added support for compiler warnings
- Added warning when compilation does not converge within five attempts due to intense use of introspection features
- Added warnings for empty emphasis (`__` and `**`)
- Improved error message for invalid field assignments
- Improved error message after single `#`
- Improved error message when a keyword is used where an identifier is expected
- Fixed parameter autocompletion for functions that are in modules
- Import autocompletion now only shows the latest package version until a colon is typed
- Fixed autocompletion for dictionary key containing a space
- Fixed autocompletion for `for` loops

Command line interface

- Added `typst query` subcommand to execute a query[3.12.6/3] on the command line
- The `--root` and `--font-path` arguments cannot appear in front of the command anymore **(Breaking change)**
- Local and cached packages are now stored in directories of the form `{namespace}/{name}/{version}` instead of `{namespace}/{name}-{version}` **(Breaking change)**
- Now prioritizes explicitly given fonts (via `--font-path`) over system and embedded fonts when both exist
- Fixed `typst watch` not working with some text editors
- Fixed displayed compilation time (now includes export)

Miscellaneous Improvements

- Added `bookmarked`[3.6.10/5.7] argument to heading to control whether a heading becomes part of the PDF outline
- Added `caption-pos`[3.6.8/6.1.1] argument to control the position of a figure's caption
- Added `metadata`[3.12.5] function for exposing an arbitrary value to the introspection system
- Fixed that a `state`[3.12.7] was identified by the pair `(key, init)` instead of just its `key`
- Improved indent logic of enumerations[3.6.12]. Instead of requiring at least as much indent as the end of the marker, they now require only one more space indent than the start of the marker. As a result, even long markers like `12.` work with just 2 spaces of indent.
- Fixed bug with indent logic of `raw`[3.7.6] blocks
- Fixed a parsing bug with dictionaries

Development

- Extracted parser and syntax tree into `typst-syntax` crate

– The `World::today` implementation of Typst dependents may need fixing if they have the same bug[A] that the CLI world had

[A] https://github.com/typst/typst/issues/1842

Typst 0.6.0 (June 30, 2023)

Package Management

- Typst now has built-in package management[3.3.9]
- You can import published[A] community packages or create and use system-local[B] ones
- Published packages are also supported in the web app

Math

- Added support for optical size variants of glyphs in math mode
- Added argument to enable `limits`[3.8.10/3.3] conditionally depending on whether the equation is set in `display`[3.8.21/1.1] or `inline`[3.8.21/1.2] style
- Added `gt.eq.slant` and `lt.eq.slant` symbols
- Increased precedence of factorials in math mode (`$1/n!$` works correctly now)
- Improved underlines[3.8.25/1.1] and overlines[3.8.25/1.2] in math mode
- Fixed usage of `limits`[3.8.10/3.3] function in show rules
- Fixed bugs with line breaks in equations

Text and Layout

- Added support for alternating page margins[3.10.17/4.5] with the `inside` and `outside` keys
- Added support for specifying the page `binding`[3.10.17/4.7]
- Added `to`[3.10.18/2.2] argument to pagebreak function to skip to the next even or odd page
- Added basic i18n for a few more languages (TR, SQ, TL)
- Fixed bug with missing table row at page break
- Fixed bug with underlines[3.7.13]
- Fixed bug superfluous table lines
- Fixed smart quotes after line breaks
- Fixed a crash related to text layout

Command line interface

- **Breaking change:** Added requirement for `--root`/`TYPST_ROOT` directory to contain the input file because it designates the *project* root. Existing setups that use `TYPST_ROOT` to emulate package management should switch to local packages[C]
- **Breaking change:** Now denies file access outside of the project root
- Added support for local packages and on-demand package download
- Now watches all relevant files, within the root and all packages
- Now displays compilation time

Miscellaneous Improvements

- Added `outline.entry`[3.6.14/5.1] to customize outline entries with show rules
- Added some hints for error messages
- Added some missing syntaxes for `raw`[3.7.6] highlighting
- Improved rendering of rotated images in PNG export and web app
- Made footnotes[3.6.9] reusable and referenceable
- Fixed bug with citations and bibliographies in `locate`[3.12.3]
- Fixed inconsistent tense in documentation

Development

- Added contribution guide[D]

[A] https://typst.app/universe
[B] https://github.com/typst/packages#local-packages
[C] https://github.com/typst/packages#local-packages
[D] https://github.com/typst/typst/blob/main/CONTRIBUTING.md

– Reworked `World` interface to accommodate for package management and make it a bit simpler to implement *(Breaking change for implementors)*

Typst 0.5.0 (June 09, 2023)

Text and Layout

- Added `raw`[3.7.6] syntax highlighting for many more languages
- Added support for Korean numbering[3.6.13]
- Added basic i18n for a few more languages (NL, SV, DA)
- Improved line breaking for East Asian languages
- Expanded functionality of outline `indent`[3.6.14/4.4] property
- Fixed footnotes in columns
- Fixed page breaking bugs with footnotes[3.6.9]
- Fixed bug with handling of footnotes in lists, tables, and figures
- Fixed a bug with CJK punctuation adjustment
- Fixed a crash with rounded rectangles
- Fixed alignment of `line`[3.11.8] elements

Math

- **Breaking change:** The syntax rules for mathematical attachments[3.8.10/3.1] were improved: `$f^abs(3)$` now parses as `$f^(abs(3))$` instead of `$(f^abs)(3)$`. To disambiguate, add a space: `$f^zeta (3)$`.
- Added forced size[3.8.21] commands for math (e.g., `display`[3.8.21/1.1])
- Added `supplement`[3.8.15/3.4] parameter to `equation`[3.8.15], used by references[3.6.18]
- New symbols[3.9.2]: `bullet`, `xor`, `slash.big`, `sigma.alt`, `tack.r.not`, `tack.r.short`, `tack.r.double.not`
- Fixed a bug with symbols in matrices
- Fixed a crash in the `attach`[3.8.10/3.1] function

Scripting

- Added new `datetime`[3.5.9] type and `datetime.today`[3.5.9/5.1] to retrieve the current date
- Added `str.from-unicode`[3.5.27/4.9] and `str.to-unicode`[3.5.27/4.8] functions
- Added `fields`[3.5.8/3.4] method on content
- Added `base` parameter to `str`[3.5.27] function
- Added `calc.exp`[3.5.7/1.3] and `calc.ln`[3.5.7/1.20]
- Improved accuracy of `calc.pow`[3.5.7/1.2] and `calc.log`[3.5.7/1.19] for specific bases
- Fixed removal[3.5.11/3.4] order for dictionary
- Fixed `.at(default: ..)` for strings[3.5.27/4.4] and content[3.5.8/3.3]
- Fixed field access on styled elements
- Removed deprecated `calc.mod` function

Command line interface

- Added PNG export via `typst compile source.typ output-{n}.png`. The output path must contain `{n}` if the document has multiple pages.
- Added `--diagnostic-format=short` for Unix-style short diagnostics
- Doesn't emit color codes anymore if stderr isn't a TTY
- Now sets the correct exit when invoked with a nonexistent file
- Now ignores UTF-8 BOM in Typst files

Miscellaneous Improvements

- Improved errors for mismatched delimiters
- Improved error message for failed length comparisons
- Fixed a bug with images not showing up in Apple Preview
- Fixed multiple bugs with the PDF outline
- Fixed citations and other searchable elements in `hide`[3.10.11]
- Fixed bugs with reference supplements[3.6.18/4.2]
- Fixed Nix flake

Typst 0.4.0 (May 20, 2023)

Footnotes

- Implemented support for footnotes
- The `footnote`[3.6.9] function inserts a footnote
- The `footnote.entry`[3.6.9/4.1] function can be used to customize the footnote listing
- The `"chicago-notes"` citation style[3.6.4/3.4] is now available

Documentation

- Added a Guide for LaTeX Users[4.1]
- Now shows default values for optional arguments
- Added richer outlines in "On this Page"
- Added initial support for search keywords: "Table of Contents" will now find the outline[3.6.14] function. Suggestions for more keywords are welcome!
- Fixed issue with search result ranking
- Fixed many more small issues

Math

- **Breaking change**: Alignment points (&) in equations now alternate between left and right alignment
- Added support for writing roots with Unicode: For example, `$root(x+y)$` can now also be written as `$√(x+y)$`
- Fixed uneven vertical `attachment`[3.8.10/3.1] alignment
- Fixed spacing on decorated elements (e.g., spacing around a canceled[3.8.12] operator)
- Fixed styling for stretchable symbols
- Added `tack.r.double`, `tack.l.double`, `dotless.i` and `dotless.j` symbols[3.9.2]
- Fixed show rules on symbols (e.g. `show sym.tack: set text(blue)`)
- Fixed missing rename from `ast.op` to `ast` that should have been in the previous release

Scripting

- Added function scopes: A function can now hold related definitions in its own scope, similar to a module. The new `assert.eq`[3.5.3/3.1] function, for instance, is part of the `assert`[3.5.3] function's scope. Note that function scopes are currently only available for built-in functions.
- Added `assert.eq`[3.5.3/3.1] and `assert.ne`[3.5.3/3.2] functions for simpler equality and inequality assertions with more helpful error messages
- Exposed list[3.6.3/4.1], enum[3.6.12/4.1], and term list[3.6.21/4.1] items in their respective functions' scope
- The `at` methods on strings[3.5.27/4.4], arrays[3.5.2/3.4], dictionaries[3.5.11/3.2], and content[3.5.8/3.3] now support specifying a default value
- Added support for passing a function to `replace`[3.5.27/4.18] that is called with each match.
- Fixed replacement[3.5.27/4.18] strings: They are now inserted completely verbatim instead of supporting the previous (unintended) magic dollar syntax for capture groups
- Fixed bug with trailing placeholders in destructuring patterns
- Fixed bug with underscore in parameter destructuring
- Fixed crash with nested patterns and when hovering over an invalid pattern
- Better error messages when casting to an integer[3.5.16] or float[3.5.14] fails

Text and Layout

- Implemented sophisticated CJK punctuation adjustment
- Disabled overhang[3.7.12/2.13] for CJK punctuation
- Added basic translations for Traditional Chinese
- Fixed alignment[3.7.6/4.4] of text inside raw blocks (centering a raw block, e.g. through a figure, will now keep the text itself left-aligned)
- Added support for passing a array instead of a function to configure table cell alignment[3.6.20/3.7] and fill[3.6.20/3.8] per column

- Fixed automatic figure `kind`[3.6.8/5.6] detection
- Made alignment of enum numbers[3.6.12/3.9] configurable, defaulting to `end`
- Figures can now be made breakable with a show-set rule for blocks in figure
- Initial fix for smart quotes in RTL languages

Export

- Fixed ligatures in PDF export: They are now copyable and searchable
- Exported PDFs now embed ICC profiles for images that have them
- Fixed export of strokes with zero thickness

Web app

- Projects can now contain folders
- Added upload by drag-and-drop into the file panel
- Files from the file panel can now be dragged into the editor to insert them into a Typst file
- You can now copy-paste images and other files from your computer directly into the editor
- Added a button to resend confirmation email
- Added an option to invert preview colors in dark mode
- Added tips to the loading screen and the Help menu. Feel free to propose more!
- Added syntax highlighting for YAML files
- Allowed middle mouse button click on many buttons to navigate into a new tab
- Allowed more project names
- Fixed overridden Vim mode keybindings
- Fixed many bugs regarding file upload and more

Miscellaneous Improvements

- Improved performance of counters, state, and queries
- Improved incremental parsing for more efficient recompilations
- Added support for `.yaml` extension in addition to `.yml` for bibliographies
- The CLI now emits escape codes only if the output is a TTY
- For users of the `typst` crate: The `Document` is now `Sync` again and the `World` doesn't have to be `'static` anymore

Typst 0.3.0 (April 26, 2023)

Breaking changes

- Renamed a few symbols: What was previous `dot.op` is now just `dot` and the basic dot is `dot.basic`. The same applies to `ast` and `tilde`.
- Renamed `mod` to `rem`[3.5.7/1.37] to more accurately reflect the behavior. It will remain available as `mod` until the next update as a grace period.
- A lone underscore is not a valid identifier anymore, it can now only be used in patterns
- Removed `before` and `after` arguments from `query`[3.12.6]. This is now handled through flexible selectors[3.5.25] combinator methods
- Added support for attachments[3.8.10/3.1] (sub-, superscripts) that precede the base symbol. The `top` and `bottom` arguments have been renamed to `t` and `b`.

New features

- Added support for more complex strokes[3.11.12] (configurable caps, joins, and dash patterns)
- Added `cancel`[3.8.12] function for equations
- Added support for destructuring[3.3.3] in argument lists and assignments
- Added `alt`[3.11.7/2.5] text argument to image function
- Added `toml`[3.13.6] function for loading data from a TOML file
- Added `zip`[3.5.2/3.17], `sum`[3.5.2/3.19], and `product`[3.5.2/3.20] methods for arrays
- Added `fact`, `perm`, `binom`, `gcd`, `lcm`, `atan2`, `quo`, `trunc`, and `fract` calculation[3.5.7] functions

Improvements

- Text in SVGs now displays properly
- Typst now generates a PDF heading outline
- References[3.6.18] now provides the referenced element as a field in show rules
- Refined linebreak algorithm for better Chinese justification
- Locations are now a valid kind of selector
- Added a few symbols for algebra
- Added Spanish smart quote support
- Added `selector`[3.5.25] function to turn a selector-like value into a selector on which combinator methods can be called
- Improved some error messages
- The outline and bibliography headings can now be styled with show-set rules
- Operations on numbers now produce an error instead of overflowing

Bug fixes

- Fixed wrong linebreak before punctuation that follows inline equations, citations, and other elements
- Fixed a bug with argument sinks[3.5.1]
- Fixed strokes with thickness zero
- Fixed hiding and show rules in math
- Fixed alignment in matrices
- Fixed some alignment bugs in equations
- Fixed grid cell alignment
- Fixed alignment of list marker and enum markers in presence of global alignment settings
- Fixed path closing
- Fixed compiler crash with figure references
- A single trailing line breaks is now ignored in math, just like in text

Command line interface

- Font path and compilation root can now be set with the environment variables `TYPST_FONT_PATHS` and `TYPST_ROOT`
- The output of `typst fonts` now includes the embedded fonts

Development

- Added instrumentation for debugging and optimization
- Added `--update` flag and `UPDATE_EXPECT` environment variable to update reference images for tests
- You can now run a specific subtest with `--subtest`
- Tests now run on multiple threads

Typst 0.2.0 (April 11, 2023)

Breaking changes

- Removed support for iterating over index and value in for loops[3.3.5]. This is now handled via unpacking and enumerating. Same goes for the `map`[3.5.2/3.15] method.
- Dictionaries[3.5.11] now iterate in insertion order instead of alphabetical order.

New features

- Added unpacking syntax[3.3.3] for let bindings, which allows things like `let (1, 2) = array`
- Added `enumerate`[3.5.2/3.16] method
- Added `path` function for drawing Bézier paths
- Added `layout`[3.10.12] function to access the size of the surrounding page or container
- Added `key` parameter to `sorted`[3.5.2/3.30] method

Command line interface

- Fixed `--open` flag blocking the program
- New Computer Modern font is now embedded into the binary
- Shell completions and man pages can now be generated by setting the `GEN_ARTIFACTS` environment variable to a target directory and then building Typst

Miscellaneous improvements

- Fixed page numbering in outline
- Added basic i18n for a few more languages (AR, NB, CS, NN, PL, SL, ES, UA, VI)
- Added a few numbering patterns (Ihora, Chinese)
- Added `sinc` operator[3.8.24]
- Fixed bug where math could not be hidden with `hide`[3.10.11]
- Fixed sizing issues with box, block, and shapes
- Fixed some translations
- Fixed inversion of "R" in `cal`[3.8.26/1.6] and `frak`[3.8.26/1.3] styles
- Fixed some styling issues in math
- Fixed supplements of references to headings
- Fixed syntax highlighting of identifiers in certain scenarios
- Ratios[3.10.20] can now be multiplied with more types and be converted to floats[3.5.14] with the `float`[3.5.14] function

Typst 0.1.0 (April 04, 2023)

Breaking changes

- When using the CLI, you now have to use subcommands:
  - `typst compile file.typ` or `typst c file.typ` to create a PDF
  - `typst watch file.typ` or `typst w file.typ` to compile and watch
  - `typst fonts` to list all fonts
- Manual counters now start at zero. Read the "How to step" section here[3.12.1] for more details
- The bibliography styles[3.6.2/4.4] `"author-date"` and `"author-title"` were renamed to `"chicago-author-date"` and `"chicago-author-title"`

Figure improvements

- Figures now automatically detect their content and adapt their behavior. Figures containing tables, for instance, are automatically prefixed with "Table X" and have a separate counter
- The figure's supplement (e.g. "Figure" or "Table") can now be customized
- In addition, figures can now be completely customized because the show rule gives access to the automatically resolved kind, supplement, and counter

Bibliography improvements

- The `bibliography`[3.6.2] now also accepts multiple bibliography paths (as an array)
- Parsing of BibLaTeX files is now more permissive (accepts non-numeric edition, pages, volumes, dates, and Jabref-style comments; fixed abbreviation parsing)
- Labels and references can now include `:` and `.` except at the end
- Fixed APA bibliography ordering

Drawing additions

- Added `polygon`[3.11.9] function for drawing polygons
- Added support for clipping in boxes[3.10.5/2.9] and blocks[3.10.4/2.12]

Command line interface

- Now returns with non-zero status code if there is an error
- Now watches the root directory instead of the current one
- Now puts the PDF file next to input file by default
- Now accepts more kinds of input files (e.g. `/dev/stdin`)
- Added `--open` flag to directly open the PDF

Miscellaneous improvements

- Added `yaml`[3.13.8] function to load data from YAML files
- Added basic i18n for a few more languages (IT, RU, ZH, FR, PT)
- Added numbering support for Hebrew
- Added support for integers[3.5.16] with base 2, 8, and 16
- Added symbols for double bracket and laplace operator
- The `link`[3.6.11] function now accepts labels[3.5.17]
- The link syntax now allows more characters
- Improved justification of Japanese and Chinese text
- Calculation functions behave more consistently w.r.t to non-real results
- Replaced deprecated angle brackets
- Reduced maximum function call depth from 256 to 64
- Fixed `first-line-indent`[3.6.15/4.6] being not applied when a paragraph starts with styled text
- Fixed extraneous spacing in unary operators in equations
- Fixed block spacing, e.g. in `block(above: 1cm, below: 1cm, ..)`
- Fixed styling of text operators in math

- Fixed invalid parsing of language tag in raw block with a single backtick
- Fixed bugs with displaying counters and state
- Fixed crash related to page counter
- Fixed crash when `symbol`[3.5.28] function was called without arguments
- Fixed crash in bibliography generation
- Fixed access to label of certain content elements
- Fixed line number in error message for CSV parsing
- Fixed invalid autocompletion after certain markup elements

Changes in early, unversioned Typst

March 28, 2023

- **Breaking changes:**
  - Enumerations now require a space after their marker, that is, `1.ok` must now be written as `1. ok`
  - Changed default style for term lists[3.6.21]: Does not include a colon anymore and has a bit more indent
- Command line interface
  - Added `--font-path` argument for CLI
  - Embedded default fonts in CLI binary
  - Fixed build of CLI if `git` is not installed
- Miscellaneous improvements
  - Added support for disabling matrix[3.8.18] and vector[3.8.27] delimiters. Generally with `#set math.mat(delim: none)` or one-off with `$mat(delim: #none, 1, 2; 3, 4)$`.
  - Added `separator`[3.6.21/3.2] argument to term lists
  - Added `round`[3.8.17/2.7] function for equations
  - Numberings now allow zeros. To reset a counter, you can write `#counter(..).update(0)`
  - Added documentation for `page()` and `position()` methods on `location`[3.12.4] type
  - Added symbols for double, triple, and quadruple dot accent
  - Added smart quotes for Norwegian Bokmål
  - Added Nix flake
  - Fixed bibliography ordering in IEEE style
  - Fixed parsing of decimals in math: `$1.2/3.4$`
  - Fixed parsing of unbalanced delimiters in fractions: `$1/(2 (x)$`
  - Fixed unexpected parsing of numbers as enumerations, e.g. in `1.2`
  - Fixed combination of page fill and header
  - Fixed compiler crash if `repeat`[3.10.22] is used in page with automatic width
  - Fixed matrices[3.8.18] with explicit delimiter
  - Fixed `indent`[3.6.21/3.3] property of term lists
  - Numerous documentation fixes
  - Links in bibliographies are now affected by link styling
  - Fixed hovering over comments in web app

March 21, 2023

- Reference and bibliography management
  - Bibliographies[3.6.2] and citations[3.6.4] (currently supported styles are APA, Chicago Author Date, IEEE, and MLA)
  - You can now reference[3.6.18] sections, figures, formulas, and works from the bibliography with `@label`
  - You can make an element referenceable with a label:
    - `= Introduction <intro>`
    - `$ A = pi r^2 $ <area>`
- Introspection system for interactions between different parts of the document
  - `counter`[3.12.1] function
    - Access and modify counters for pages, headings, figures, and equations
    - Define and use your own custom counters
    - Time travel: Find out what the counter value was or will be at some other point in the document (e.g. when you're building a list of figures, you can determine the value of the figure counter at any given figure).
    - Counters count in layout order and not in code order
  - `state`[3.12.7] function
    - Manage arbitrary state across your document
    - Time travel: Find out the value of your state at any position in the document
    - State is modified in layout order and not in code order
  - `query`[3.12.6] function

  - Find all occurrences of an element or a label, either in the whole document or before/after some location
  - Link to elements, find out their position on the pages and access their fields
  - Example use cases: Custom list of figures or page header with current chapter title
- `locate`[3.12.3] function
  - Determines the location of itself in the final layout
  - Can be accessed to get the `page` and `x`, `y` coordinates
  - Can be used with counters and state to find out their values at that location
  - Can be used with queries to find elements before or after its location

- New `measure`[3.10.14] function
  - Measure the layouted size of elements
  - To be used in combination with the new `style` function that lets you generate different content based on the style context something is inserted into (because that affects the measured size of content)

- Exposed content representation
  - Content is not opaque anymore
  - Content can be compared for equality
  - The tree of content elements can be traversed with code
  - Can be observed in hover tooltips or with `repr`[3.5.24]
  - New methods[3.5.8] on content: `func`, `has`, `at`, and `location`
  - All optional fields on elements are now settable
  - More uniform field names (`heading.title` becomes `heading.body`, `list.items` becomes `list.children`, and a few more changes)

- Further improvements
  - Added `figure`[3.6.8] function
  - Added `numbering`[3.8.15/3.2] parameter on equation function
  - Added `numbering`[3.10.17/4.10] and `number-align`[3.10.17/4.12] parameters on page function
  - The page function's `header`[3.10.17/4.13] and `footer`[3.10.17/4.15] parameters do not take functions anymore. If you want to customize them based on the page number, use the new `numbering`[3.10.17/4.10] parameter or `counter`[3.12.1] function instead.
  - Added `footer-descent`[3.10.17/4.16] and `header-ascent`[3.10.17/4.14] parameters
  - Better default alignment in header and footer
  - Fixed Arabic vowel placement
  - Fixed PDF font embedding issues
  - Renamed `math.formula` to `math.equation`[3.8.15]
  - Font family must be a named argument now: `#set text(font: "..")`
  - Added support for hanging indent[3.6.15/4.7]
  - Renamed paragraph `indent` to `first-line-indent`[3.6.15/4.6]
  - More accurate logarithm[3.5.7/1.19] when base is `2` or `10`
  - Improved some error messages
  - Fixed layout of `terms`[3.6.21] list

- Web app improvements
  - Added template gallery
  - Added buttons to insert headings, equations, raw blocks, and references
  - Jump to the source of something by clicking on it in the preview panel (works for text, equations, images, and more)
  - You can now upload your own fonts and use them in your project
  - Hover debugging and autocompletion now takes multiple files into account and works in show rules
  - Hover tooltips now automatically collapse multiple consecutive equal values
  - The preview now automatically scrolls to the right place when you type
  - Links are now clickable in the preview area
  - Toolbar, preview, and editor can now all be hidden
  - Added autocompletion for raw block language tags
  - Added autocompletion in SVG files
  - New back button instead of four-dots button
  - Lots of bug fixes

February 25, 2023

- Font changes
  - New default font: Linux Libertine
  - New default font for raw blocks: DejaVu Sans Mono
  - New default font for math: Book weight of New Computer Modern Math
  - Lots of new math fonts available
  - Removed Latin Modern fonts in favor of New Computer Modern family
  - Removed unnecessary smallcaps fonts which are already accessible through the corresponding main font and the `smallcaps`[3.7.7] function
- Improved default spacing for headings
- Added `panic`[3.5.20] function
- Added `clusters`[3.5.27/4.6] and `codepoints`[3.5.27/4.7] methods for strings
- Support for multiple authors in `set document`[3.6.6/3.4]
- Fixed crash when string is accessed at a position that is not a char boundary
- Fixed semicolon parsing in `#var ;`
- Fixed incremental parsing when inserting backslash at end of `#"abc"`
- Fixed names of a few font families (including Noto Sans Symbols and New Computer Modern families)
- Fixed autocompletion for font families
- Improved incremental compilation for user-defined functions

February 15, 2023

- Box[3.10.5] and block[3.10.4] have gained `fill`, `stroke`, `radius`, and `inset` properties
- Blocks may now be explicitly sized, fixed-height blocks can still break across pages
- Blocks can now be configured to be `breakable`[3.10.4/2.3] or not
- Numbering style[3.6.12/3.2] can now be configured for nested enums
- Markers[3.6.3/3.2] can now be configured for nested lists
- The `eval`[3.5.13] function now expects code instead of markup and returns an arbitrary value. Markup can still be evaluated by surrounding the string with brackets.
- PDFs generated by Typst now contain XMP metadata
- Link boxes are now disabled in PDF output
- Tables don't produce small empty cells before a pagebreak anymore
- Fixed raw block highlighting bug

February 12, 2023

- Shapes, images, and transformations (move/rotate/scale/repeat) are now block-level. To integrate them into a paragraph, use a `box`[3.10.5] as with other elements.
- A colon is now required in an "everything" show rule: Write `show: it => ..` instead of `show it => ...` This prevents intermediate states that ruin your whole document.
- Non-math content like a shape or table in a math formula is now centered vertically
- Support for widow and orphan prevention within containers
- Support for RTL[3.7.12/2.19] in lists, grids, and tables
- Support for explicit `auto` sizing for boxes and shapes
- Support for fractional (i.e. `1fr`) widths for boxes
- Fixed bug where columns jump to next page
- Fixed bug where list items have no leading
- Fixed relative sizing in lists, squares and grid auto columns
- Fixed relative displacement in `place`[3.10.19] function
- Fixed that lines don't have a size
- Fixed bug where `set document(..)` complains about being after content
- Fixed parsing of `not in` operation
- Fixed hover tooltips in math
- Fixed bug where a heading show rule may not contain a pagebreak when an outline is present
- Added `baseline`[3.10.5/2.3] property on `box`[3.10.5]
- Added `tg`[3.8.24] and `ctg`[3.8.24] operators in math
- Added delimiter setting for `cases`[3.8.13] function

– Parentheses are now included when accepting a function autocompletion

February 2, 2023

– Merged text and math symbols, renamed a few symbols (including `infty` to `infinity` with the alias `oo`)
– Fixed missing italic mappings
– Math italics correction is now applied properly
– Parentheses now scale in $zeta(x/2)$
– Fixed placement of large root index
– Fixed spacing in $abs(-x)$
– Fixed inconsistency between text and identifiers in math
– Accents are now ignored when positioning superscripts
– Fixed vertical alignment in matrices
– Fixed `text` set rule in `raw` show rule
– Heading and list markers now parse consistently
– Allow arbitrary math directly in content

January 30, 2023

Go to the announcement blog post.[A]
– New expression syntax in markup/math
  – Blocks cannot be directly embedded in markup anymore
  – Like other expressions, they now require a leading hash
  – More expressions available with hash, including literals (`#"string"`) as well as field access and method call without space: `#emoji.face`
– New import syntax
  – `#import "module.typ"` creates binding named `module`
  – `#import "module.typ": a, b` or `#import "module.typ": *` to import items
  – `#import emoji: face, turtle` to import from already bound module
– New symbol handling
  – Removed symbol notation
  – Symbols are now in modules: `sym`, `emoji`, and `math`
  – Math module also reexports all of `sym`
  – Modified through field access, still order-independent
  – Unknown modifiers are not allowed anymore
  – Support for custom symbol definitions with `symbol` function
  – Symbols now listed in documentation
– New `math` module
  – Contains all math-related functions
  – Variables and function calls directly in math (without hash) access this module instead of the global scope, but can also access local variables
  – Can be explicitly used in code, e.g. `#set math.vec(delim: "[")`
– Delimiter matching in math
  – Any opening delimiters matches any closing one
  – When matched, they automatically scale
  – To prevent scaling, escape them
  – To forcibly match two delimiters, use `lr` function
  – Line breaks may occur between matched delimiters
  – Delimiters may also be unbalanced
  – You can also use the `lr` function to scale the brackets (or just one bracket) to a specific size manually
– Multi-line math with alignment
  – The \ character inserts a line break
  – The & character defines an alignment point
  – Alignment points also work for underbraces, vectors, cases, and matrices
  – Multiple alignment points are supported
– More capable math function calls

[A] https://typst.app/blog/2023/january-update

  - Function calls directly in math can now take code expressions with hash
  - They can now also take named arguments
  - Within math function calls, semicolons turn preceding arguments to arrays to support matrices: `$mat(1, 2; 3, 4)$`
- Arbitrary content in math
  - Text, images, and other arbitrary content can now be embedded in math
  - Math now also supports font fallback to support e.g. CJK and emoji
- More math features
  - New text operators: `op` function, `lim`, `max`, etc.
  - New matrix function: `mat`
  - New n-ary roots with `root` function: `$root(3, x)$`
  - New under- and overbraces, -brackets, and -lines
  - New `abs` and `norm` functions
  - New shorthands: `[|`, `|]`, and `||`
  - New `attach` function, overridable attachments with `script` and `limit`
  - Manual spacing in math, with `h`, `thin`, `med`, `thick` and `quad`
  - Symbols and other content may now be used like a function, e.g. `$zeta(x)$`
  - Added Fira Math font, removed Noto Sans Math font
  - Support for alternative math fonts through `#show math.formula: set text("Fira Math")`
- More library improvements
  - New `calc` module, `abs`, `min`, `max`, `even`, `odd` and `mod` moved there
  - New `message` argument on `assert` function
  - The `pairs` method on dictionaries now returns an array of length-2 arrays instead of taking a closure
  - The method call `dict.at("key")` now always fails if `"key"` doesn't exist Previously, it was allowed in assignments. Alternatives are `dict.key = x` and `dict.insert("key", x)`.
- Smarter editor functionality
  - Autocompletion for local variables
  - Autocompletion for methods available on a value
  - Autocompletion for symbols and modules
  - Autocompletion for imports
  - Hover over an identifier to see its value(s)
- Further editor improvements
  - New Font menu with previews
  - Single projects may now be shared with share links
  - New dashboard experience if projects are shared with you
  - Keyboard Shortcuts are now listed in the menus and there are more of them
  - New Offline indicator
  - Tooltips for all buttons
  - Improved account protection
  - Moved Status indicator into the error list button
- Further fixes
  - Multiple bug fixes for incremental parser
  - Fixed closure parameter capturing
  - Fixed tons of math bugs
  - Bugfixes for performance, file management, editing reliability
  - Added redirection to the page originally navigated to after signin