

Typst Documentation

Table of Contents

1. Overview.....	8
2. Tutorial	8
2.1 Writing in Typst.....	10
2.2 Formatting.....	22
2.3 Advanced Styling	32
2.4 Making a Template.....	49
3. Reference	61
3.1 Syntax.....	62
3.2 Styling	70
3.3 Scripting	75
3.4 Context	87
3.5 Foundations	96
3.5.1 Arguments	97
3.5.2 Array	99
3.5.3 Assert.....	113
3.5.4 Auto	116
3.5.5 Boolean.....	116
3.5.6 Bytes	116
3.5.7 Calculation	119
3.5.8 Content	141
3.5.9 Datetime	144
3.5.10 Decimal.....	152
3.5.11 Dictionary.....	155
3.5.12 Duration	159
3.5.13 Evaluate	161

3.5.14 Float.....	163
3.5.15 Function	168
3.5.16 Integer	172
3.5.17 Label	181
3.5.18 Module	182
3.5.19 None.....	183
3.5.20 Panic.....	183
3.5.21 Plugin	184
3.5.22 Regex	188
3.5.23 Representation.....	189
3.5.24 Selector.....	190
3.5.25 String.....	194
3.5.26 Style	203
3.5.27 System	203
3.5.28 Type	204
3.5.29 Version.....	206
3.6 Model.....	208
3.6.1 Bibliography	209
3.6.2 Bullet List	217
3.6.3 Cite	222
3.6.4 Document	231
3.6.5 Emphasis	233
3.6.6 Figure.....	234
3.6.7 Footnote	247
3.6.8 Heading	253
3.6.9 Link.....	259
3.6.10 Numbered List	261

3.6.11 Numbering	261
3.6.12 Outline	264
3.6.13 Paragraph	274
3.6.14 Paragraph Break	284
3.6.15 Quote	285
3.6.16 Reference	290
3.6.17 Strong Emphasis	295
3.6.18 Table	296
3.6.19 Term List	316
3.7 Text	320
3.7.1 Highlight	321
3.7.2 Line Break	325
3.7.3 Lorem	326
3.7.4 Lowercase	327
3.7.5 Overline	328
3.7.6 Raw Text / Code	331
3.7.7 Small Capitals	339
3.7.8 Smartquote	341
3.7.9 Strikethrough	344
3.7.10 Subscript	346
3.7.11 Superscript	348
3.7.12 Text	349
3.7.13 Underline	373
3.7.14 Uppercase	376
3.8 Math	376
3.8.1 Accent	381
3.8.2 Attach	383

3.8.3 Binomial	387
3.8.4 Cancel	388
3.8.5 Cases	391
3.8.6 Class	394
3.8.7 Equation	395
3.8.8 Fraction	399
3.8.9 Left/Right	400
3.8.10 Matrix	404
3.8.11 Primes	409
3.8.12 Roots	410
3.8.13 Sizes	411
3.8.14 Stretch	414
3.8.15 Styles	415
3.8.16 Text Operator	415
3.8.17 Under/Over	416
3.8.18 Variants	423
3.8.19 Vector	426
3.9 Symbols	428
3.9.1 General Symbols	429
3.9.2 Emoji Symbols	446
3.9.3 Symbol Type	468
3.10 Layout	470
3.10.1 Align	470
3.10.2 Alignment	473
3.10.3 Angle	476
3.10.4 Block	476
3.10.5 Box	485

3.10.6 Column Break.....	489
3.10.7 Columns.....	490
3.10.8 Direction.....	492
3.10.9 Fraction.....	494
3.10.10 Grid.....	495
3.10.11 Hide.....	515
3.10.12 Layout.....	516
3.10.13 Length	518
3.10.14 Measure	521
3.10.15 Move	523
3.10.16 Padding.....	524
3.10.17 Page	527
3.10.18 Page Break.....	541
3.10.19 Place.....	543
3.10.20 Ratio.....	550
3.10.21 Relative Length.....	551
3.10.22 Repeat	551
3.10.23 Rotate	553
3.10.24 Scale.....	555
3.10.25 Skew.....	557
3.10.26 Spacing (H)	560
3.10.27 Spacing (V).....	562
3.10.28 Stack.....	564
3.11 Visualize	566
3.11.1 Circle.....	566
3.11.2 Color.....	569
3.11.3 Ellipse	591

3.11.4 Gradient	594
3.11.5 Image	594
3.11.6 Line	600
3.11.7 Path.....	602
3.11.8 Pattern	605
3.11.9 Polygon	609
3.11.10 Rectangle.....	609
3.11.11 Square	614
3.11.12 Stroke	617
3.12 Introspection	623
3.12.1 Counter.....	624
3.12.2 Here	634
3.12.3 Locate	636
3.12.4 Location.....	638
3.12.5 Metadata.....	639
3.12.6 Query	640
3.12.7 State	645
3.13 Data Loading.....	653
3.13.1 CBOR.....	654
3.13.2 CSV	655
3.13.3 JSON.....	657
3.13.4 Read	660
3.13.5 TOML	662
3.13.6 XML.....	663
3.13.7 YAML.....	666
4. Guides	668
4.1 Guide for LaTeX users.....	668

4.2 Page setup guide	690
4.3 Table guide	716
5. Changelog.....	763
5.1 0.12.0.....	764
5.2 0.11.1.....	781
5.3 0.11.0.....	783
5.4 0.10.0.....	793
5.5 0.9.0.....	798
5.6 0.8.0.....	806
5.7 0.7.0.....	810
5.8 0.6.0.....	816
5.9 0.5.0.....	818
5.10 0.4.0.....	820
5.11 0.3.0.....	824
5.12 0.2.0.....	827
5.13 0.1.0.....	829
5.14 Earlier.....	832
6. Roadmap	844
7. Community	852

1. Overview

Welcome to Typst's documentation! Typst is a new markup-based typesetting system for the sciences. It is designed to be an alternative both to advanced tools like LaTeX and simpler tools like Word and Google Docs. Our goal with Typst is to build a typesetting tool that is highly capable *and* a pleasure to use. This documentation is split into two parts: A beginner-friendly tutorial that introduces Typst through a practical use case and a comprehensive reference that explains all of Typst's concepts and features.

We also invite you to join the community we're building around Typst. Typst is still a very young project, so your feedback is more than valuable.

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](#).

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.

What you will learn

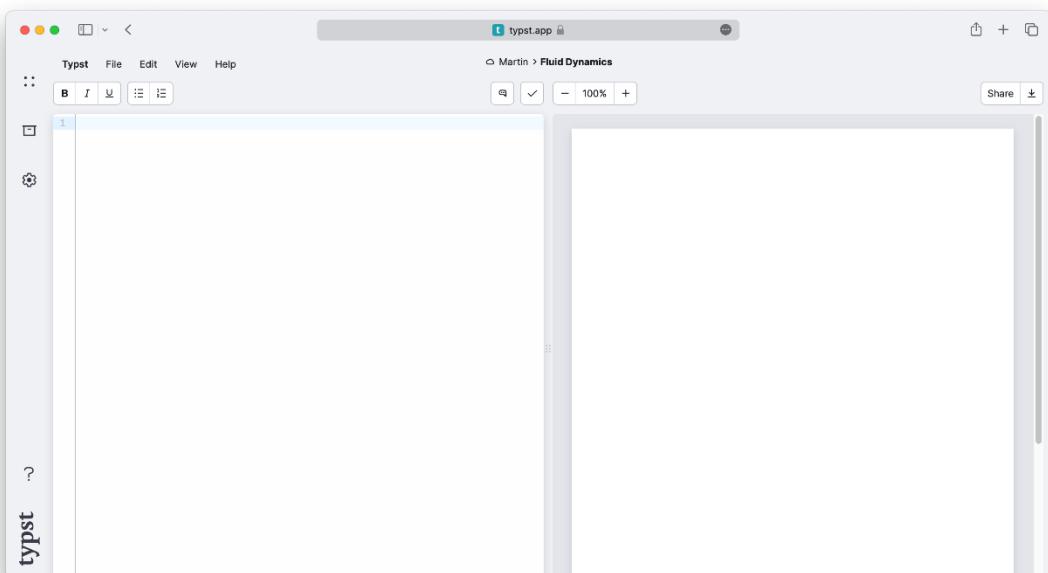
This tutorial has four chapters. Each chapter builds on the previous one. Here is what you will learn in each of them:

1. [Writing in Typst](#): Learn how to write text and insert images, equations, and other elements.
2. [Formatting](#): Learn how to adjust the formatting of your document, including font size, heading styles, and more.
3. [Advanced Styling](#): 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](#): Build a reusable template from the paper you created in the previous chapter.

We hope you'll enjoy 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

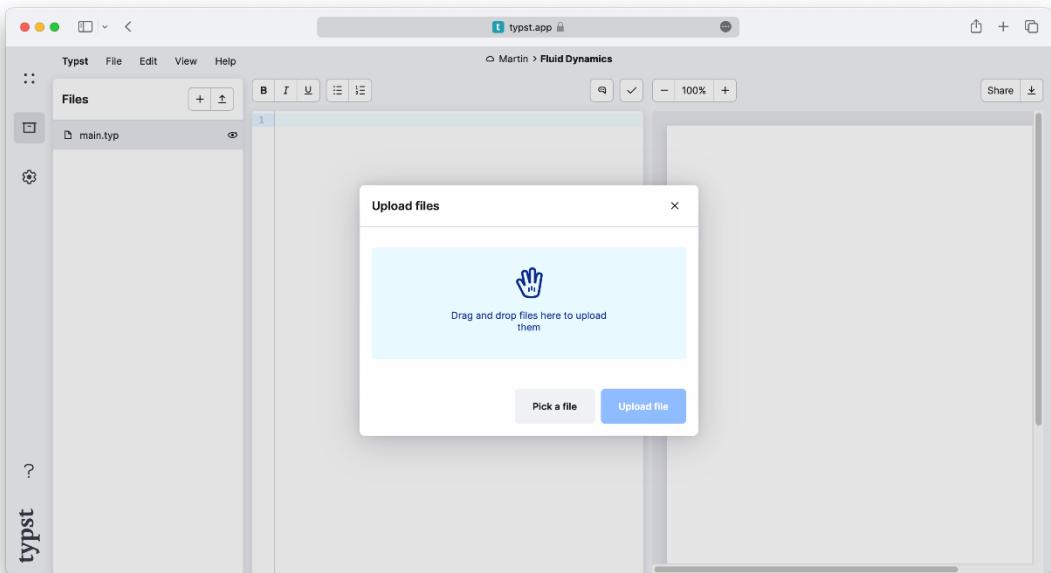
If we wanted to add a bulleted list, we would use the – character instead of the + character. We can also nest lists: For example, we can add a sub-list to the first item of the list above by indenting it.

- + The climate
 - Temperature
 - Precipitation
- + The topography
- + The geology

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

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, and SVG. 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](#) 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](#), 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](#). 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](#) 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.*

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. On the contrary, strings work wherever content is expected because text is a valid kind of content.

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](#) function. This function expects a path to a bibliography file.

Typst's native bibliography format is [Hayagriva](#), 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](#) for more details.

= 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.

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 $Q = \rho Av + C$ 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:

$$Q = \rho Av + C$$

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:

$$Q = \rho Av + \text{time offset}$$

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` 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` function is used to typeset calligraphic letters commonly used for sets. The [math section of the reference](#) 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.squiggle 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](#) for a list of all available symbols.

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](#), [shapes](#), and [code blocks](#). You can peruse the reference 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.

Set rules

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

```
#par (justify: true) [
    = 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).
]
```

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. The `par` function justifies all paragraphs 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 par(justify: true)
```

= 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).

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).

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

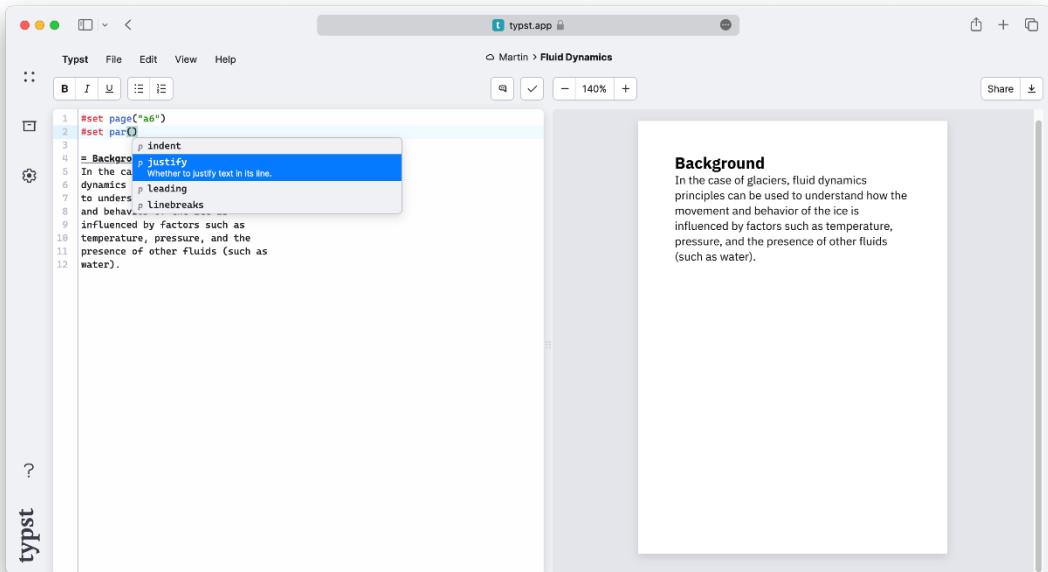
Set rules can be conceptualized as setting default values for some of the parameters of a function for all future uses of that function.

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.



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](#) to set font family, size, color, and other properties of text
- [page](#) to set the page size, margins, headers, enable columns, and footers
- [par](#) to justify paragraphs, set line spacing, and more
- [heading](#) to set the appearance of headings and enable numbering
- [document](#) 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.*
]
```

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.

Glacier displacement is influenced by a number of factors, including

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

This report will present a physical model of glacier displacement and dynamics, and will explore the influence of these factors on the movement of large bodies of ice.



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

There are a few things of note here.

First is the [page](#) 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](#), but you can also specify a custom page size. The margins are specified as a [dictionary](#). 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](#) 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](#) 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.

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](#) function.

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

= Introduction
#lorem(10)

== Background
#lorem(12)
```

== Methods

#lorem(15)

1. Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

1.1. Background

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor.

1.2. Methods

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.

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](#) for our headings:

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

= Introduction

#lorem(10)

== Background

#lorem(12)

== Methods

#lorem(15)

1 Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

1.a Background

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor.

1.b Methods

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.

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

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.

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](#) 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](#).

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.

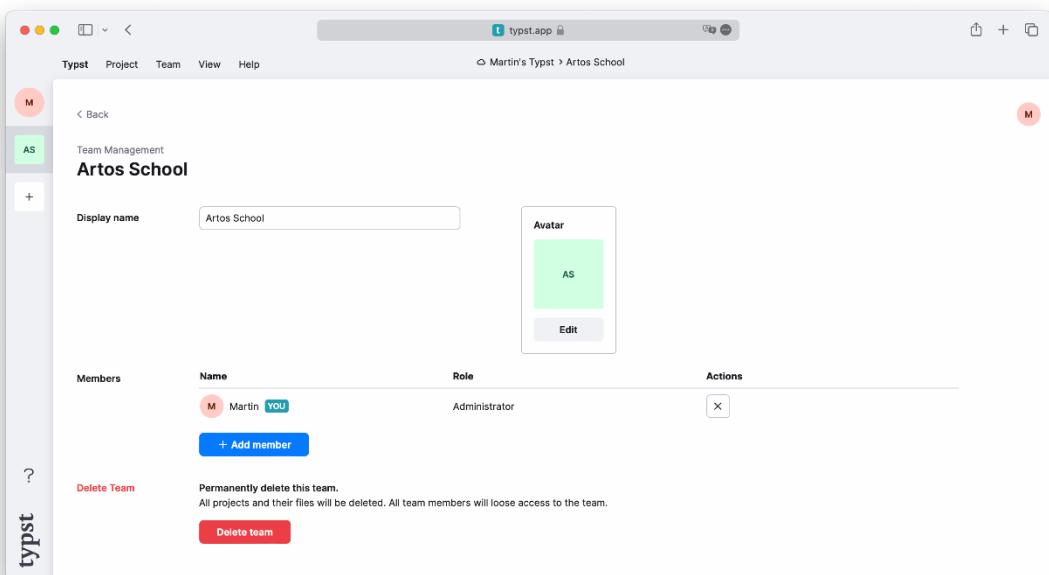
You have handed in your report. Your supervisor was so happy with it that they want to adapt it into a conference paper! In the next section, we will learn how to format your document as a paper using more advanced show rules and functions.

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](#) to find other users and try teams with them!

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.

Writing the right set rules

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

```
#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)
```

A fluid dynamic model for glacier flow

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequa 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 iridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitibus aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non quoque unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa, nollem me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum dedocere. Sol Democrito magnus videtur, quippe homini eruditio in geometriaque perfecto, huic pedalis fortasse; tantum enim esse omnino in nostris poetis aut inertissimae segnitiae est aut fastidii delicatissimi. Mihi quidem videtur, inermis ac nudus est. Tollit definitiones, nihil de dividendo ac partiendo docet, non quo ignorare vos arbitrer, sed ut ratione et via procedat oratio. Quaerimus igitur, quid sit extremum et ultimum bonorum, quod omnium philosophorum sententia tale debet esse, ut eius magnitudinem celeritas, diurnitatem allevatio consoletur. Ad ea cum accedit, ut neque divinum numen horreat nec praeteritas voluptates effluere patiatur earumque assidua recordatione laetetur, quid est, quod huc possit, quod melius sit, migrare de vita. His rebus instructus semper est in voluptate esse aut in armatum hostem impetum fecisse aut in poetis evolvendis, ut ego et Triarius te hortatore facimus, consumeret, in quibus hoc primum est in quo admirer, cur in gravissimis rebus non delectet eos sermo patrius, cum idem fabellas Latinas ad verbum e Graecis expressas non invit legant. Quis enim tam inimicus paene nomini Romano est, qui Ennii Medeam aut Antropam Pacuvii spernat aut reiciat, quod se isdem Euripidis fabulis delectari dicat, Latinas litteras oderit? Synephebos ego, inquit, potius Caecilii aut Andriam Terentii quam utramque Menandri legam? A quibus tantum dissentio, ut, cum Sophocles vel optime scriperit Electram, tamen male conversam Atilii mihi legendam putem, de quo Lucilius: 'ferreum scriptorem', verum, opinor, scriptorem tamen, ut legendus sit. Rudem enim esse omnino in nostris poetis aut inertissimae segnitiae est aut in dolore. Omnis autem privatione doloris putat Epicurus terminari summam voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos iridente, statua est in voluptate aut a voluptate discedere. Nam cum ignoratione rerum bonarum et malarum maxime hominum vita vexetur, ob eumque errorem et voluptatibus maximis saepe priventur et durissimis animi doloribus torqueantur, sapientia est adhibenda, quae et terroribus cupiditatibusque detractis et omnium falsarum opinionum temeritate derapta certissimam se nobis ducem praebeat ad voluptatem. Sapientia enim est una, quae maestitiam pellat ex animis, quae nos exhorrescere metu non sinat. Qua praeceptrice in tranquillitate vivi potest omnium.

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](#) 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.

Creating a title and abstract

Now, let's add a title and an abstract. We'll start with the title. We center align it and increase its font weight by enclosing it in `*stars*`.

```
#align(center, text(17pt) [
  *A fluid dynamic model
  for glacier flow*
])
```

A fluid dynamic model for glacier flow

This looks right. We used the `text` function to override the previous text set rule locally, increasing the size to 17pt for the function's argument. 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")
    ]
)
```

A fluid dynamic model for glacier flow

Therese Tungsten	Dr. John Doe
Artos Institute	Artos Institute
tung@artos.edu	doe@artos.edu

The two author blocks are laid out next to each other. We use the `grid` 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](#) or [fractions](#). 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.

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)
]
```

A fluid dynamic model for glacier flow

A fluid dynamic model for glacier flow

Therese Tungsten
Artos Institute
tung@artos.edu

Dr. John Doe
Artos Institute
doe@artos.edu

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 distingue 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.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 distingue possit, augeri ampli-

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 save the paper title in a variable, so that we do not have to type it twice, for header and title. We can do that with the `let` keyword:

```
#let title = [
    A fluid dynamic model
    for glacier flow
]

...

#set page(
    header: align(
        right + horizon,
        title
    ),
    ...
)

#align(center, text(17pt) [
    *#title*
])
```

...

A fluid dynamic model for glacier flow

A fluid dynamic model for glacier flow

Therese Tungsten
 Artos Institute
 tung@artos.edu

Dr. John Doe
 Artos Institute
 doe@artos.edu

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequa 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.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequa 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non quo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torqueum illum hosti detraxisse, ut aliquam ex eo est consecutus? - Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. - Filium morte multavit. - Si sine causa, nolle me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisse quam illum etiam ipsum dedocere. Sol Democrito magnus videtur, quippe homini eruditio in geometriaque perfecto, huic pedalis fortasse; tantum enim esse omnino in nostris poetis aut inertissimae segnitiae est aut fastidii delicatissimi. Mihi quidem videtur, inermis ac nudus est. Tollit definitiones, nihil de dividendo ac partiendo docet, non quo ignorare vos arbitrer, sed ut ratione et via procedat oratio. Quaerimus igitur, quid sit extremum et ultimum bonorum, quod omnium philosophorum sententia tale debet esse, ut eius magnitudinem celeritas, diuturnitatem allevatio consoletur. Ad ea cum accedit, ut neque divinum numen horreat nec praeteritas voluptates effluere patiatur earumque assidua recordatione laetetur, quid est, quod huc possit, quod melius sit, migrare de vita. His rebus instructus semper est in voluptate esse aut in armatum hostem impetum fecisse aut in poetis evolvendis, ut ego et Triarius te hortatore facimus, consumeret, in quibus hoc primum est in quo admirer, cur in gravissimis rebus non delectet eos sermo patrius, cum idem fabellas Latinas ad verbum e Graecis expressas non invitit legant. Quis enim tam inimicus paene nomini Romano est, qui Ennii Medeam aut Antiopam Pacuvii spernat aut reiciat, quod se isdem Euripidis fabulis delectari dicat, Latinas litteras oderit? Synephebos ego, inquit, potius Caecilii aut Andriam Terentii quam utramque Menandri legam? A quibus tantum dissentio, ut, cum Sophocles vel optime scripserit Electram, tamen male conversam Atilii mihi legendam putem, de quo Lucilius: 'ferreum scriptorem', verum, opinor, scrip-

After we bound the content to the `title` variable, we can use it in functions

and also within markup (prefixed by `#`, like functions). This way, if we decide on another title, we can easily change it in one place.

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`. `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)
```

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna
aliquam quaerat voluptatem. Ut enim aequo
doleamus animo, cum corpore dolemus, fieri.

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 like 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,
        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)
```

A fluid dynamic model for glacier flow

Therese Tungsten
Artos Institute
tung@artos.edu

Dr. John Doe
Artos Institute
doe@artos.edu

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 iridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendum. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudianda sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedit, saluto: 'chaere', inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albuscius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torque illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa,

nolle me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum dedocere. Sol Democrito magnus videtur, quippe homini eruditio in geometriaque perfecto, huic pedalis fortasse; tantum enim esse omnino in nostris poetis aut inertissimae segnitiae est aut fastidii delicatissimi. Mihi quidem videtur, inermis ac nudus est. Tollit definitiones, nihil de dividendo ac partiendo docet, non quo ignorare vos arbitrer, sed ut.

Related Work

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 iridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendum. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut

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` 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. Because the `heading` function does not offer a way to set any of that, we need to write our own heading show rule.

```
#show heading: it => [
    #set align(center)
    #set text(13pt, weight: "regular")
    #block(smallcaps(it.body))
]
```

...

INTRODUCTION

 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequae doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si.

MOTIVATION

 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequae doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem.

This looks great! We used a show rule that applies to all headings. We give it a function that gets passed the heading as a parameter. That parameter can be used as content but it also has some fields like `title`, `numbers`, and

`level` from which we can compose a custom look. Here, we are center-aligning, setting the font weight to "regular" because headings are bold by default, and use the [`smallcaps`](#) function to render the heading's title in small capitals.

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 set rule: This is a [`method`](#) we can call on headings (and other elements) that allows us to filter them by their level. We can use it to differentiate between section and subsection headings:

```
#show heading.where(
    level: 1
): it => block(width: 100%) [
    #set align(center)
    #set text(13pt, weight: "regular")
    #smallcaps(it.body)
]

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

INTRODUCTION

Lorem ipsum dolor sit amet, consectetur adipisc-ing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si.

Motivation. *Lorem ipsum dolor sit amet, consectetur adipisc-ing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem.*

This looks great! We wrote two show rules that each 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 runs. 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:

A fluid dynamic model for glacier flow

A fluid dynamic model for glacier flow

Therese Tungsten
Artos Institute
tungsten@artos.edu

Dr. John Doe
Artos Institute
doe@artos.edu

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequae 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 distingue 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis.

INTRODUCTION
 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequae 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 distingue 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis.

Motivation. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequae 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 distingue 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint

et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedit, saluto: 'chaere' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere'.

Problem Statement. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequae 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 distingue 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint

RELATED WORK

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequae 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 distingue 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint

Review

You have now learned how to create headers and footers, how to use functions and scopes to locally override styles, how to create more complex layouts with the [grid](#) function and how to write show rules for individual functions, and the whole document. You also learned how to use the [where selector](#) 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!

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 ✨!

This function takes a single argument, `term`, and returns a content block with the `term` surrounded by sparkles. We also put the whole thing in a box so that the term we are amazed by cannot be separated from its sparkles by a line break.

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 behind 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.

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.

```
#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
#lorem(90)

...
```

INTRODUCTION

Lorem ipsum dolor sit amet, consectetur adipisc-ing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 distin-guique 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis.

Motivation. Lorem ipsum dolor sit amet, con-sectetur adipisc-ing elit, sed do eiusmod tempor incidi-dunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo doleamus an-imо, 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 distin-guique 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis deb-itibus aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedit, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et cari-tatem, quae sunt vitae.

Problem Statement. Lorem ipsum dolor sit amet, con-sectetur adipisc-ing elit, sed do eiusmod tempor incidi-dunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo doleamus an-imо, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

RELATED WORK

Lorem ipsum dolor sit amet, consectetur adipisc-ing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 distin-guique 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis deb-itibus aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedit, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et cari-tatem, quae sunt vitae.

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.

Templates with named arguments

Our paper in the previous chapter had a title and an author list. Let's add these things to our template. In addition to the title, we want our template to accept a list of authors with their affiliations and the paper's abstract. To keep things readable, we'll add those as named arguments. In the end, we want it to work like this:

```
#show: doc => conf(
    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),
doc,
)

```

...

Let's build this new template function. First, we add a default value to the `title` argument. This way, we can call the template without specifying a `title`. We also add the 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](#) of [dictionaries](#) 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\(\)`](#) 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](#).

We still have to provide an argument to the grid for each author: Here is where the array's [map method](#) 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](#). 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(
    title: none,
    authors: (),
    abstract: [],
    doc,
) = {
    // Set and show rules from before.
    ...

    set align(center)
    text(17pt, 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
]

set align(left)
doc
}

```

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` 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](#) are designed to work with this style of function call.

```

#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.dekran@hstate.hn",
),
),
abstract: lorem(80),
)
```

= Introduction

```
#lorem(90)
```

== Motivation

```
#lorem(140)
```

== Problem Statement

```
#lorem(50)
```

= Related Work

```
#lorem(200)
```

Towards Improved Modelling

Theresa Tungsten
 Artos Institute
 tung@artos.edu

Eugene Deklan
 Honduras State
 e.deklan@hstate.hn

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis.

INTRODUCTION

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis.

Motivation. *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non*

recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedit, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere.'

Problem Statement. *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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.*

RELATED WORK

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo 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 collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non

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

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](#), on our [Discord server](#), on [GitHub](#), or via the web app's feedback form (always available in the Help menu).

So what are you waiting for? [Sign up](#) and write something!

3. Reference

This reference documentation is a comprehensive guide to all of Typst's syntax, concepts, types, and functions. If you are completely new to Typst, we recommend starting with the tutorial and then coming back to the reference to learn more about Typst's features as you need them.

Language

The reference starts with a language part that gives an overview over [Typst's syntax](#) and contains information about concepts involved in [styling documents](#), using [Typst's scripting capabilities](#).

Functions

The second part includes chapters on all functions used to insert, style, transform, and layout content in Typst documents. Each function is documented with a description of its purpose, a list of its parameters, and examples of how to use it.

The final part of the reference explains all functions that are used within Typst's code mode to manipulate and transform data. Just as in the previous part, each function is documented with a description of its purpose, a list of its parameters, and examples of how to use it.

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.

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.

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
Strong emphasis	*strong*	strong
Emphasis	_emphasis_	emph
Raw text	`print(1)`	raw
Link	https://typst.app/	link
Label	<intro>	label
Reference	@intro	ref
Heading	= Heading	heading
Bullet list	- item	list
Numbered list	+ item	enum

Term list	/ Term: description	terms
Math	$\$x^2\$$	Math
Line break	\	linebreak
Smart quote	'single' or "double"	smartquote
Symbol shorthand	~, ---	Symbols
Code expression	#rect (width: 1cm)	Scripting
Character escape	Tweet at us \#ad	Below
Comment	/* block */, // line	Below

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
Block-level math	$\$x^2\$$	Math
Bottom attachment	$\$x_1\$$	attach
Top attachment	$\$x^2\$$	attach
Fraction	$\$1 + (a+b) / 5\$$	frac

Line break	<code>\$x \ y\$</code>	linebreak
Alignment point	<code>\$x &= 2 \ &= 3\$</code>	Math
Variable access	<code>\$#x\$, \$pi\$</code>	Math
Field access	<code>\$arrow.r.long\$</code>	Scripting
Implied multiplication	<code>\$x y\$</code>	Math
Symbol shorthand	<code>\$->\$, \$!=</code>	Symbols
Text/string in math	<code>\$a "is natural"\$</code>	Math
Math function call	<code>\$floor(x)\$</code>	Math
Code expression	<code>\$#rect(width: 1cm)\$</code>	Scripting
Character escape	<code>\$x\^2\$</code>	Below
Comment	<code>/* comment */\$</code>	Below

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	<code>none</code>	none
Auto	<code>auto</code>	auto
Boolean	<code>false, true</code>	bool
Integer	<code>10, 0xff</code>	int
Floating-point number	<code>3.14, 1e5</code>	float

Length	<code>2pt, 3mm, 1em, ..</code>	length
Angle	<code>90deg, 1rad</code>	angle
Fraction	<code>2fr</code>	fraction
Ratio	<code>50%</code>	ratio
String	<code>"hello"</code>	str
Label	<code><intro></code>	label
Math	<code>\$x^2\$</code>	Math
Raw text	<code>`print(1)`</code>	raw
Variable access	<code>x</code>	Scripting
Code block	<code>{ let x = 1; x + 2 }</code>	Scripting
Content block	<code>[*Hello*]</code>	Scripting
Parenthesized expression	<code>(1 + 2)</code>	Scripting
Array	<code>(1, 2, 3)</code>	Array
Dictionary	<code>(a: "hi", b: 2)</code>	Dictionary
Unary operator	<code>-x</code>	Scripting
Binary operator	<code>x + y</code>	Scripting
Assignment	<code>x = 1</code>	Scripting
Field access	<code>x.y</code>	Scripting
Method call	<code>x.flatten()</code>	Scripting
Function call	<code>min(x, y)</code>	Function
Argument spreading	<code>min(..nums)</code>	Arguments

Unnamed function	<code>(x, y) => x + y</code>	Function
Let binding	<code>let x = 1</code>	Scripting
Named function	<code>let f(x) = 2 * x</code>	Function
Set rule	<code>set text(14pt)</code>	Styling
Set-if rule	<code>set text(..) if ..</code>	Styling
Show-set rule	<code>show heading: set block(..)</code>	Styling
Show rule with function	<code>show raw: it => {..}</code>	Styling
Show-everything rule	<code>show: template</code>	Styling
Context expression	<code>context text.lang</code>	Context
Conditional	<code>if x == 1 {...} else ...</code>	Scripting
For loop	<code>for x in (1, 2, 3) ...</code>	Scripting
While loop	<code>while x < 10 ...</code>	Scripting
Loop control flow	<code>break, continue</code>	Scripting
Return from function	<code>return x</code>	Function
Include module	<code>include "bar.typ"</code>	Scripting
Import module	<code>import "bar.typ"</code>	Scripting
Import items from module	<code>import "bar.typ": a, b, c</code>	Scripting
Comment	<code>/* block */, // line</code>	Below

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:

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](#).

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

I got an ice cream for \$1.50! 😊

Paths

Typst has various features that require a file path to reference external resources such as images, Typst files, or data files. Paths are represented as [strings](#). There are two kinds of paths: Relative and absolute.

- A **relative path** searches from the location of the Typst file where the feature is invoked. It is the default:

```
#image("images/logo.png")
```

- An **absolute path** searches from the *root* of the project. It starts with a leading /:

```
#image("/assets/logo.png")
```

Project root

By default, the project root is the parent directory of the main Typst file. For

security reasons, you cannot read any files outside of the root directory.

If you want to set a specific folder as the root of your project, you can use the CLI's `--root` flag. Make sure that the main file is contained in the folder's subtree!

```
typst compile --root .. file.typ
```

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).

Paths and packages

A package can only load files from its own directory. Within it, absolute paths point to the package root, rather than the project root. For this reason, it cannot directly load files from the project directory. If a package needs resources from the project (such as a logo image), you must pass the already loaded image, e.g. as a named parameter `logo: image ("mylogo.svg")`. Note that you can then still customize the image's appearance with a `set` rule within the package. In the future, paths might become a [distinct type from strings](#), so that they can retain knowledge of where they were constructed. This way, resources could be loaded from a different root.

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.

Set rules

With set rules, you can customize the appearance of elements. They are written as a [function call](#) to an [element function](#) 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](#) and [heading numbering](#).

```
#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 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

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](#), a colon and then a set rule. The most basic form of selector is an [element function](#). 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 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](#). This function receives the element in question and can return arbitrary content. The available [fields](#) on the element passed to the function again match the parameters of the respective element function. Below, we define a show rule that formats headings for a fantasy encyclopedia.

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

= 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.

~ *Dragon (I)* ~

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

~ *Manticore (II)* ~

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](#) 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](#) 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.

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 (;).

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

Hello

😊
5

The example above shows a few of the available expressions, including [function calls](#), [field accesses](#), and [method calls](#). 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).

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](#). Content blocks are delimited by

square brackets and can contain arbitrary markup. A content block results in a

value of type [content](#). 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

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. 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](#). 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 add(x, y) = x + y
Sum is #add(2, 3).
```

This is Typst's documentation. It explains Typst.

Sum is 5.

Let bindings can also be used to destructure [arrays](#) and [dictionaries](#). In this case, the left-hand side of the assignment should mirror an array or dictionary. The `..` operator can be used once in the pattern to collect the remainder of the array's or 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 work 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

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` expression. When the condition evaluates to `true`, the conditional yields the value resulting from the `if`'s body, otherwise 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` [...]

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](#). The destructuring syntax described in [Let binding](#) can also be used here.

- `for pair in dict {..}`

Iterates over the key-value pairs of the [dictionary](#). 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](#). 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](#), which can be converted from a [string](#) or [read](#) from a file without encoding. Each byte value is an [integer](#) 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 [..]`

Fields

You can use *dot notation* to access fields on a value. For values of type [content](#), you can also use the [fields](#) function to list the fields.

The value in question can be either:

- a [dictionary](#) that has the specified key,
- a [symbol](#) that has the specified modifier,
- a [module](#) containing the specified definition,
- [content](#) consisting of an element that has the specified field. The available fields match the arguments of the [element function](#) 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
😊
```

Methods

A *method call* is a convenient way to call a function that is scoped to a value's [type](#). For example, we can call the [str.len](#) 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[---])
```

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

4
3
 $a - b - c$

3 is the same as 3

There are a few special functions that modify the value they are called on (e.g. [array.push](#)). 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)`.

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 `bar.typ` and returns the resulting [content](#).

- **Import:** `import "bar.typ"`

Evaluates the file at the path `bar.typ` and inserts the resulting [module](#) 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 `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 path, you can also use a [module value](#), as shown in the following example:

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



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](#).

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

Operators

The following table lists all available unary and binary operators with effect, arity (unary, binary) and precedence level (higher binds stronger).

Operator	Effect	Arity	Precedence
-	Negation	Unary	7
+	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](#) provide context¹ and numberings in the outline, for instance, also provide the proper context to resolve counters.

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](#) 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.

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` 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:

```
#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` function to directly extract the current `location` 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` with `here`, which in combination is equivalent to `get`, and thus get `(2,)`.
- Finally, we use `at` with a `label` 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` function, which works similarly to `counter.at`: It takes a location or other `selector` 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](#). Take a look at the [introspection](#) category for more details on those.

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](#) is.

```
#let foo() = lem.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

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](#)). If Typst cannot resolve everything within five attempts, it will stop and output the warning "layout did not converge within 5 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.

1

Currently, all show rules provide styling context, but only show rules on [locatable](#) elements provide a location context.

3.5 Foundations

Foundational types and functions.

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

Definitions

- [arguments](#)Captured arguments to a function.
- [array](#)A sequence of values.
- [assert](#)Ensures that a condition is fulfilled.
- [auto](#)A value that indicates a smart default.
- [bool](#)A type with two states.
- [bytes](#)A sequence of bytes.
- [calc](#)Module for calculations and processing of numeric values.
- [content](#)A piece of document content.
- [datetime](#)Represents a date, a time, or a combination of both.
- [decimal](#)A fixed-point decimal number type.
- [dictionary](#)A map from string keys to values.
- [duration](#)Represents a positive or negative span of time.
- [eval](#)Evaluates a string as Typst code.
- [float](#)A floating-point number.
- [function](#)A mapping from argument values to a return value.
- [int](#)A whole number.
- [label](#)A label for an element.

- [module](#)An evaluated module, either built-in or resulting from a file.
- [none](#)A value that indicates the absence of any other value.
- [panic](#)Fails with an error.
- [plugin](#)A WebAssembly plugin.
- [regex](#)A regular expression.
- [repr](#)Returns the string representation of a value.
- [selector](#)A filter for selecting elements within the document.
- [str](#)A sequence of Unicode codepoints.
- [style](#)Provides access to active styles.
- [sysModule](#)for system interactions.
- [type](#)Describes a kind of value.
- [version](#)A version with an arbitrary number of components.

3.5.1 Arguments

Captured arguments to a function.

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

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

Constructor

Construct spreadable arguments in place.

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

```

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

```

Body

```

arguments
any
RequiredPositional

```

Variadic

The arguments to construct.

Definitions

at

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

If the key is an [integer](#), this is equivalent to first calling [pos](#) and then [array.at](#).

If it is a [string](#), this is equivalent to first calling [named](#) and then [dictionary.at](#).

```
self.at (
  intstr,
  default: any,
) -> any
```

key

[int](#) or [str](#)

RequiredPositional

The index or name of the argument to get.

default

[any](#)

A default value to return if the key is invalid.

pos

Returns the captured positional arguments as an array.

```
self.pos() -> array
```

named

Returns the captured named arguments as a dictionary.

```
self.named() -> dictionary
```

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](#). Arrays can be added together with the `+` operator, [joined together](#) 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 `()`.

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
```

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.

```
array(
bytesarrayversion
) -> array
#let hi = "Hello ?"
#array(bytes(hi))

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

value
bytes or array or version

RequiredPositional

The value that should be converted to an array.

Definitions

len

The number of values in the array.

```
self.len() -> int
first
```

Returns the first item in the array. May be used on the left-hand side of an assignment. Fails with an error if the array is empty.

```
self.first() -> any
last
```

Returns the last item in the array. May be used on the left-hand side of an assignment. Fails with an error if the array is empty.

```
self.last() -> any
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
index
int
RequiredPositional
```

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

```
default
any
```

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

push

Adds a value to the end of the array.

```
self.push (
    any
)
value
any
RequiredPositional
```

The value to insert at the end of the array.

pop

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

```
self.pop () -> any
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](#).

```
self.insert (
```

```
int,
any,
)
index
int
RequiredPositional
```

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

```
value
any
RequiredPositional
```

The value to insert into the array.

remove

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

```
self.remove(
int,
default: any,
) -> any
index
int
RequiredPositional
```

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

```
default
any
```

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

slice

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

```
self.slice(
int,
noneint,
count: int,
) -> array
start
int
RequiredPositional
```

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

end**none** or **int*****Positional***

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

Default: **none**

count**int**

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

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
```

value**any*****RequiredPositional***

The value to search for.

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
```

```
) -> anynone
```

searcher

```
function
```

RequiredPositional

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

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
) -> noneint
searcher
function
RequiredPositional
```

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

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.

```
array.range(
    int,
    int,
    step: int,
) -> array
    #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)
```

start

int

Positional

The start of the range (inclusive).

Default: 0

```
end
int
RequiredPositional
```

The end of the range (exclusive).

```
step
int
```

The distance between the generated numbers.

Default: 1

```
filter
```

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

```
self.filter (
  function
) -> array
test
function
RequiredPositional
```

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

```
map
```

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

```
self.map (
  function
) -> array
mapper
function
RequiredPositional
```

The function to apply to each item.

```
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](#) with a let binding or for loop.

```
self.enumerate(
start: int
) -> array
start
int
```

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

Default: 0

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
exact
bool
```

Whether all arrays have to have the same length. For example, (1,

2).zip((1, 2, 3), exact: true) produces an error.

Default: false

others

array

RequiredPositional**Variadic**

The arrays to zip with.

fold

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

```
self.fold(
    any,
    function,
) -> any
init
any
RequiredPositional
```

The initial value to start with.

folderfunction***RequiredPositional***

The folding function. Must have two parameters: One for the accumulated value

and one for an item.

sum

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

```
self.sum(
    default: any
) -> any
default
any
```

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

product

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

```
self.product(
    default: any
) -> any
default
any
```

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

any

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

```
self.any(
    function
) -> bool
test
function
RequiredPositional
```

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

all

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

```
self.all(
    function
) -> bool
test
function
RequiredPositional
```

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

flatten

Combine all nested arrays into a single flat one.

```
self.flatten() -> array
rev
```

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

```
self.rev() -> array
split
```

Split the array at occurrences of the specified value.

```
self.split(
    any
) -> array
at
any
RequiredPositional
```

The value to split at.

join

Combine all items in the array into one.

```
self.join()
```

```
anynone,
last: any,
) -> any
separator
any or none
```

Positional

A value to insert between each item of the array.

Default: none

```
last
any
```

An alternative separator between the last two items.

intersperse

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

```
self.intersperse(
any
) -> array
separator
any
```

RequiredPositional

The value that will be placed between each adjacent element.

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.

```
self.chunks(
int,
exact: bool,
) -> array
#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))
```

chunk-sizeint*RequiredPositional*

How many elements each chunk may at most contain.

exactbool

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

Default: `false`

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.

```
self.windows(
int
) -> 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),  
)
```

window-sizeint*RequiredPositional*

How many elements each window will contain.

sorted

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

Returns an error if two values could not be compared or if the key function (if given) yields an error.

```
self.sorted(
    key: function
) -> array
key
function
```

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

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.

```
self.dedup(
    key: function
) -> array
# (1, 1, 2, 3, 1).dedup()
```

(1, 2, 3)

key
function

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

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.

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

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

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`](#) with the first element of the array as the initial accumulator value, folding every subsequent element into it.

```
self.reduce(
    function
) -> any
reducer
function
RequiredPositional
```

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](#) and [assert.ne](#).

Example

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

Parameters

```
assert(  
    bool,  
    message: str,  
)  
condition  
bool  
RequiredPositional
```

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

```
message  
str
```

The error message when the assertion fails.

Definitions

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(  
    any,  
    any,  
    message: str,  
)
```

View example

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

left

any

RequiredPositional

The first value to compare.

right

any

RequiredPositional

The second value to compare.

message

`str`

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

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(
    any,
    any,
    message: str,
)
```

View example

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

left

any

RequiredPositional

The first value to compare.

right

any

RequiredPositional

The second value to compare.

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](#) parameter. Setting it to `auto` lets Typst automatically determine the direction from the [text language](#).

3.5.5 Boolean

A type with two states.

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

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](#) between `0` and `255`, but represented much more efficiently. You can iterate over it using a [for loop](#).

You can convert

- a [string](#) or an [array](#) of integers to bytes with the [bytes](#) constructor
- bytes to a string with the [str](#) constructor, with UTF-8 encoding
- bytes to an array of integers with the [array](#) constructor

When [reading](#) 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
```

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(
  strbytesarray
) -> bytes
```

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

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

value

str or bytes or array

RequiredPositional

The value that should be converted to bytes.

Definitions

len

The length in bytes.

```
self.len() -> int
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
index
int
```

RequiredPositional

The index at which to retrieve the byte.

```
default
any
```

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

slice

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

```
self.slice()
```

```
int,
noneint,
count: int,
) -> bytes
start
int
RequiredPositional
```

The start index (inclusive).

end

`none or int`

Positional

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

Default: `none`

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`.

Functions

abs

Calculates the absolute value of a numeric value.

```
calc.abs(
intfloatlengthangleratiofractiondecimal
) -> any
```

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

5
51.69pt
2fr
342.440

value

[int](#) or [float](#) or [length](#) or [angle](#) or [ratio](#) or [fraction](#) or [decimal](#)

RequiredPositional

The value whose absolute value to calculate.

pow

Raises a value to some exponent.

```
calc.pow(
    intfloatdecimal,
    intfloat,
    ) -> intfloatdecimal
    #calc.pow(2, 3) \
    #calc.pow(decimal("2.5"), 2)
```

8
6.25

base

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The base of the power.

If this is a [decimal](#), the exponent can only be an [integer](#).

exponent

[int](#) or [float](#)

RequiredPositional

The exponent of the power.

exp

Raises a value to some exponent of e.

```
calc.exp(
intfloat
) -> float
#calc.exp(1)
```

2.718281828459045

exponent

int or float

RequiredPositional

The exponent of the power.

sqrt

Calculates the square root of a number.

```
calc.sqrt(
intfloat
) -> float
#calc.sqrt(16) \
#calc.sqrt(2.5)
```

4

1.5811388300841898

value

int or float

RequiredPositional

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

root

Calculates the real nth root of a number.

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

```
calc.root(
    float,
    int,
) -> float
#calc.root(16.0, 4) \
#calc.root(27.0, 3)
```

2
3

radicand`float`*RequiredPositional*

The expression to take the root of

index`int`*RequiredPositional*

Which root of the radicand to take

`sin`

Calculates the sine of an angle.

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

```
calc.sin(
    intfloatangle
) -> float
#calc.sin(1.5) \
#calc.sin(90deg)
```

0.9974949866040544

1

angle`int` or `float` or `angle`*RequiredPositional*

The angle whose sine to calculate.

cos

Calculates the cosine of an angle.

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

```
calc.cos(
intfloatangle
) -> float
#calc.cos(1.5) \
#calc.cos(90deg)
```

```
0.0707372016677029
0.0000000000000006123233995736766
```

angle

[int](#) or [float](#) or [angle](#)

RequiredPositional

The angle whose cosine to calculate.

tan

Calculates the tangent of an angle.

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

```
calc.tan(
intfloatangle
) -> float
#calc.tan(1.5) \
#calc.tan(90deg)
```

```
14.101419947171719
16331239353195370
```

angle

[int](#) or [float](#) or [angle](#)

RequiredPositional

The angle whose tangent to calculate.

asin

Calculates the arcsine of a number.

```
calc.asin(
intfloat
) -> angle
#calc.asin(0) \
#calc.asin(1)
```

0deg
90deg

value

int or float

RequiredPositional

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

acos

Calculates the arccosine of a number.

```
calc.acos(
intfloat
) -> angle
#calc.acos(0) \
#calc.acos(1)
```

90deg
0deg

value

int or float

RequiredPositional

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

atan

Calculates the arctangent of a number.

```
calc.atan(
```

```
intfloat
) -> angle
#calc.atan(0) \
#calc.atan(1)
```

0deg
45deg

value

int or float

RequiredPositional

The number whose arctangent to calculate.

atan2

Calculates the four-quadrant arctangent of a coordinate.

The arguments are (`x`, `y`), not (`y`, `x`).

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

45deg
-123.69deg

x

int or float

RequiredPositional

The X coordinate.

y

int or float

RequiredPositional

The Y coordinate.

sinh

Calculates the hyperbolic sine of a hyperbolic angle.

```
calc.sinh(
    float
) -> float
    #calc.sinh(0) \
    #calc.sinh(1.5)
```

```
0
2.1292794550948173
```

valuefloat*RequiredPositional*

The hyperbolic angle whose hyperbolic sine to calculate.

cosh

Calculates the hyperbolic cosine of a hyperbolic angle.

```
calc.cosh(
    float
) -> float
    #calc.cosh(0) \
    #calc.cosh(1.5)
```

```
1
2.352409615243247
```

valuefloat*RequiredPositional*

The hyperbolic angle whose hyperbolic cosine to calculate.

tanh

Calculates the hyperbolic tangent of an hyperbolic angle.

```
calc.tanh(
    float
) -> float
```

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

0
0.9051482536448664

valuefloat*RequiredPositional*

The hyperbolic angle whose hyperbolic tangent to calculate.

log

Calculates the logarithm of a number.

If the base is not specified, the logarithm is calculated in base 10.

```
calc.log(
intfloat,
base: float,
) -> float
#calc.log(100)
```

2

valueint or float*RequiredPositional*

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

basefloat

The base of the logarithm. May not be zero.

Default: `10.0`

ln

Calculates the natural logarithm of a number.

```
calc.ln()
```

```
intfloat
) -> float
#calc.ln(calc.e)
```

1

valueint or float*RequiredPositional*

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

fact

Calculates the factorial of a number.

```
calc.fact(
int
) -> int
#calc факт(5)
```

120

numberint*RequiredPositional*

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

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.

```
calc.perm(
int,
int,
) -> int
$ "perm"(n, k) &= n! / ((n - k)!) \
"perm"(5, 3) &= #calc.perm(5, 3) $
```

$$\text{perm}(n, k) = \frac{n!}{(n - k)!}$$

$$\text{perm}(5, 3) = 60$$

baseint*RequiredPositional*

The base number. Must be non-negative.

numbersint*RequiredPositional*

The number of permutations. Must be non-negative.

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.

```
calc.binom(
    int,
    int,
    ) -> int
        #calc.binom(10, 5)
```

252

nint*RequiredPositional*

The upper coefficient. Must be non-negative.

kint*RequiredPositional*

The lower coefficient. Must be non-negative.

gcd

Calculates the greatest common divisor of two integers.

```
calc.gcd(
    int,
    int,
) -> int
#calc.gcd(7, 42)
```

7

a
`int`
RequiredPositional

The first integer.

b
`int`
RequiredPositional

The second integer.

`lcm`

Calculates the least common multiple of two integers.

```
calc.lcm(
    int,
    int,
) -> int
#calc.lcm(96, 13)
```

1248

a
`int`
RequiredPositional

The first integer.

b
`int`
RequiredPositional

The second integer.

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](#), and will error if the resulting [float](#) or [decimal](#) is larger than the maximum 64-bit signed integer or smaller than the minimum for that type.

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

500

value

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The number to round down.

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](#), and will error if the resulting [float](#) or [decimal](#) is larger than the maximum 64-bit signed integer or smaller than the minimum for that type.

```
calc.ceil(
    int|float|decimal
) -> int
    #calc.ceil(500.1)
    #assert(calc.ceil(3) == 3)
```

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

501

value

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The number to round up.

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](#), and will error if the resulting [float](#) or [decimal](#) is larger than the maximum 64-bit signed integer or smaller than the minimum for that type.

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

15

value

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The number to truncate.

fract

Returns the fractional part of a number.

If the number is an integer, returns 0.

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

-0.10000000000000009

value

int or float or decimal

RequiredPositional

The number to truncate.

round

Rounds a number to the nearest integer away from zero.

Optionally, a number of decimal places can be specified.

If the number of digits is 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 will return a float, and to a decimal, another decimal. You may explicitly convert the output of this function to an integer with int, 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(
    intfloatdecimal,
    digits: int,
) -> intfloatdecimal
    #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

value

`int` or `float` or `decimal`

RequiredPositional

The number to round.

digits

`int`

If positive, the number of decimal places.

If negative, the number of significant integer digits that should be removed before the decimal point.

Default: `0`

clamp

Clamps a number between a minimum and maximum value.

```
calc.clamp(
    intfloatdecimal,
    intfloatdecimal,
    intfloatdecimal,
) -> intfloatdecimal
    #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

value

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The number to clamp.

min

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The inclusive minimum value.

max

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The inclusive maximum value.

min

Determines the minimum of a sequence of values.

```
calc.min(
    ..any
) -> any
```

```
#calc.min(1, -3, -5, 20, 3, 6) \
#calc.min("typst", "is", "cool")
```

-5
cool

values

any

*RequiredPositional**Variadic*

The sequence of values from which to extract the minimum. Must not be empty.

max

Determines the maximum of a sequence of values.

```
calc.max(
..any
) -> any
#calc.max(1, -3, -5, 20, 3, 6) \
#calc.max("typst", "is", "cool")
```

20
typst

values

any

*RequiredPositional**Variadic*

The sequence of values from which to extract the maximum. Must not be empty.

even

Determines whether an integer is even.

```
calc.even(
int
) -> bool
#calc.even(4) \
#calc.even(5) \
```

```
#range(10).filter(calc.even)
```

```
true
false
(0, 2, 4, 6, 8)
```

valueint*RequiredPositional*

The number to check for evenness.

odd

Determines whether an integer is odd.

```
calc.odd(
    int
) -> bool
#calc.odd(4) \
#calc.odd(5) \
#range(10).filter(calc.odd)
```

```
false
true
(1, 3, 5, 7, 9)
```

valueint*RequiredPositional*

The number to check for oddness.

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](#) input and the dividend is too small in magnitude compared to the divisor.

```
calc.rem(
    intfloatdecimal,
    intfloatdecimal,
) -> intfloatdecimal
    #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

dividend

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The dividend of the remainder.

divisor

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The divisor of the remainder.

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.

```
calc.div-euclid(
    intfloatdecimal,
    intfloatdecimal,
) -> intfloatdecimal
    #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
```

dividend

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The dividend of the division.

divisor

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The divisor of the division.

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](#) input and the dividend is too small in magnitude compared to the divisor.

```
calc.rem-euclid(
    int|float|decimal,
    int|float|decimal,
) -> int|float|decimal
```

```
#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"))
```

```
1
1
2
2
0.25
0.25
```

dividend

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The dividend of the remainder.

divisor

[int](#) or [float](#) or [decimal](#)

RequiredPositional

The divisor of the remainder.

quo

Calculates the quotient (floored division) of two numbers.

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

```
calc.quo(
    int|float|decimal,
    int|float|decimal,
) -> int
$ "quo"(a, b) &= floor(a/b) \
    "quo"(14, 5) &= #calc.quo(14, 5) \
```

```
"quo" (3.46, 0.5) &= #calc.quo(3.46, 0.5) $
```

$$\text{quo}(a, b) = \left\lfloor \frac{a}{b} \right\rfloor$$

`quo(14, 5) = 2`

`quo(3.46, 0.5) = 6`

dividend

`int` or `float` or `decimal`

RequiredPositional

The dividend of the quotient.

divisor

`int` or `float` or `decimal`

RequiredPositional

The divisor of the quotient.

3.5.8 Content

A piece of document content.

This type is at the heart of Typst. All markup you write and most [functions](#) 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.

Example

```
Type of *Hello!* is
#type([*Hello!*])
```

Type of **Hello!** is content

Content can be added with the `+` operator, [joined together](#) and multiplied with integers. Wherever content is expected, you can also pass a [string](#) or [none](#).

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](#).

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](#) 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](#). 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](#) function.

Definitions

`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  
has
```

Whether the content has the specified field.

```
self.has(  
  str  
) -> bool  
field  
  str  
RequiredPositional
```

The field to look for.

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  
field  
  str  
RequiredPositional
```

The field to access.

default
 any

A default value to return if the field does not exist.

fields

Returns the fields of this content.

```
self.fields() -> dictionary  
  #rect(  
    width: 10cm,  
    height: 10cm,  
  ) .fields()
```

```
(width: 0% + 283.46pt, height: 0% +
283.46pt)
```

`location`

The location of the content. This is only available on content returned by [query](#) or provided by a [show rule](#), for other content it will be `none`. The resulting location can be used with [counters](#), [state](#) and [queries](#).

```
self.location() -> nonelocation
```

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](#).

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

Datetime and Duration

You can get a [duration](#) 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

Format

You can specify a customized formatting using the [display](#) 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.

`opadding`: Can be either `zero`, `space` or `none`. Specifies how the year is padded.

`orepr`: 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.

`osign`: Can be either `automatic` or `mandatory`. Specifies when the sign should be displayed.

- `month`: Displays the month of the datetime.

`opadding`: Can be either `zero`, `space` or `none`. Specifies how the month is padded.

`orepr`: 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.

`opadding`: Can be either `zero`, `space` or `none`. Specifies how the day is padded.

- `week_number`: Displays the week number of the datetime.

`opadding`: Can be either `zero`, `space` or `none`. Specifies how the week number is padded.

`orepr`: 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.

`orepr` 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).

`oone_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.

`opadding`: Can be either `zero`, `space` or `none`. Specifies how the hour is padded.

`orepr`: 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

`ocase`: Can be `lower` to display it in lower case and `upper` to display it in upper case.

- `minute`: Displays the minute of the date.

`opadding`: Can be either `zero`, `space` or `none`. Specifies how the minute is padded.

- `second`: Displays the second of the date.

`opadding`: Can be either `zero`, `space` or `none`. Specifies how the second is padded.

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 you cannot use components such as `hour` or `minute`, which would only work on datetimes that have a specified time.

Constructor

Creates a new datetime.

You can specify the [datetime](#) 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](#) method will choose a different formatting by default.

```
datetime (
    year: int,
    month: int,
    day: int,
    hour: int,
    minute: int,
    second: int,
) -> datetime
#datetime (
    year: 2012,
    month: 8,
    day: 3,
).display()
```

2012-08-03

year

`int`

The year of the datetime.

month

`int`

The month of the datetime.

day

`int`

The day of the datetime.

hour

`int`

The hour of the datetime.

minute

`int`

The minute of the datetime.

secondint

The second of the datetime.

Definitions

today

Returns the current date.

```
datetime.today(
    offset: autoint
) -> datetime
    Today's date is
    #datetime.today().display().
```

Today's date is 1970-01-01.

offsetauto or int

An offset to apply to the current UTC date. If set to `auto`, the offset will be the local offset.

Default: `auto`

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](#) for more information.

```
self.display(
```

```
  autostr
```

```
) -> str
```

```
pattern
```

```
  auto or str
```

Positional

The format used to display the datetime.

Default: **auto**

year

The year if it was specified, or **none** for times without a date.

```
self.year() -> noneint
```

month

The month if it was specified, or **none** for times without a date.

```
self.month() -> noneint
```

weekday

The weekday (counting Monday as 1) or **none** for times without a date.

```
self.weekday() -> noneint
```

day

The day if it was specified, or **none** for times without a date.

```
self.day() -> noneint
```

hour

The hour if it was specified, or **none** for dates without a time.

```
self.hour() -> noneint
```

minute

The minute if it was specified, or **none** for dates without a time.

```
self.minute() -> noneint
```

second

The second if it was specified, or **none** for dates without a time.

```
self.second() -> noneint
```

ordinal

The ordinal (day of the year), or **none** for times without a date.

```
self.ordinal() -> noneint
```

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.

Example

```
Decimal: #(decimal("0.1") + decimal("0.2")) \
Float: #(0.1 + 0.2)
```

```
Decimal: 0.3
Float: 0.3000000000000004
```

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](#) to a decimal with the

`decimal(int)` constructor, e.g. `decimal(59)`. However, note that constructing a decimal from a [floating-point number](#), 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`](#), but there are still no precision guarantees for that kind of conversion.

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.

Displaying decimals

To display a decimal, simply insert the value into the document. To only display a certain number of digits, [`round`](#) 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`](#) 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).

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`](#), 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`](#) 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.

Constructor

Converts a value to a `decimal`.

It is recommended to use a string to construct the decimal number, or an [integer](#) (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](#) 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(
  intfloatstr
) -> decimal
#decimal("1.2222222222222222")
```

`1.222222222222222`

value

[int](#) or [float](#) or [str](#)

RequiredPositional

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, 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](#) (`.key`) to access the value. Dictionaries can be added with the `+` operator and [joined together](#). 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](#). This will iterate in the order the pairs were inserted / declared.

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
```

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.

```
dictionary(
module
) -> dictionary
#dictionary(sys).at("version")
```

0.12.0

value
module
RequiredPositional

The value that should be converted to a dictionary.

Definitions

len

The number of pairs in the dictionary.

```
self.len() -> int
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.

```
self.at (
    str,
    default: any,
) -> any
key
str
RequiredPositional
```

The key at which to retrieve the item.

```
default
any
```

A default value to return if the key is not part of the dictionary.

insert

Inserts a new pair into the dictionary. If the dictionary already contains this key, the value is updated.

```
self.insert (
    str,
    any,
)
key
str
RequiredPositional
```

The key of the pair that should be inserted.

```
value
any
RequiredPositional
```

The value of the pair that should be inserted.

remove

Removes a pair from the dictionary by key and return the value.

```
self.remove (
    str,
    default: any,
) -> any
key
str
```

RequiredPositional

The key of the pair to remove.

default

`any`

A default value to return if the key does not exist.

keys

Returns the keys of the dictionary as an array in insertion order.

`self.keys() -> array`

values

Returns the values of the dictionary as an array in insertion order.

`self.values() -> array`

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`

3.5.12 Duration

Represents a positive or negative span of time.

Constructor

Creates a new duration.

You can specify the [duration](#) using weeks, days, hours, minutes and seconds.

You can also get a duration by subtracting two [datetimes](#).

```
duration(
    seconds: int,
    minutes: int,
    hours: int,
    days: int,
    weeks: int,
) -> duration
#duration(
    days: 3,
    hours: 12,
```

```
) .hours()
```

84

seconds

int

The number of seconds.

Default: 0

minutes

int

The number of minutes.

Default: 0

hours

int

The number of hours.

Default: 0

days

int

The number of days.

Default: 0

weeks

int

The number of weeks.

Default: 0

Definitions

seconds

The duration expressed in seconds.

This function returns the total duration represented in seconds as a floating-point number rather than the second component of the duration.

```
self.seconds() -> float
minutes
```

The duration expressed in minutes.

This function returns the total duration represented in minutes as a floating-point number rather than the second component of the duration.

```
self.minutes() -> float
hours
```

The duration expressed in hours.

This function returns the total duration represented in hours as a floating-point number rather than the second component of the duration.

```
self.hours() -> float
days
```

The duration expressed in days.

This function returns the total duration represented in days as a floating-point number rather than the second component of the duration.

```
self.days() -> float
weeks
```

The duration expressed in weeks.

This function returns the total duration represented in weeks as a floating-point number rather than the second component of the duration.

```
self.weeks() -> float
```

3.5.13 Evaluate

This function should only be used as a last resort.

Example

```
#eval("1 + 1") \
#eval("(1, 2, 3, 4)").len() \
#eval("*Markup!*", mode: "markup") \
```

2
4
Markup!

Parameters

```
eval(
    str,
    mode: str,
    scope: dictionary,
) -> any
source
str
RequiredPositional
```

A string of Typst code to evaluate.

mode
str

The [syntactical mode](#) in which the string is parsed.

Variant	Details
"code"	Evaluate as code, as after a hash.
"markup"	Evaluate as markup, like in a Typst file.
"math"	Evaluate as math, as in an equation.

Default: "code"

[View example](#)

```
#eval ("= Heading", mode: "markup")
#eval ("1_2^3", mode: "math")
```

Heading

$\frac{1}{2}^3$

scope
dictionary

A scope of definitions that are made available.

Default: `(:)`

View example

```
#eval ("x + 1", scope: (x: 2)) \
#eval (
  "abc/xyz",
  mode: "math",
  scope: (
    abc: $a + b + c$,
    xyz: $x + y + z$,
  ),
)
```

$$\frac{a+b+c}{x+y+z}$$

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](#).

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.

Example

```
#3.14 \
#1e4 \
#(10 / 4)
```

```
3.14
10000
2.5
```

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(
bool int float ratio str decimal
) -> float
  #float(false) \
  #float(true) \
  #float(4) \
  #float(40%) \
  #float("2.7") \
  #float("1e5")
```

```
0
1
4
0.4
2.7
100000
```

value

```
bool or int or float or ratio or str or decimal
```

RequiredPositional

The value that should be converted to a float.

Definitions

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.

```
self.is-nan() -> bool
#float.is-nan(0) \
#float.is-nan(1) \
#float.is-nan(float.nan)
```

```
false
false
true
```

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.

```
self.is-infinite() -> bool
#float.is-infinite(0) \
#float.is-infinite(1) \
#float.is-infinite(float.inf)
```

```
false
false
true
```

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`.

```
self.signum() -> float
#(5.0).signum() \
#(-5.0).signum() \
#(0.0).signum() \
#float.nan.signum()
```

```
1
-1
1
NaN
```

from-bytes

Converts bytes to a float.

```
float.from-bytes(
bytes,
Endian: str,
) -> 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
```

bytes

bytes

RequiredPositional

The bytes that should be converted to a float.

Must be of length exactly 8 so that the result fits into a 64-bit float.

Endian

str

The endianness of the conversion.

Variant	Details
"big"	Big-endian byte order: The highest-value byte is at the beginning of the bytes.

	Little-endian byte order: The lowest-value byte is at the beginning of the bytes.
--	---

Default: "little"

to-bytes

Converts a float to bytes.

```
self.to-bytes(
    endian: str
) -> 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)
```

endian

str

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.

Default: "little"

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 blocks 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](#) for more details.

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](#) and [show](#) rules to learn about additional ways you can work with functions in Typst.

Element functions

Some functions are associated with *elements* like [headings](#) or [tables](#). When called, these create an element of their respective kind. In contrast to normal functions, they can further be used in [set rules](#), [show rules](#), and [selectors](#).

Function scopes

Functions can hold related definitions in their own scope, similar to a [module](#). Examples of this are [assert.eq](#) or [list.item](#). However, this feature is currently only available for built-in functions.

Defining functions

You can define your own function with a [let binding](#) 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](#). 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](#), 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](#) instead. The return type of such functions is not explicitly specified in the documentation.

(An example of this is [array.push](#)).

```
#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
]
```

Warning:
Danger is imminent!

Warning:
KEEP OFF TRACKS

Importing functions

Functions can be imported from one file ([module](#)) 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`.

Unnamed functions

You can also created 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 for settable properties that take functions like the page function's [footer](#) property.

```
#show "once?": it => [#it #it]
once?
```

once? once?

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\(value\)](#). These can modify the values they are called on.

Definitions

`with`

Returns a new function that has the given arguments pre-applied.

```
self.with(
..any
) -> function
arguments
any
RequiredPositional
Variadic
```

The arguments to apply to the function.

where

Returns a selector that filters for elements belonging to this function whose fields have the values of the given arguments.

```
self.where(
..any
) -> selector
#show heading.where(level: 2): set text(blue)
= Section
== Subsection
==== Sub-subsection
```

Section

Subsection

Sub-subsection

fields

```
any
RequiredPositional
Variadic
```

The fields to filter for.

3.5.16 Integer

A whole number.

The number can be negative, zero, or positive. As Typst uses 64 bits to store integers, integers cannot be smaller than `-9223372036854775808` or larger than `9223372036854775807`.

The number can also be specified as hexadecimal, octal, or binary by starting it with a zero followed by either `x`, `o`, or `b`.

You can convert a value to an integer with this type's constructor.

Example

```
#(1 + 2) \
#(2 - 5) \
#(3 + 4 < 8)

#0xff \
#0o10 \
#0b1001
```

```
3
-3
true

255
8
9
```

Constructor

Converts a value to an integer. Raises an error if there is an attempt to produce an integer larger than the maximum 64-bit signed integer or smaller than the minimum 64-bit signed integer.

- Booleans are converted to `0` or `1`.
- Floats and decimals are truncated to the next 64-bit integer.

- Strings are parsed in base 10.

```
int(
bool int float str decimal
) -> int
#int(false) \
#int(true) \
#int(2.7) \
#int(decimal("3.8")) \
#(int("27") + int("4"))
```

0
1
2
3
31

value
bool or int or float or str or decimal

RequiredPositional

The value that should be converted to an integer.

Definitions

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.

```
self.signum() -> int
#(5).signum() \
#(-5).signum() \
#(0).signum()
```

```
1
-1
0
```

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.

```
self.bit-not() -> int
#4.bit-not() \
#(-1).bit-not()
```

```
-5
0
```

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.

```
self.bit-and(
int
) -> int
#128.bit-and(192)
```

```
128
```

rhs

```
int
```

RequiredPositional

The right-hand operand of the bitwise AND.

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.

```
self.bit-or(
    int
) -> int
#64.bit-or(32)
```

96

rhs
`int`
RequiredPositional

The right-hand operand of the bitwise OR.

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.

```
self.bit-xor(
    int
) -> int
#64.bit-xor(96)
```

32

rhs
`int`
RequiredPositional

The right-hand operand of the bitwise XOR.

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.

```
self.bit-lshift(
    int
) -> int
#33.bit-lshift(2) \
#(-1).bit-lshift(3)
```

132
-8

shift
int
RequiredPositional

The amount of bits to shift. Must not be negative.

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.

```
self.bit-rshift(
    int,
    logical: bool,
) -> int
#64.bit-rshift(2) \
#(-8).bit-rshift(2) \
#(-8).bit-rshift(2, logical: true)
```

16
-2
4611686018427387902

shiftint**RequiredPositional**

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.

logicalbool

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.

Default: `false`

from-bytes

Converts bytes to an integer.

```
int.from-bytes(
    bytes,
    endian: str,
    signed: bool,
) -> int
    #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

bytesbytes*RequiredPositional*

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.

Endianstr

The endianness of the conversion.

Variant**Details**

Big-endian byte order: The highest-value byte is at the beginning of "big" the bytes.

Little-endian byte order: The lowest-value byte is at the beginning of "little" the bytes.

Default: "little"

signedbool

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.

Default: `true`

to-bytes

Converts an integer to bytes.

```
self.to-bytes(
    endian: str,
    size: int,
) -> 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)
```

Endian[str](#)

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.
Default: "little"	
size	
<u>int</u>	
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 <code>int.from-bytes</code> , the resulting number might be positive, as the most significant bit might not be set to 1.	
Default: 8	

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](#), [queried](#) for, and [styled](#) through its label.

Once constructed, you can get the name of a label using [str](#).

Example

```
#show <a>: set text(blue)
#show label("b"): set text(red)

= Heading <a>
*strong* #label("b")
```

Heading
Strong

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 `..`.

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.

Constructor

Creates a label from a string.

```
label (
  str
) -> label
name
str
RequiredPositional
```

The name of the label.

3.5.18 Module

An evaluated module, either built-in or resulting from a file.

You can access definitions from the module using [field access notation](#) and interact with it using the [import and include syntaxes](#). Alternatively, it is possible to convert a module to a dictionary, and therefore access its contents dynamically, using the [dictionary constructor](#).

Example

```
#import "utils.typ"
utils.add(2, 5)

#import utils: sub
#sub(1, 4)
```

7

-3

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](#) with any value, yielding the other value.

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.

Example

The code below produces the error `panicked` with: "this is wrong".

```
#panic("this is wrong")
```

Parameters

```
panic(  
..any
```

```
)  

values  

any  

RequiredPositional  

Variadic
```

The values to panic with and display to the user.

3.5.21 Plugin

A WebAssembly plugin.

Typst is capable of interfacing with plugins compiled to WebAssembly. Plugin functions may accept multiple [byte buffers](#) 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.

Plugins run in isolation from your system, which means that printing, reading files, or anything like that will not be supported for security reasons. To run as a plugin, a program needs to be compiled to a 32-bit shared WebAssembly library. Many compilers will use the [WASI ABI](#) 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](#).

Plugins and Packages

Plugins are distributed as packages. A package can make use of a plugin simply by including a WebAssembly file and loading it. Because the byte-based plugin interface is quite low-level, plugins are typically exposed through wrapper functions, that also live in the same package.

Purity

Plugin functions must be pure: Given the same arguments, they 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.

Example

```
#let myplugin = plugin("hello.wasm")
#let concat(a, b) = str(
    myplugin.concatenate(
        bytes(a),
        bytes(b),
    )
)

#concat("hello", "world")
```

hello*world

Protocol

To be used as a plugin, a WebAssembly module must conform to the following protocol:

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, \dots, 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 $a_1 + a_2 + \dots + a_n$, 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.

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](#) 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.

Resources

For more resources, check out the [wasm-minimal-protocol repository](#). It

contains:

- A list of example plugin implementations and a test runner for these examples
- Wrappers to help you write your plugin in Rust (Zig wrapper in development)
- A stubber for WASI

Constructor

Creates a new plugin from a WebAssembly file.

```
plugin(
  str
) -> plugin
path
str
RequiredPositional
```

Path to a WebAssembly file.

For more details, see the [Paths section](#).

3.5.22 Regex

A regular expression.

Can be used as a [show rule selector](#) and with [string methods](#) like `find`, `split`, and `replace`.

[See here](#) for a specification of the supported syntax.

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.

Constructor

Create a regular expression from a string.

```
regex(  
  str  
) -> regex  
regex  
str  
RequiredPositional
```

The regular expression as a string.

Most regex escape sequences just work because they are not valid Typst escape sequences. To produce regex escape sequences that are also valid in Typst (e.g. `\\"`), you need to escape twice. Thus, to match a verbatim backslash, you would need to write `regex("\\\\")`.

If you need many escape sequences, you can also create a raw element and extract its text to use it for your regular expressions:

[View example](#)

```
regex(`\d+\.\d+\.\d+` .text).
```

3.5.23 Representation

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.

Note: This function is for debugging purposes. Its output should not be considered stable and may change at any time!

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])
```

Parameters

```
repr(
any
) -> str
value
any
RequiredPositional
```

The value whose string representation to produce.

3.5.24 Selector

A filter for selecting elements within the document.

You can construct a selector in the following ways:

- you can use an element [function](#)
- you can filter for an element function with [specific fields](#)
- you can use a [string](#) or [regular expression](#)
- you can use a [`<label>`](#)
- you can use a [`location`](#)
- call the [`selector`](#) 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](#) to elements. You can also use selectors to [query](#) 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`](#) 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.

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.

Constructor

Turns a value into a selector. The following values are accepted:

- An element function like a `heading` or `figure`.
- A `<label>`.

- A more complex selector like `heading.where(level: 1)`.

```
selector(
  ...strregexlabelselectorlocationfunction
) -> selector
target

str or regex or label or selector or location or function
```

RequiredPositional

Can be an element function like a `heading` or `figure`, a `<label>` or a more complex selector like `heading.where(level: 1)`.

Definitions

or

Selects all elements that match this or any of the other selectors.

```
self.or(
  ...strregexlabelselectorlocationfunction
) -> selector
others

str or regex or label or selector or location or function
```

RequiredPositional

Variadic

The other selectors to match on.

and

Selects all elements that match this and all of the other selectors.

```
self.and(
  ...strregexlabelselectorlocationfunction
) -> selector
others

str or regex or label or selector or location or function
```

RequiredPositional

Variadic

The other selectors to match on.

before

Returns a modified selector that will only match elements that occur before the first match of `end`.

```
self.before(
    labelselectorlocationfunction,
    inclusive: bool,
) -> selector
end

label or selector or location or function
```

RequiredPositional

The original selection will end at the first match of `end`.

inclusive
bool

Whether `end` itself should match or not. This is only relevant if both selectors match the same type of element. Defaults to `true`.

Default: `true`

after

Returns a modified selector that will only match elements that occur after the first match of `start`.

```
self.after(
    labelselectorlocationfunction,
    inclusive: bool,
) -> selector
start

label or selector or location or function
```

RequiredPositional

The original selection will start at the first match of `start`.

inclusive
bool

Whether `start` itself should match or not. This is only relevant if both selectors match the same type of element. Defaults to `true`.

Default: `true`

3.5.25 String

A sequence of Unicode codepoints.

You can iterate over the grapheme clusters of the string using a [for loop](#).

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](#) and multiplied with integers.

Typst provides utility methods for string manipulation. Many of these methods (e.g., `split`, `trim` and `replace`) operate on *patterns*: A pattern can be either a string or a [regular expression](#). 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 this type's constructor.

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
```

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

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.
- 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](#) and [from-unicode](#) functions.

```
str(
    int float str bytes label decimal version type,
    base: int,
) -> str
    #str(10) \
    #str(4000, base: 16) \
    #str(2.7) \
    #str(1e8) \
    #str(<intro>)
```

```
10
fa0
2.7
100000000
intro
```

value

`int` or `float` or `str` or `bytes` or `label` or `decimal` or `version` or `type`

RequiredPositional

The value that should be converted to a string.

base

`int`

The base (radix) to display integers in, between 2 and 36.

Default: `10`

Definitions

len

The length of the string in UTF-8 encoded bytes.

`self.len() -> int`

first

Extracts the first grapheme cluster of the string. Fails with an error if the string is empty.

`self.first() -> str`

last

Extracts the last grapheme cluster of the string. Fails with an error if the string is empty.

`self.last() -> str`

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  
index  
int  
RequiredPositional
```

The byte index. If negative, indexes from the back.

default
any

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

slice

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

```
self.slice(  
    int,  
    noneint,  
    count: int,  
    ) -> str  
start  
int  
RequiredPositional
```

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

end

none or int

Positional

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

Default: none

count
int

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

clusters

Returns the grapheme clusters of the string as an array of substrings.

```
self.clusters() -> array
codepoints
```

Returns the Unicode codepoints of the string as an array of substrings.

```
self.codepoints() -> array
to-unicode
```

Converts a character into its corresponding code point.

```
str.to-unicode(
str
) -> int
#"a".to-unicode() \
#("a\u{0300}" 
    .codepoints()
    .map(str.to-unicode))
```

97
 (97, 768)

character

```
str
RequiredPositional
```

The character that should be converted.

from-unicode

Converts a unicode code point into its corresponding string.

```
str.from-unicode(
int
) -> str
#str.from-unicode(97)
```

a

value

```
int
RequiredPositional
```

The code point that should be converted.

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(
    strregex
) -> bool
pattern
```

`str` or `regex`

RequiredPositional

The pattern to search for.

starts-with

Whether the string starts with the specified pattern.

```
self.startsWith(
    strregex
) -> bool
pattern
```

`str` or `regex`

RequiredPositional

The pattern the string might start with.

ends-with

Whether the string ends with the specified pattern.

```
self.endsWith(
    strregex
) -> bool
pattern
```

`str` or `regex`

RequiredPositional

The pattern the string might end with.

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(  
    strregex  
    ) -> nonestr  
pattern  
  
str or regex
```

RequiredPositional

The pattern to search for.

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(  
    strregex  
    ) -> noneint  
pattern  
  
str or regex
```

RequiredPositional

The pattern to search for.

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.

```
self.match(
    strregex
) -> nonedictionary
pattern
```

str or regex

RequiredPositional

The pattern to search for.

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.

```
self.matches(
    strregex
) -> array
pattern
```

str or regex

RequiredPositional

The pattern to search for.

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(
    strregex,
    strfunction,
    count: int,
) -> str
pattern
```

`str or regex`**RequiredPositional**

The pattern to search for.

replacement`str or function`**RequiredPositional**

The string to replace the matches with or a function that gets a dictionary for each match and can return individual replacement strings.

count`int`

If given, only the first `count` matches of the pattern are placed.

trim

Removes matches of a pattern from one or both sides of the string, once or repeatedly and returns the resulting string.

```
self.trim(  
    nonestrregex,  
    at: alignment,  
    repeat: bool,  
) -> str
```

pattern`none or str or regex`**Positional**

The pattern to search for. If `none`, trims white spaces.

Default: `none`

at`alignment`

Can be `start` or `end` to only trim the start or end of the string. If omitted, both sides are trimmed.

repeat`bool`

Whether to repeatedly removes matches of the pattern or just once. Defaults to

`true`.

Default: `true`

split

Splits a string at matches of a specified pattern and returns an array of the resulting parts.

```
self.split(  
    none str regex  
) -> array  
pattern  
  
none or str or regex
```

Positional

The pattern to split at. Defaults to whitespace.

Default: `none`

rev

Reverse the string.

```
self.rev() -> str
```

3.5.26 Style

3.5.27 System

Module for system interactions.

This module defines the following items:

- The `sys.version` constant (of type [version](#)) that specifies the currently active Typst compiler version.
- The `sys.inputs` [dictionary](#), 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](#). More complex data may be parsed manually using functions like [json.decode](#).

3.5.28 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 types known from programming](#) languages, Typst provides a special type for [content](#). A value of this type can hold anything that you can enter into your document: Text, elements like headings and shapes, and style information.

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
```

Compatibility

In Typst 0.7 and lower, the type function returned a string instead of a type.

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.

- Checks like `int == "integer"` evaluate to `true`
- Adding/joining a type and string will yield a string
- The `in` operator on a type and a dictionary will evaluate to `true` if the dictionary has a string key matching the type's name

Constructor

Determines a value's type.

```
type(
any
) -> type
#type(12) \
#type(14.7) \
#type("hello") \
#type(<glacier>) \
#type([Hi]) \
#type(x => x + 1) \
```

```
#type (ttype)

    integer
    float
    string
    label
    content
    function
    type
```

value

any

RequiredPositional

The value whose type's to determine.

3.5.29 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](#) constructor.

Constructor

Creates a new version.

It can have any number of components (even zero).

`version (`

```
..intarray
) -> version
#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
```

components

int or array

RequiredPositional

Variadic

The components of the version (array arguments are flattened)

Definitions

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
index
int
RequiredPositional
```

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

- [bibliography](#)A bibliography / reference listing.
- [cite](#)Cite a work from the bibliography.
- [document](#)The root element of a document and its metadata.
- [emph](#)Emphasizes content by toggling italics.
- [enum](#)A numbered list.
- [figure](#)A figure with an optional caption.
- [footnote](#)A footnote.
- [heading](#)A section heading.
- [link](#)Links to a URL or a location in the document.
- [list](#)A bullet list.
- [numbering](#)Applies a numbering to a sequence of numbers.
- [outline](#)A table of contents, figures, or other elements.
- [par](#)Arranges text, spacing and inline-level elements into a paragraph.
- [parbreak](#)A paragraph break.
- [quote](#)Displays a quote alongside an optional attribution.
- [ref](#)A reference to a label or bibliography.

- [`strong`](#) Strongly emphasizes content by increasing the font weight.

- [`table`](#) A table of items.

- [`terms`](#) A list of terms and their descriptions.

3.6.1 Bibliography

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 `.yml` file. Hayagriva is a new bibliography file format designed for use with Typst. Visit its [documentation](#) 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`](#) function (`#cite(<key>)`). The bibliography will only show entries for works that were referenced in the document.

Styles

Typst offers a wide selection of built-in [citation and bibliography styles](#). Beyond those, you can add and use custom [`CSL`](#) (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"

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 culinarily holistic investigation of distributed dumplings,” *Armenian Journal of Proceedings*, vol. 61, pp. 192–219, 2020.

Parameters

```
bibliography(
    strarray,
    title: noneautocontent,
```

```
full: bool,
style: str,
) -> content
path
str or array
```

RequiredPositional

Path(s) to Hayagriva .yml and/or BibLaTeX .bib files.

title

```
none or auto or content
```

Settable

The title of the bibliography.

- When set to `auto`, an appropriate title for the [text language](#) 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.")
```

Default: `auto`

full

```
bool
```

Settable

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](#) set to `none`.

Default: `false`

style

```
str
```

Settable

The bibliography style.

Should be either one of the built-in styles (see below) or a path to a [CSL file](#).

Some of the styles listed below appear twice, once with their full name and once with a short alias.

[View options](#)

Variant	Details
"alphanumeric"	Alphanumeric
"american-anthropological-association"	American Anthropological Association
"american-chemical-society"	American Chemical Society
"american-geophysical-union"	American Geophysical Union
"american-institute-of-aeronautics-and-astronautics"	American Institute of Aeronautics and Astronautics
"american-institute-of-physics"	American Institute of Physics 4th edition
"american-medical-association"	American Medical Association 11th edition
"american-meteorological-society"	American Meteorological Society
"american-physics-society"	American Physical Society
"american-physiological-society"	American Physiological Society
"american-political-science-association"	American Political Science Association

"american-psychological-association"	American Psychological Association 7th edition
"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"	American Sociological Association 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
"cell"	Cell

"chicago-author-date"	Chicago Manual of Style 17th edition (author-date)
"chicago-fullnotes"	Chicago Manual of Style 17th edition (full note)
"chicago-notes"	Chicago Manual of Style 17th edition (note)
"copernicus"	Copernicus Publications
"council-of-science-editors"	Council of Science Editors, Citation-Sequence (numeric, brackets)
"council-of-science-editors-author-date"	Council of Science Editors, Name-Year (author-date)
"current-opinion"	Current Opinion journals
"deutsche-gesellschaft-für-psychologie"	Deutsche Gesellschaft für Psychologie 5. Auflage (Deutsch)
"deutsche-sprache"	Deutsche Sprache (Deutsch)
"elsevier-harvard"	Elsevier - Harvard (with titles)
"elsevier-vancouver"	Elsevier - Vancouver
"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)
"gost-r-705-2008-numeric"	Russian GOST R 7.0.5-2008 (numeric)
"harvard-cite-them-right"	Cite Them Right 12th edition - Harvard
"institute-of-electrical-and-electronics-engineers"	IEEE
"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 - Vancouver
"modern-humanities-research-association"	Modern Humanities Research Association 4th edition (note with bibliography)
"modern-language-association"	Modern Language Association 9th edition

	Modern Language Association 8th
"modern-language-association-8"	edition
"multidisciplinary-digital-publishing-institute"	Multidisciplinary Digital Publishing Institute
"nature"	Nature
"pensoft"	Pensoft Journals
"public-library-of-science"	Public Library of Science
"royal-society-of-chemistry"	Royal Society of Chemistry
"sage-vancouver"	SAGE - 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 - Vancouver (brackets)
"taylor-and-francis-chicago-author-date"	Taylor & Francis - Chicago Manual of Style (author-date)
"taylor-and-francis-national-library-of-medicine"	Taylor & Francis - National Library of Medicine
"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"	Turabian 9th edition (author-date)
"turabian-fullnote-8"	Turabian 8th edition (full note)
"vancouver"	Vancouver
"vancouver-superscript"	Vancouver (superscript)

Default: "ieee"

3.6.2 Bullet List

Displays a sequence of items vertically, with each item introduced by a marker.

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

Syntax

This functions 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.

Parameters

```
list(
    tight: bool,
    marker: contentarrayfunction,
```

```
indent: length,
body-indent: length,
spacing: autolength,
...content,
) -> content
tight
bool
Settable
```

Defines the default [spacing](#) of the list. If it is `false`, the items are spaced apart with [paragraph spacing](#). If it is `true`, they use [paragraph leading](#) 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.

Default: `true`

[View example](#)

- 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.

marker

`content` or `array` or `function`

Settable

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.

Default: `([•], [‣], [-])`

[View example](#)

```
#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

`indent`

`length`

Settable

The indent of each item.

Default: `Opt`

`body-indent`

`length`

Settable

The spacing between the marker and the body of each item.

Default: `0.5em`

`spacing`

`auto` or `length`

Settable

The spacing between the items of the list.

If set to `auto`, uses paragraph [leading](#) for tight lists and paragraph

[spacing](#) for wide (non-tight) lists.

Default: `auto`

`children`

[content](#)

RequiredPositional***Variadic***

The bullet list's children.

When using the list syntax, adjacent items are automatically collected into lists,

even through constructs like for loops.

[View example](#)

```
#for letter in "ABC" [
    - Letter #letter
]
```

- Letter A
- Letter B
- Letter C

Definitions

itemElement

A bullet list item.

```
list.item(
    content
```

```
) -> content
```

```
body
```

```
content
```

```
RequiredPositional
```

The item's body.

3.6.3 Cite

Cite a work from the bibliography.

Before you starting citing, you need to add a [bibliography](#) somewhere in your document.

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 culinarily 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"))`.

Syntax

This function indirectly has dedicated syntax. [References](#) can be used to cite

works from the bibliography. The label then corresponds to the citation key.

Parameters

```
cite(
  label,
  supplement: nonecontent,
  form: nonestr,
  style: autostr,
) -> content
key
label
RequiredPositional
```

The citation key that identifies the entry in the bibliography that shall be cited, as a label.

[View example](#)

```
// All the same
@netwok \
#cite(<netwok>) \
#cite(label("netwok"))
```

(Astley & Morris, 2020)
(Astley & Morris, 2020)
(Astley & Morris, 2020)

supplement

none or content

Settable

A supplement for the citation such as page or chapter number.

In reference syntax, the supplement can be added in square brackets:

Default: none

[View example](#)

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.”

form

none or str

Settable

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.

Variant	Details
<code>"normal"</code>	Display in the standard way for the active style.
<code>"prose"</code>	Produces a citation that is suitable for inclusion in a sentence.
<code>"full"</code>	Mimics a bibliography entry, with full information about the cited work.
<code>"author"</code>	Shows only the cited work's author(s).
<code>"year"</code>	Shows only the cited work's year.

Default: `"normal"`

[View example](#)

```
#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.

style

`auto` or `str`

Settable

The citation style.

Should be either `auto`, one of the built-in styles (see below) or a path to a [CSL file](#). Some of the styles listed below appear twice, once with their full name and once with a short alias.

When set to `auto`, automatically use the [bibliography's style](#) for the citations.

[View options](#)

Variant	Details
<code>"alphanumeric"</code>	Alphanumeric
<code>"american-anthropological-association"</code>	American Anthropological Association
<code>"american-chemical-society"</code>	American Chemical Society
<code>"american-geophysical-union"</code>	American Geophysical Union
<code>"american-institute-of-aeronautics-and-astronautics"</code>	American Institute of Aeronautics and Astronautics
<code>"american-institute-of-physics"</code>	American Institute of Physics 4th edition
<code>"american-medical-association"</code>	American Medical Association 11th edition
<code>"american-meteorological-society"</code>	American Meteorological Society
<code>"american-physics-society"</code>	American Physical Society
<code>"american-physiological-society"</code>	American Physiological Society
<code>"american-political-science-association"</code>	American Political Science Association
<code>"american-psychological-association"</code>	American Psychological Association 7th edition

"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"	American Sociological Association
"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
"cell"	Cell
"chicago-author-date"	Chicago Manual of Style 17th edition (author-date)

"chicago-fullnotes"	Chicago Manual of Style 17th edition (full note)
"chicago-notes"	Chicago Manual of Style 17th edition (note)
"copernicus"	Copernicus Publications
"council-of-science-editors"	Council of Science Editors, Citation-Sequence (numeric, brackets)
"council-of-science-editors-author-date"	Council of Science Editors, Name-Year (author-date)
"current-opinion"	Current Opinion journals
"deutsche-gesellschaft-für-psychologie"	Deutsche Gesellschaft für Psychologie 5. Auflage (Deutsch)
"deutsche-sprache"	Deutsche Sprache (Deutsch)
"elsevier-harvard"	Elsevier - Harvard (with titles)
"elsevier-vancouver"	Elsevier - Vancouver
"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)
"gost-r-705-2008-numeric"	Russian GOST R 7.0.5-2008 (numeric)
"harvard-cite-them-right"	Cite Them Right 12th edition - Harvard
"institute-of-electrical-and-electronics-engineers"	IEEE
"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 - Vancouver
"modern-humanities-research-association"	Modern Humanities Research Association 4th edition (note with bibliography)
"modern-language-association"	Modern Language Association 9th edition
"modern-language-association-8"	Modern Language Association 8th edition

"multidisciplinary-digital-publishing-institute"	Multidisciplinary Digital Publishing Institute
"nature"	Nature
"pensoft"	Pensoft Journals
"public-library-of-science"	Public Library of Science
"royal-society-of-chemistry"	Royal Society of Chemistry
"sage-vancouver"	SAGE - 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 - Vancouver (brackets)

"taylor-and-francis-chicago-author-date"	Taylor & Francis - Chicago Manual of Style (author-date)
"taylor-and-francis-national-library-of-medicine"	Taylor & Francis - National Library of Medicine
"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"	Turabian 9th edition (author-date)
"turabian-fullnote-8"	Turabian 8th edition (full note)
"vancouver"	Vancouver
"vancouver-superscript"	Vancouver (superscript)

Default: `auto`

3.6.4 Document

The root element of a document and its metadata.

All documents are automatically wrapped in a `document` element. You cannot create a `document` element yourself. This function is only used with [set rules](#) to specify document metadata. Such a set rule must not occur inside of any layout container.

```
#set document(title: [Hello])
```

This has no visible output, but

embeds metadata into the PDF!

This has no visible output, but embeds metadata into the PDF!

Note that metadata set with this function is not rendered within the document.

Instead, it is embedded in the compiled PDF file.

Parameters

```
document (
    title: nonecontent,
    author: strarray,
    keywords: strarray,
    date: noneautodatetime,
) -> content
```

title

none or content

Settable

The document's title. This is often rendered as the title of the PDF viewer window.

While this can be arbitrary content, PDF viewers only support plain text titles, so the conversion might be lossy.

Default: none

author

str or array

Settable

The document's authors.

Default: ()

keywords

str or array

Settable

The document's keywords.

Default: ()

date

none or auto or datetime

Settable

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`.

Default: `auto`

3.6.5 Emphasis

- If the current `text style` is "normal", this turns it into "italic".
- If it is already "italic" or "oblique", it turns it back to "normal".

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.

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.

Parameters

```
emph (
  content
) -> content
body
content
RequiredPositional
```

The content to emphasize.

3.6.6 Figure

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.

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](#) 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.

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](#) 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](#) documentation for more information about breakable and non-breakable blocks.

Caption customization

You can modify the appearance of the figure's caption with its associated [caption](#) 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](#) 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

Parameters

```
figure(
content,
placement: noneautoalignment,
scope: str,
caption: nonecontent,
kind: autostrfunction,
supplement: noneautocontentfunction,
numbering: nonestrfunction,
gap: length,
outlined: bool,
) -> content
body
content
RequiredPositional
```

The content of the figure. Often, an [image](#).

placement

none or auto or alignment

Settable

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](#) argument on the `place` function.

Default: `none`

[View example](#)

```
#set page (height: 200pt)
```

```
= Introduction
#figure(
    placement: bottom,
    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 magna



Figure 1: A glacier

aliquam quaerat voluptatem. Ut enim aequa
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.

scope

str

Settable

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](#).

Variant

Details

"column" Place into the current column.

"parent" Place relative to the parent, letting the content span over all columns.

Default: "column"

[View example](#)

```
#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 magna

 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 distingue
possit, augeri
amplificarique non
possit. At.

caption

none or content

Settable

The figure's caption.

Default: `none`

`kind`

`auto` or `str` or `function`

Settable

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](#) and [code](#). In other cases, the inferred kind is that of an [image](#).

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](#), a [table](#) or [code](#),
- 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](#), [raw](#) or [image](#), you will need to manually specify the figure's supplement.

Default: `auto`

[View example](#)

```
#figure(
    circle(radius: 10pt),
    caption: [A curious atom.],
    kind: "atom",
    supplement: [Atom],
```

)



Atom 1: A curious atom.

supplement

`none` or `auto` or `content` or `function`

Settable

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](#). 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.

Default: `auto`

[View example](#)

```
#figure(
    [The contents of my figure!],
    caption: [My custom figure],
    supplement: [Bar],
    kind: "foo",
)
```

The contents of my figure!
Bar 1: My custom figure

numbering

`none` or `str` or `function`

Settable

How to number the figure. Accepts a [numbering pattern or function](#).

Default: "1"

gap
length
Settable

The vertical gap between the body and caption.

Default: 0.65em

outlined
bool
Settable

Whether the figure should appear in an [outline](#) of figures.

Default: true

Definitions

captionElement

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 `pos` and `body`, the `caption` also provides the figure's `kind`, `supplement`, `counter`, and `numbering` as fields. These parts can be used in [where](#) selectors and show rules to build a completely custom caption.

```
figure.caption(
  position: alignment,
  separator: autocontent,
  content,
) -> content
  #show figure.caption: emph

  #figure(
    rect[Hello],
    caption: [A rectangle],
  )
```



Figure 1: A rectangle

position

alignment

Settable

The caption's position in the figure. Either `top` or `bottom`.

Default: `bottom`

[View example](#)

```
#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!

separatorauto or content**Settable**

The separator which will appear between the number and body.

If set to `auto`, the separator will be adapted to the current [language](#) and [region](#).

Default: `auto`

[View example](#)

```
#set figure.caption(separator: [ --- ])
```

```
#figure(
    rect[Hello],
    caption: [A rectangle],
)
```



Figure 1 — A rectangle

bodycontent**RequiredPositional**

The caption's body.

Can be used alongside `kind`, `supplement`, `counter`, `numbering`, and `location` to completely customize the caption.

[View example](#)

```
#show figure.caption: it => [
    #underline(it.body) |
    #it.supplement
    #context it.counter.display(it.numbering)
]

#figure(
    rect[Hello],
    caption: [A rectangle],
)
```



A rectangle | Figure 1

3.6.7 Footnote

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](#). The footnote itself is realized as a normal superscript, so you can use a set rule on the [super](#) function to customize it. You can also apply a show rule to customize only the footnote marker (superscript number) in the running text.

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](#).

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](#) for more information.

Parameters

```
footnote (
    numbering: strfunction,
    labelcontent,
) -> content
numbering

str or function
```

Settable

How to number footnotes.

By default, the footnote numbering continues throughout your document. If you prefer per-page footnote numbering, you can reset the footnote [counter](#) in the page [header](#). In the future, there might be a simpler way to achieve this.

Default: "1"

[View example](#)

```
#set footnote(numbering: "*")
```

Footnotes:

```
#footnote[Star],  
#footnote[Dagger]
```

Footnotes:^{*},[†]

^{*}Star

[†]Dagger

body

[label](#) or [content](#)

RequiredPositional

The content to put into the footnote. Can also be the label of another footnote this one should point to.

Definitions

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.

```
footnote.entry(  
    content,  
    separator: content,  
    clearance: length,  
    gap: length,  
    indent: length,
```

```
) -> content
#show footnote.entry: set text (red)
```

```
My footnote listing
#footnote[It's down here]
has red text!
```

My footnote listing¹ has red text!

¹It'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.

note
content
RequiredPositional

The footnote for this entry. It's location can be used to determine the footnote counter state.

[View example](#)

```
#show footnote.entry: it => {
    let loc = it.note.location()
    numbering(
        "1: ",
        ..counter(footnote).at(loc),
    )
    it.note.body
}
```

```
Customized #footnote[Hello]
listing #footnote[World! ?]
```

Customized¹ listing²

1: Hello
2: World! 

separator

content

Settable

The separator between the document body and the footnote listing.

Default: `line(length: 30% + 0pt, stroke: 0.5pt)`

[View example](#)

```
#set footnote.entry(
    separator: repeat[.]
```

Testing a different separator.

```
#footnote[
    Unconventional, but maybe
    not that bad?
]
```

Testing a different separator.¹

¹Unconventional, but maybe not that bad?

clearance

length

Settable

The amount of clearance between the document body and the separator.

Default: `1em`

[View example](#)

```
#set footnote.entry(clearance: 3em)
```

Footnotes also need ...

```
#footnote[
```

```
    ... some space to breathe.  
]
```

Footnotes also need ...¹

¹... some space to breathe.

gap
length
Settable

The gap between footnote entries.

Default: `0.5em`

[View example](#)

```
#set footnote.entry(gap: 0.8em)
```

Footnotes:
`#footnote[Spaced],`
`#footnote[Apart]`

Footnotes:^{1, 2}

¹Spaced

²Apart

indent
length
Settable

The indent of each footnote entry.

Default: `1em`

[View example](#)

```
#set footnote.entry(indent: 0em)
```

Footnotes:
`#footnote[No],`

```
#footnote [Indent]
```

Footnotes:¹,²

¹No

²Indent

3.6.8 Heading

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).

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](#).

Independently of the numbering, Typst can also automatically generate an [outline](#) of all headings for you. To exclude one or more headings from this outline, you can set the `outlined` parameter to `false`.

Example

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

```
= Introduction
```

```
In recent years, ...
```

```
== Preliminaries
```

```
To start, ...
```

1) Introduction

In recent years, ...

1.a) Preliminaries

To start, ...

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.

Parameters

```
heading(
    level: autoint,
    depth: int,
    offset: int,
    numbering: nonestrfunction,
    supplement: noneautocontentfunction,
    outlined: bool,
    bookmarked: autobool,
    hanging-indent: autolength,
    content,
) -> content
level
```

`auto` or `int`

Settable

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](#) (either with [where](#) selectors or by accessing the `level` directly on a shown heading).

Default: `auto`

[View example](#)

```
#show heading.where(level: 2): set text(red)
```

= Level 1
== Level 2

```
#set heading(offset: 1)  

= Also level 2  

== Level 3
```

Level 1

Level 2

Also level 2

Level 3

depth
int
Settable

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.

Default: 1

offset
int
Settable

The starting offset of each heading's `level`, used to turn its relative `depth` into its absolute `level`.

Default: 0

[View example](#)

= Level 1

```
#set heading(offset: 1, numbering: "1.1")
= Level 2
```

```
#heading(offset: 2, depth: 2) [
    I'm level 4
]
```

Level 1

0.1 Level 2

0.1.0.1 I'm level 4

numbering

`none` or `str` or `function`

Settable

How to number the heading. Accepts a [numbering pattern or function](#).

Default: `none`

[View example](#)

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

```
= A section
-- A subsection
--- A sub-subsection
```

1. A section

1.a. A subsection

1.a.a. A sub-subsection

supplement

`none` or `auto` or `content` or `function`

Settable

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.

Default: `auto`

[View example](#)

```
#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.

1. Introduction

In Chapter 1, we see how to turn Sections into Chapters. And in Part 1, it is done manually.

outlined

bool

Settable

Whether the heading should appear in the [outline](#).

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.

Default: `true`

[View example](#)

```
#outline()
```

```
#heading [Normal]
```

This is a normal heading.

```
#heading(outlined: false) [Hidden]
This heading does not appear
in the outline.
```

Contents

Normal	1
--------------	---

Normal

This is a normal heading.

Hidden

This heading does not appear in the outline.

bookmarked

 auto or  bool

Settable

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](#). Setting this property to either `true` (bookmark) or `false` (don't bookmark) bypasses that behavior.

Default: `auto`

[View example](#)

```
#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.
```

Normal heading

This heading will be shown in the PDF's bookmark outline.

Not bookmarked

This heading won't be bookmarked in the resulting PDF.

hanging-indent

`auto` or `length`

Settable

The indent all but the first line of a heading should have.

The default value of `auto` indicates that the subsequent heading lines will be indented based on the width of the numbering.

Default: `auto`

[View example](#)

```
#set heading(numbering: "1.")
#heading[A very, very, very, very, very, very long heading]
```

1. A very, very, very, very, very, very long heading

body

`content`

RequiredPositional

The heading's title.

3.6.9 Link

Links to a URL or a location in the document.

By default, links are not styled any different from normal text. However, you can easily apply a style of your choice with a `show` rule.

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

Syntax

This function also has dedicated syntax: Text that starts with `http://` or `https://` is automatically turned into a link.

Parameters

```
link(
    str|label|location|dictionary,
    content,
) -> content
dest
    str or label or location or dictionary
```

RequiredPositional

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` attached to an element. If you also want automatic text for the link based on the element, consider using a `reference` instead.
- A `location` (typically retrieved from `here`, `locate` or `query`).
- A dictionary with a `page` key of type `integer` and `x` and `y` coordinates of type `length`. Pages are counted from one, and the coordinates are relative to the page's top left corner.

[View example](#)

```
= 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

`body`

`content`

RequiredPositional

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.

3.6.10 Numbered List

3.6.11 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 repeated as-is.

Example

```
#numbering("1.1)", 1, 2, 3) \
#numbering("1.a.i", 1, 2) \
#numbering("I - 1", 12, 2) \

    (..nums) => nums
    .pos()
    .map(str)
    .join("." + ")",
        1, 2, 3,
    )

```

1.2.3)
1.b
XII – 2
1.2.3)

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](#) or [figures](#). Every time, the expected format is the same as the one described below for the [numbering](#) parameter.

The following example illustrates that a numbering function is just a regular [function](#) that accepts numbers and returns [content](#).

```
#let unary(..., last) = "|" * last
#set heading(numbering: unary)
= First heading
= Second heading
= Third heading
```

| First heading

|| Second heading

||| Third heading

Parameters

```
numbering(
strfunction,
..int,
) -> any
numbering
```

str or function

RequiredPositional

Defines how the numbering works.

Counting symbols are 1, a, A, i, I, 一, 壱, あ, い, ア, イ, も, ガ, 一, *, ①, and

①. 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 repeated as-is at the end of any rendered number.

Prefixes are all characters that are neither counting symbols nor suffixes. They are repeated 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.

numbers

int

RequiredPositional

Variadic

The numbers to apply the numbering to. Must be positive.

If `numbering` is a pattern and more numbers than counting symbols are given, the last counting symbol with its prefix is repeated.

3.6.12 Outline

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. The element's numbering and page number will be displayed in the outline alongside its title or caption. By default this generates a table of contents.

Example

```
#outline()
```

= Introduction

#lorem(5)

= Prior work

#lorem(10)

Contents

Introduction	1
Prior work	1

Introduction

Lorem ipsum dolor sit amet.

Prior work

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Alternative outlines

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)`. We could have also set it to just `figure`, but then the list would also include figures containing tables or other material. For more details on the `where` selector, [see here](#).

```
#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!

Styling the outline

The outline element has several options for customization, such as its `title` and `indent` parameters. If desired, however, it is possible to have more control over the outline's look and style through the [`outline.entry`](#) element.

Parameters

```
outline(
    title: noneautocontent,
    target: labelselectorlocationfunction,
    depth: noneint,
    indent: noneautoboolrelativefunction,
    fill: nonecontent,
) -> content
title
none or auto or content
```

Settable

The title of the outline.

- When set to `auto`, an appropriate title for the [`text language`](#) will be used. This is the default.
- 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.")`

Default: `auto`

target

label or selector or location or function

Settable

The type of element to include in the outline.

To list figures containing a specific kind of element, like a table, you can write

`figure.where(kind: table).`

Default: `heading.where(outlined: true)`

View example

```
#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

depth`none` or `int`**Settable**

The maximum level up to which elements are included in the outline. When this argument is `none`, all elements are included.

Default: `none`

[View example](#)

```
#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.

indent`none` or `auto` or `bool` or `relative` or `function`**Settable**

How to indent the outline's entries.

- `none`: No indent
- `auto`: Indents the numbering of the nested entry with the title of its parent entry. This only has an effect if the entries are numbered (e.g., via [heading](#) [numbering](#)).
- [Relative length](#): Indents the item by this length multiplied by its nesting level. Specifying `2em`, for instance, would indent top-level headings (not nested) by `0em`, second level headings by `2em` (nested once), third-level headings by `4em` (nested twice) and so on.
- [Function](#): You can completely customize this setting with a function. That function receives the nesting level as a parameter (starting at 0 for top-level headings/elements) and can return a relative length or content making up the indent. For example, `n => n * 2em` would be equivalent to just specifying `2em`, while `n => [→] * n` would indent with one arrow per nesting level.

Migration hints: Specifying `true` (equivalent to `auto`) or `false` (equivalent to `none`) for this option is deprecated and will be removed in a future release.

Default: `none`

[View example](#)

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

#outline(
    title: [Contents (Automatic)],
    indent: auto,
)

#outline(
    title: [Contents (Length)],
    indent: 2em,
)
```

```
#outline (
    title: [Contents (Function)],
    indent: n => [→] * n,
)

= About ACME Corp.
== History
==== Origins
#lorem(10)

== Products
#lorem(10)
```

Contents (Automatic)

1. About ACME Corp.	1
1.a. History	1
1.a.a. Origins	1
1.b. Products	1

Contents (Length)

1. About ACME Corp.	1
1.a. History	1
1.a.a. Origins	1
1.b. Products	1

Contents (Function)

1. About ACME Corp.	1
→ 1.a. History	1
→ → 1.a.a. Origins	1
→ 1.b. Products	1

1. About ACME Corp.

1.a. History

1.a.a. Origins

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

1.b. Products

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

fill

`none` or `content`

Settable

Content to fill the space between the title and the page number. Can be set

to `none` to disable filling.

Default: `repeat`(body: [.])

[View example](#)

```
#outline(fill: line(length: 100%))
```

= A New Beginning

Contents

A New Beginning	1
-----------------	---

A New Beginning

Definitions

entryElement

Represents each entry line in an outline, including the reference to the outlined element, its page number, and the filler content between both.

This element is intended for use with show rules to control the appearance of outlines. To customize an entry's line, you can build it from scratch by accessing the `level`, `element`, `body`, `fill` and `page` fields on the entry.

```
outline.entry(
    int,
    content,
    content,
    nonecontent,
    content,
) -> content
    #set heading(numbering: "1.")

    #show outline.entry.where(
        level: 1
    ): it => {
        v(12pt, weak: true)
        strong(it)
    }

#outline(indent: auto)
```

= Introduction

```
= Background
== History
== State of the Art
= Analysis
== Setup
```

Contents

1. Introduction	1
2. Background	1
2.1. History	1
2.2. State of the Art	1
3. Analysis	1
3.1. Setup	1

1. Introduction

2. Background

2.1. History

2.2. State of the Art

3. Analysis

3.1. Setup

level

`int`

RequiredPositional

The nesting level of this outline entry. Starts at `1` for top-level entries.

element

`content`

RequiredPositional

The element this entry refers to. Its location will be available through the

[location](#) method on content and can be [linked](#) to.

body

`content`

RequiredPositional

The content which is displayed in place of the referred element at its entry in the outline. For a heading, this would be its number followed by the heading's title, for example.

fill

none or content

RequiredPositional

The content used to fill the space between the element's outline and its page number, as defined by the outline element this entry is located in.

When none, empty space is inserted in that gap instead.

Note that, when using show rules to override outline entries, it is recommended to wrap the filling content in a box with fractional width. For example,

`box(width: 1fr, repeat[-])` would show precisely as many – characters as necessary to fill a particular gap.

page

content

RequiredPositional

The page number of the element this entry links to, formatted with the numbering set for the referenced page.

3.6.13 Paragraph

Arranges text, spacing and inline-level elements into a paragraph.

Although this function is primarily used in set rules to affect paragraph properties, it can also be used to explicitly render its argument onto a paragraph of its own.

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
 $\$c$ 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 ...$

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 small-
 est of the three integers. Then, we ...

Parameters

```
par(
    leading: length,
    spacing: length,
    justify: bool,
    linebreaks: autostr,
    first-line-indent: length,
    hanging-indent: length,
    content,
) -> content
leading
length
Settable
```

The spacing between lines.

Leading defines the spacing between the [bottom edge](#) of one line and the [top edge](#) 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.

Default: `0.65em`

spacing

length

Settable

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.

When a paragraph is adjacent to a [block](#) that is not a paragraph, that block's [above](#) or [below](#) property takes precedence over the paragraph spacing.

Headings, for instance, reduce the spacing below them by default for a better look.

Default: `1.2em`

justify

bool

Settable

Whether to justify text in its line.

Hyphenation will be enabled for justified paragraphs if the [text function's](#)

[hyphenate property](#) is set to `auto` and the current language is known.

Note that the current [alignment](#) still has an effect on the placement of the last line except if it ends with a [justified line break](#).

Default: `false`

linebreaks

`auto` or `str`

Settable

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.

Variant

Details

`"simple"` Determine the line breaks in a simple first-fit style.

Optimize the line breaks for the whole paragraph.

`"optimized"` Typst will try to produce more evenly filled lines of text by

considering the whole paragraph when calculating line breaks.

Default: `auto`

[View example](#)

```
#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.

first-line-indent

length

Settable

The indent the first line of a paragraph should have.

Only the first line of a consecutive paragraph will be indented (not the first one in a block or on the page).

By typographic convention, paragraph breaks are indicated either by some space between paragraphs or by indented first lines. Consider reducing the

[paragraph spacing](#) to the [leading](#) when using this property (e.g. using

```
#set par(spacing: 0.65em).
```

Default: `0pt`

hanging-indent

length

Settable

The indent all but the first line of a paragraph should have.

Default: `0pt`

```
body
content
RequiredPositional
```

The contents of the paragraph.

Definitions

```
lineElement
```

A paragraph line.

This element is exclusively used for line number configuration through set rules and cannot be placed.

The [numbering](#) option is used to enable line numbers by specifying a numbering format.

```
par.line(
    numbering: nonestrfunction,
    number-align: autoalignment,
    number-margin: alignment,
    number-clearance: autolength,
    numbering-scope: str,
) -> content
#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](#) 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](#) or [margin](#). In addition, you can

control whether the numbering is reset on each page through the [numbering-scope](#) option.

numbering

[none](#) or [str](#) or [function](#)

Settable

How to number each line. Accepts a [numbering pattern or function](#).

Default: [none](#)

View example

```
#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.

number-align

[auto](#) or [alignment](#)

Settable

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.

Default: [auto](#)

View example

```
#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.

number-marginalignment***Settable***

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.

Default: `start`

[View example](#)

```
#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

number-clearance**auto** or **length***Settable*

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.

Default: **auto**

[View example](#)

```
#set par.line(
    numbering: "1",
    number-clearance: 4pt,
)
```

Typesetting \

Styling \

Layout

- 1 Typesetting
- 2 Styling
- 3 Layout

numbering-scope**str***Settable*

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.

Variant	Details
<pre>"document"</pre>	Indicates the line number counter spans the whole document, that is, is never automatically reset.
<pre>"page"</pre>	Indicates the line number counter should be reset at the start of every new page.

Default: `"document"`

[View example](#)

```
#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

3.6.14 Paragraph Break

A paragraph break.

This starts a new paragraph. Especially useful when used within code like [for loops](#). Multiple consecutive paragraph breaks collapse into a single one.

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.

Syntax

Instead of calling this function, you can insert a blank line into your markup to create a paragraph break.

Parameters

`parbreak() -> content`

3.6.15 Quote

Displays a quote alongside an optional attribution.

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 γοῦν τούτου γε σμικρῷ τινι αὔτῳ τούτῳ σοφῶτερος
    εἶναι, ὅτι
```

```

ἀ 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 from his original quote:

... ἔστια γοῦν τούτου γε σμικρῷ τινὶ¹
 αὐτῷ τούτῳ σοφώτερος εἶναι, ὅτι ἀ μὴ²
 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.  
]
```

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.

Parameters

```
quote(  

  block: bool,  

  quotes: autobool,  

  attribution: none|label|content,  

  content,  

) -> content  

block  

bool  

Settable
```

Whether this is a block quote.

Default: `false`

[View example](#)

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

quotes

`auto` or `bool`

Settable

Whether double quotes should be added around this quote.

The double quotes used are inferred from the `quotes` property on [smartquote](#), which is affected by the `lang` property on [text](#).

- `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.

Default: `auto`

[View example](#)

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

```
Ein deutsch-sprechender Author
zitiert unter umständen JFK:
[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.”

attribution

none or **label** or **content**

Settable

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.

Default: **none**

[View example](#)

```
#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) ]
}



```

of Udûn. Go back to the Shadow! You
cannot pass.

]

```
#bibliography("works.bib", style: "apa")
```

“cogito, ergo sum”

“Compose papers faster” (typst.com)

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*: 1 (Vol. 1). Allen & Unwin.

body

content

RequiredPositional

The quote.

3.6.16 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 references are also links to the respective element. Reference syntax can also be used to [cite](#) from a bibliography.

Referenceable elements include [headings](#), [figures](#), [equations](#), and [footnotes](#). To create a custom referenceable element like a theorem, you can create a figure

of a custom [kind](#) 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](#) function instead.

Example

```
#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, 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")
```

1. Introduction

Recent developments in typesetting software have rekindled hope in previously frustrated researchers. [1] As shown in Section 2, we ...

2. Results

We discuss our approach in comparison with others.

2.1. Performance

Equation 1 demonstrates what slow software looks like.

$$T(n) = O(2^n) \quad (1)$$

Bibliography

- [1] B. Aldrin, “An Insight into Bibliographical Distress.”

Syntax

This function also has dedicated syntax: A 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].

Customization

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
    if el != none and el.func() == eq {
        // Override equation references.
        link(el.location(), numbering(
            el.numbering,
            ..counter(eq).at(el.location())))
    })
} else {
    // Other references as usual.
    it
}
}

= Beginnings <beginning>
In @beginning we prove @pythagoras.
$ a^2 + b^2 = c^2 $ <pythagoras>
```

1. Beginnings

In Section 1 we prove (1).

$$a^2 + b^2 = c^2 \tag{1}$$

Parameters

```
ref(
label,
supplement: noneautocontentfunction,
) -> content
target
label
RequiredPositional
```

The target label that should be referenced.

Can be a label that is defined in the document or an entry from the

[bibliography](#).

supplement

`none` or `auto` or `content` or `function`

Settable

A supplement for the reference.

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 a function is specified, it is passed the referenced element and should return content.

Default: `auto`

View example

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

1. Introduction

In Chapter 1, we see how to turn Sections into Chapters. And in Part 1, it is done manually.

3.6.17 Strong Emphasis

Strongly emphasizes content by increasing the font weight.

Increases the current font weight by a given `delta`.

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**.

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.

Parameters

```
strong(
  delta: int,
  content,
) -> content
delta
int
Settable
```

The `delta` to apply on the font weight.

Default: 300

[View example](#)

```
#set strong(delta: 0)
No *effect!*
```

No effect!

```
body
content
RequiredPositional
```

The content to strongly emphasize.

3.6.18 Table

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](#).

Because tables are just grids with different defaults for some cell properties (notably `stroke` and `inset`), refer to the [grid documentation](#) 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, Typst will annotate its output in the future such that screenreaders

will announce content in `table` as tabular while a grid's content will be announced no different than multiple content blocks in the document flow.

Note that, to override a particular cell's properties or apply show rules on table cells, you can use the [`table.cell`](#) 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.

To give a table a caption and make it [referenceable](#), put it into a [`figure`](#).

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` 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 => {
    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,
)
```

	Exam 1	Exam 2	Exam 3
John	N/A	A	N/A
Mary	N/A	A	A
Robert	B	A	B

Parameters

```
table(
    columns: autointrelativefractionarray,
    rows: autointrelativefractionarray,
    gutter: autointrelativefractionarray,
    column-gutter: autointrelativefractionarray,
    row-gutter: autointrelativefractionarray,
    fill: nonecolorgradientarraypatternfunction,
    align: autoarrayalignmentfunction,
    stroke: nonelengthcolorgradientarraystrokepatterndictionfunction,
    inset: relativearraydictionaryfunction,
    ..content,
) -> content
columns
auto or int or relative or fraction or array
```

Settable

The column sizes. See the [grid documentation](#) for more information on track sizing.

Default: ()

rows

auto or int or relative or fraction or array

Settable

The row sizes. See the [grid documentation](#) for more information on track sizing.

Default: ()

gutter

`auto` or `int` or `relative` or `fraction` or `array`

Settable

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](#) for more information on gutters.

Default: ()

column-gutter

`auto` or `int` or `relative` or `fraction` or `array`

Settable

The gaps between columns. Takes precedence over `gutter`. See the [grid documentation](#) for more information on gutters.

Default: ()

row-gutter

`auto` or `int` or `relative` or `fraction` or `array`

Settable

The gaps between rows. Takes precedence over `gutter`. See the [grid documentation](#) for more information on gutters.

Default: ()

fill

`none` or `color` or `gradient` or `array` or `pattern` or `function`

Settable

How to fill the cells.

This can be a color or a function that returns a color. The function receives the cells' column and row indices, starting from zero. This can be used to implement striped tables.

Default: `none`

[View example](#)

```
#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 €

align

`auto` or `array` or `alignment` or `function`

Settable

How to align the cells' content.

This can either be a single alignment, an array of alignments (corresponding to each column) or a function that returns an alignment. The function receives the

cells' column and row indices, starting from zero. If set to `auto`, the outer alignment is used.

Default: `auto`

[View example](#)

```
#table(
    columns: 3,
    align: (left, center, right),
    [Hello], [Hello], [Hello],
    [A], [B], [C],
)
```

Hello	Hello	Hello
A	B	C

stroke

`none` or `length` or `color` or `gradient` or `array` or `stroke` or `pattern` or `dictionary`
`y` or `function`

Settable

How to [stroke](#) 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` option, or to override the stroke between multiple specific cells, consider specifying one or more of [table.hline](#) and [table.vline](#) alongside your table cells.

See the [grid documentation](#) for more information on strokes.

Default: `1pt + black`

inset

`relative` or `array` or `dictionary` or `function`

Settable

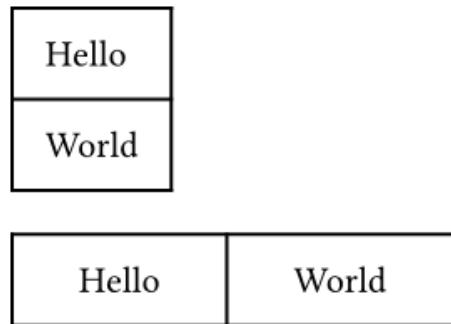
How much to pad the cells' content.

Default: `0% + 5pt`

[View example](#)

```
#table(
    inset: 10pt,
    [Hello],
    [World],
)

#table(
    columns: 2,
    inset: (
        x: 20pt,
        y: 10pt,
    ),
    [Hello],
    [World],
)
```

**children****content*****RequiredPositional******Variadic***

The contents of the table cells, plus any extra table lines specified with the [table.hline](#) and [table.vline](#) elements.

Definitions

cellElement

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.

```
table.cell(
    content,
    x: autoint,
    y: autoint,
    colspan: int,
    rowspan: int,
    fill: noneautocolorgradientpattern,
    align: autoalignment,
    inset: autorelativedictionary,
    stroke: nonelengthcolorgradientstrokepatterndictionary,
    breakable: autobool,
) -> content
    #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

body
content
RequiredPositional

The cell's body.

x
auto or int
Settable

The cell's column (zero-indexed). Functions identically to the `x` field in

[grid.cell](#).

Default: `auto`

y

`auto` or `int`

Settable

The cell's row (zero-indexed). Functions identically to the `y` field in

[grid.cell](#).

Default: `auto`

colspan

`int`

Settable

The amount of columns spanned by this cell.

Default: `1`

rowspan

`int`

Settable

The amount of rows spanned by this cell.

Default: `1`

fill

`none` or `auto` or `color` or `gradient` or `pattern`

Settable

The cell's `fill` override.

Default: `auto`

align

`auto` or `alignment`

Settable

The cell's `alignment` override.

Default: `auto`

inset

`auto` or `relative` or `dictionary`

Settable

The cell's [inset](#) override.

Default: `auto`

stroke

`none` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

Settable

The cell's [stroke](#) override.

Default: `(::)`

breakable

`auto` or `bool`

Settable

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.

Default: `auto`

hlineElement

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](#) 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](#) field or [table.cell's stroke](#) field instead if the line you want to place is part of all your tables' designs.

```
table.hline(
    y: autoint,
    start: int,
    end: noneint,
    stroke: nonelengthcolorgradientstrokepatterndictionary,
    position: alignment,
) -> content

#set table.hline(stroke: .6pt)



```

09:00	Badge pick up
09:45	Opening Keynote
10:30	Talk: Typst's Future
11:15	Session: Good PRs
Noon	<i>Lunch break</i>
14:00	Talk: Tracked Layout
15:00	Talk: Automations
16:00	Workshop: Tables
19:00	Day 1 Attendee Mixer

y**auto** or **int****Settable**

The row above which the horizontal line is placed (zero-indexed). Functions identically to the `y` field in [grid.hline](#).

Default: `auto`**start****int****Settable**

The column at which the horizontal line starts (zero-indexed, inclusive).

Default: `0`**end****none** or **int****Settable**

The column before which the horizontal line ends (zero-indexed, exclusive).

Default: `none`**stroke****none** or **length** or **color** or **gradient** or **stroke** or **pattern** or **dictionary**

Settable

The line's stroke.

Specifying `none` removes any lines previously placed across this line's range, including `hlines` or per-cell stroke below it.

Default: `1pt + black`

position**alignment*****Settable***

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.

Default: `top`

vlineElement

A vertical line in the table. See the docs for [grid.vline](#) 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`](#) option.

Similar to [`table.hline`](#), 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`](#) field or [`table.cell's stroke`](#) field instead if the line you want to place is part of all your tables' designs.

```
table.vline(
    x: autoint,
    start: int,
    end: noneint,
    stroke: nonelengthcolorgradientstrokepatterndictionary,
    position: alignment,
) -> content
x
auto or int
```

Settable

The column before which the horizontal line is placed (zero-indexed).

Functions identically to the `x` field in [grid.vline](#).

Default: `auto`

start

`int`

Settable

The row at which the vertical line starts (zero-indexed, inclusive).

Default: `0`

end

`none or int`

Settable

The row on top of which the vertical line ends (zero-indexed, exclusive).

Default: `none`

stroke

`none or length or color or gradient or stroke or pattern or dictionary`

Settable

The line's stroke.

Specifying `none` removes any lines previously placed across this line's range, including vlines or per-cell stroke below it.

Default: `1pt + black`

positionalignment**Settable**

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.

Default: `start`

headerElement

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 will use this function to attach accessibility metadata to tables in the future and ensure universal access to your document.

You can use the `repeat` parameter to control whether your table's header will be repeated across pages.

```
table.header(
  repeat: bool,
  ..content,
) -> content
  #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


```

	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	

	Blue chip	Fresh IPO	Penny st'k
	0.01	195	4
7.34	57	2	

repeatbool**Settable**

Whether this header should be repeated across pages.

Default: `true`

childrencontent**RequiredPositional****Variadic**

The cells and lines within the header.

footerElement

A repeatable table footer.

Just like the [table.header](#) 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
repeat
bool
Settable
```

Whether this footer should be repeated across pages.

Default: `true`

```
children
content
RequiredPositional
Variadic
```

The cells and lines within the footer.

3.6.19 Term List

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.

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.

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.

Parameters

```
terms (
    tight: bool,
    separator: content,
    indent: length,
    hanging-indent: length,
    spacing: autolength,
    ...contentarray,
) -> content
```

tight
bool
Settable

Defines the default spacing of the term list. If it is false, the items are spaced apart with paragraph_spacing. If it is true, they use paragraph_leading 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.

Default: true

[View example](#)

/ **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.

separator**content****Settable**

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.

Default: `h(amount: 0.6em, weak: true)`

[View example](#)

```
#set terms(separator: [:])
```

/ **Colon:** A nice separator symbol.

Colon: A nice separator symbol.

indent**length****Settable**

The indentation of each item.

Default: `0pt`

hanging-indent

length

Settable

The hanging indent of the description.

This is in addition to the whole item's `indent`.

Default: `2em`

[View example](#)

```
#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.

spacing

auto or length

Settable

The spacing between the items of the term list.

If set to `auto`, uses paragraph [`leading`](#) for tight term lists and paragraph

[`spacing`](#) for wide (non-tight) term lists.

Default: `auto`

children

content or array

RequiredPositional

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.

[View example](#)

```
#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.

Definitions

item*Element*

A term list item.

```
terms.item(
    content,
    content,
) -> content
```

term

```
content
```

RequiredPositional

The term described by the list item.

description

```
content
```

RequiredPositional

The description of the term.

3.7 Text

Text styling.

The [text function](#) is of particular interest.

Definitions

- [highlight](#) Highlights text with a background color.
- [linebreak](#) Inserts a line break.
- [lorem](#) Creates blind text.
- [lower](#) Converts a string or content to lowercase.
- [overline](#) Adds a line over text.
- [raw](#) Raw text with optional syntax highlighting.
- [smallcaps](#) Displays text in small capitals.
- [smartquote](#) A language-aware quote that reacts to its context.
- [strike](#) Strikes through text.
- [sub](#) Renders text in subscript.
- [super](#) Renders text in superscript.
- [text](#) Customizes the look and layout of text in a variety of ways.
- [underline](#) Underlines text.
- [upper](#) Converts a string or content to uppercase.

3.7.1 Highlight

Highlights text with a background color.

Example

This is `#highlight[important]`.

This is **important**.

Parameters

```
highlight(
  fill: none|color|gradient|pattern,
```

```
stroke: none length color gradient stroke patterndictionary,
top-edge: lengthstr,
bottom-edge: lengthstr,
extent: length,
radius: relativedictionary,
content,
) -> content
fill

none or color or gradient or pattern
```

Settable

The color to highlight the text with.

Default: `rgb ("#ffffd11a1")`

[View example](#)

```
This is #highlight (
    fill: blue
) [highlighted with blue].
```

This is highlighted with blue.

stroke

```
none or length or color or gradient or stroke or pattern or dictionary
```

Settable

The highlight's border color. See the [rectangle's documentation](#) for more details.

Default: `(:)`

[View example](#)

```
This is a #highlight (
    stroke: fuchsia
) [stroked highlighting].
```

This is a stroked highlighting.

top-edge

length or str

Settable

The top end of the background rectangle.

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.

Default: "ascender"

[View example](#)

```
#set highlight (top-edge: "ascender")
highlight[a] #highlight[aib]
```

```
#set highlight (top-edge: "x-height")
highlight[a] #highlight[aib]
```




bottom-edge

length or str

Settable

The bottom end of the background rectangle.

Variant	Details
"baseline"	The baseline on which the letters rest.

The font's descender, which typically exceeds the depth of all "descender" glyphs.

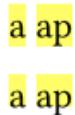
"bounds" The bottom edge of the glyph's bounding box.

Default: "descender"

[View example](#)

```
#set highlight(bottom-edge: "descender")
highlight[a] #highlight[ap]
```

```
#set highlight(bottom-edge: "baseline")
highlight[a] #highlight[ap]
```



extent

length

Settable

The amount by which to extend the background to the sides beyond (or within if negative) the content.

Default: `Opt`

[View example](#)

```
A long #highlight(extent: 4pt) [background].
```

radius

relative or dictionary

Settable

How much to round the highlight's corners. See the [rectangle's documentation](#) for more details.

Default: (:

[View example](#)

```
Listen #highlight(
    radius: 5pt, extent: 2pt
) [carefully], it will be on the test.
```

Listen **carefully**, it will be on the test.

body

content

RequiredPositional

The content that should be highlighted.

3.7.2 Line Break

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.

Example

```
*Date:* 26.12.2022 \
*Topic:* Infrastructure Test \
*Severity:* High \
```

Date: 26.12.2022
Topic: Infrastructure Test
Severity: High

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.

Parameters

```
linebreak(
justify: bool
) -> content
justify
bool
Settable
```

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.

Default: `false`

[View example](#)

```
#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 *Lo&em; 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.

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 magna
 aliquam quaerat voluptatem. Ut enim aequa
 doleamus animo, cum corpose dolemus, fieri.

More Blind Text

 Lorem ipsum dolor sit amet, consectetur
 adipiscing elit, sed do eiusmod tempor
 incididunt ut labore.

Parameters

```
lorem(
int
) -> str
words
int
RequiredPositional
```

The length of the blind text in words.

3.7.4 Lowercase

Converts a string or content to lowercase.

Example

```
#lower("ABC") \
#lower[*My Text*] \
#lower[already low]
```

```
abc
my text
already low
```

Parameters

```
lower(
  strcontent
) -> strcontent
```

text

```
str or content
```

RequiredPositional

The text to convert to lowercase.

3.7.5 Overline

Adds a line over text.

Example

```
#overline[A line over text.]
```

Parameters

```
overline(
  stroke: autolengthcolorgradientstrokepatterndictionary,
  offset: autolength,
  extent: length,
  evade: bool,
  background: bool,
  content,
) -> content
```

stroke

```
auto or length or color or gradient or stroke or pattern or dictionary
```

Settable

How to [stroke](#) the line.

If set to `auto`, takes on the text's color and a thickness defined in the current font.

Default: `auto`

[View example](#)

```
#set text(fill: olive)
#overline(
    stroke: green.darken(20%) ,
    offset: -12pt,
    [The Forest Theme] ,
)
```

The Forest Theme

offset

`auto` or `length`

Settable

The position of the line relative to the baseline. Read from the font tables if `auto`.

Default: `auto`

[View example](#)

```
#overline(offset: -1.2em) [
    The Tale Of A Faraway Line II
]
```

The Tale Of A Faraway Line II

extent

`length`

Settable

The amount by which to extend the line beyond (or within if negative) the content.

Default: `0pt`

[View example](#)

```
#set overline(extent: 4pt)
#set underline(extent: 4pt)
#overline(underline[Typography Today])
```

evade

`bool`

Settable

Whether the line skips sections in which it would collide with the glyphs.

Default: `true`

[View example](#)

```
#overline(
    evade: false,
    offset: -7.5pt,
    stroke: 1pt,
    extent: 3pt,
    [Temple],
)
```

background

`bool`

Settable

Whether the line is placed behind the content it overlines.

Default: `false`

[View example](#)

```
#set overline(stroke: (thickness: 1em, paint: maroon, cap: "round"))
#overline(background: true) [This is stylized.] \
#overline(background: false) [This is partially hidden.]
```



body
content
RequiredPositional

The content to add a line over.

3.7.6 Raw Text / Code

Raw text with optional syntax highlighting.

Displays the text verbatim and in a monospace font. This is typically used to embed computer code into your document.

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.

```
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` element programmatically from a string (and provide the language tag via the optional `lang` argument).

```
#raw("fn " + "main() {}", lang: "rust")
```

```
fn main() {}
```

Syntax

This function also has dedicated syntax. You can enclose text in 1 or 3+ backticks (`) to make it raw. Two backticks produce empty raw text. This works both in markup and code.

When you use three or more backticks, you can additionally specify a language tag for syntax highlighting directly after the opening backticks. Within raw blocks, everything (except for the language tag, if applicable) is rendered as is, in particular, there are no escape sequences.

The language tag is an identifier that directly follows the opening backticks only if there are three or more backticks. If your text starts with something that looks

like an identifier, but no syntax highlighting is needed, start the text with a single space (which will be trimmed) or use the single backtick syntax. If your text should start or end with a backtick, put a space before or after it (it will be trimmed).

Parameters

```
raw(
    str,
    block: bool,
    lang: nonestr,
    align: alignment,
    syntaxes: strarray,
    theme: noneautostr,
    tab-size: int,
) -> content
text
str
RequiredPositional
```

The raw text.

You can also use raw blocks creatively to create custom syntaxes for your automations.

[View example](#)

```
// 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

blockbool*Settable*

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.

Default: `false`

[View example](#)

```
// 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 :`

```
rg "Hello World"
```

`lang`

`none` or `str`

Settable

The language to syntax-highlight in.

Apart from typical language tags known from Markdown, this supports the

`"typ"`, `"typc"`, and `"typm"` tags for [Typst markup](#), [Typst code](#), and [Typst math](#), respectively.

Default: `none`

[View example](#)

```
```typ
This is *Typst!*
```

```

This is ````typ` also `*Typst!````, but inline!

This is `*Typst!*`

This is also `*Typst*`, but inline!

`align`

[alignment](#)

Settable

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).

Default: `start`

[View example](#)

```
#set raw(align: center)
```

```
```typc
let f(x) = x
code = "centered"
```

```

```
let f(x) = x
code = "centered"
```

syntaxes

[str or array](#)

Settable

One or multiple additional syntax definitions to load. The syntax definitions should be in the [sublime-syntax file format](#).

Default: `()`

[View example](#)

```
#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)))))
```

**theme**

[none](#) or [auto](#) or [str](#)

*Settable*

The theme to use for syntax highlighting. Theme files should be in the

[tmTheme file format](#).

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](#) function and the background with a [filled block](#). You could also use the [xml](#) function to extract these properties from the theme. Additionally, you can set the theme to [none](#) to disable highlighting.

Default: [auto](#)

View example

```
#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"
```

**tab-size**int*Settable*

The size for a tab stop in spaces. A tab is replaced with enough spaces to align with the next multiple of the size.

Default: 2

View example

#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

## Definitions

**lineElement**

A highlighted line of raw text.

This is a helper element that is synthesized by [raw](#) 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
number
int
RequiredPositional
```

The line number of the raw line inside of the raw block, starts at 1.

```
count
int
RequiredPositional
```

The total number of lines in the raw block.

```
text
str
RequiredPositional
```

The line of raw text.

```
body
content
RequiredPositional
```

The highlighted raw text.

## 3.7.7 Small Capitals

Displays text in small capitals.

### Example

```
Hello \
#smallcaps[Hello]
```

```
Hello
HELLO
```

## Smallcaps fonts

By default, this enables the OpenType `smcp` feature for the font. Not all fonts support this feature. 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.

## Smallcaps headings

You can use a [show rule](#) 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 magna aliquam quaerat voluptatem. Ut enim aequo doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere.

## Parameters

```
smallcaps (
 content
) -> content
body
content
RequiredPositional
```

The content to display in small capitals.

## 3.7.8 Smartquote

A language-aware quote that reacts to its context.

Automatically turns into an appropriate opening or closing quote based on the active [text language](#).

## 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. »

## Syntax

This function also has dedicated syntax: The normal quote characters (' and ").

Typst automatically makes your quotes smart.

## Parameters

```
smartquote(
 double: bool,
 enabled: bool,
 alternative: bool,
 quotes: autstrarraydictionary,
) -> content
double
bool
Settable
```

Whether this should be a double quote.

Default: `true`

```
enabled
bool
Settable
```

Whether smart quotes are enabled.

To disable smartness for a single quote, you can also escape it with a backslash.

Default: `true`

[View example](#)

```
#set smartquote(enabled: false)
```

These are "dumb" quotes.

These are "dumb" quotes.

### **alternative**

bool

*Settable*

Whether to use alternative quotes.

Does nothing for languages that don't have alternative quotes, or if explicit quotes were set.

Default: `false`

[View example](#)

```
#set text(lang: "de")
#set smartquote(alternative: true)
```

"Das ist in anderen Anführungszeichen."

»Das ist in anderen Anführungszeichen.«

### **quotes**

auto or str or array or dictionary

*Settable*

The quotes to use.

- When set to `auto`, the appropriate single quotes for the [text language](#) will be used. This is the default.
- Custom quotes can be passed as a string, array, or dictionary of either
  - [string](#): a string consisting of two characters containing the opening and closing double quotes (characters here refer to Unicode grapheme clusters)
  - [array](#): an array containing the opening and closing double quotes

o [dictionary](#): an array containing the double and single quotes, each specified as

either `auto`, string, or array

Default: `auto`

[View example](#)

```
#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 Strikethrough

Strikes through text.

#### Example

This is `#strike[not]` relevant.

This is `not` relevant.

#### Parameters

```
strike(
stroke: autolengthcolorgradientstrokepatterndictionary,
offset: autolength,
extent: length,
background: bool,
content,
) -> content
```

**stroke**

[auto](#) or [length](#) or [color](#) or [gradient](#) or [stroke](#) or [pattern](#) or [dictionary](#)

**Settable**

How to [stroke](#) 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.

Default: `auto`

[View example](#)

```
This is #strike(stroke: 1.5pt + red) [very stricken through]. \
This is #strike(stroke: 10pt) [redacted].
```

This is ~~very stricken through~~.

This is .

**offset**

[auto](#) or [length](#)

**Settable**

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.

Default: `auto`

[View example](#)

```
#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~~.

**extent**length**Settable**

The amount by which to extend the line beyond (or within if negative) the content.

Default: `0pt`

[View example](#)

```
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.

**background**bool**Settable**

Whether the line is placed behind the content.

Default: `false`

[View example](#)

```
#set strike(stroke: red)
#strike(background: true) [This is behind.] \
#strike(background: false) [This is in front.]
```

~~This is behind.~~  
~~This is in front.~~

**body**content**RequiredPositional**

The content to strike through.

## 3.7.10 Subscript

The text is rendered smaller and its baseline is lowered.

## Example

```
Revenue#sub[yearly]
```

Revenue<sub>yearly</sub>

## Parameters

```
sub (
 typographic: bool,
 baseline: length,
 size: length,
 content,
) -> content
typographic
bool
Settable
```

Whether to prefer the dedicated subscript characters of the font.

If this is enabled, Typst first tries to transform the text to subscript codepoints. If that fails, it falls back to rendering lowered and shrunk normal letters.

Default: `true`

[View example](#)

```
N#sub(typographic: true) [1]
N#sub(typographic: false) [1]
```

The image shows two characters side-by-side: a standard subscript 'N' and a synthetic subscript 'N'. The synthetic subscript is lower and smaller than the standard one, demonstrating the difference in rendering when typographic is set to false.

```
baseline
length
Settable
```

The baseline shift for synthetic subscripts. Does not apply if `typographic` is `true` and the font has subscript codepoints for the given `body`.

Default: `0.2em`

**size**length*Settable*

The font size for synthetic subscripts. Does not apply if `typographic` is true and the font has subscript codepoints for the given `body`.

Default: `0.6em`

**body**content*RequiredPositional*

The text to display in subscript.

### 3.7.11 Superscript

Renders text in superscript.

The text is rendered smaller and its baseline is raised.

#### Example

```
1 #super[st] try!
```

`1st try!`

### Parameters

```
super(
 typographic: bool,
 baseline: length,
 size: length,
 content,
) -> content
typographic
bool
Settable
```

Whether to prefer the dedicated superscript characters of the font.

If this is enabled, Typst first tries to transform the text to superscript codepoints.

If that fails, it falls back to rendering raised and shrunk normal letters.

Default: `true`

[View example](#)

```
N#super (typographic: true) [1]
N#super (typographic: false) [1]
```

**baseline**

length

*Settable*

The baseline shift for synthetic superscripts. Does not apply if

`typographic` is `true` and the font has superscript codepoints for the given

`body`.

Default: `-0.5em`

**size**

length

*Settable*

The font size for synthetic superscripts. Does not apply if `typographic` is

`true` and the font has superscript codepoints for the given `body`.

Default: `0.6em`

**body**

content

*RequiredPositional*

The text to display in superscript.

## 3.7.12 Text

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.

## Example

```
#set text(18pt)
With a set rule.
```

```
#emph(text(blue) [
 With a function call.
])
```

With a set rule.

*With a function call.*

## Parameters

```
text(
 font: strarray,
 fallback: bool,
 style: str,
 weight: intstr,
 stretch: ratio,
 size: length,
 fill: colorgradientpattern,
 stroke: nonelengthcolorgradientstrokepatterndictionary,
 tracking: length,
 spacing: relative,
 cjk-latin-spacing: noneauto,
 baseline: length,
 overhang: bool,
 top-edge: lengthstr,
 bottom-edge: lengthstr,
 lang: str,
 region: nonestr,
 script: autostr,
 dir: autodirection,
 hyphenate: autobool,
```

```

costs: dictionary,
kerning: bool,
alternates: bool,
stylistic-set: noneintarray,
ligatures: bool,
discretionary-ligatures: bool,
historical-ligatures: bool,
number-type: autostr,
number-width: autostr,
slashed-zero: bool,
fractions: bool,
features: arraydictionary,
content,
str,
) -> content
font
str or array

```

**Settable**

A font family name or priority list of font family names.

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.

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-paths` > 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.

**Default:** `"libertinus serif"`

View example

```
#set text(font: "PT Sans")
This is sans-serif.
```

```
#set text(font: (
 "Inria Serif",
 "Noto Sans Arabic",
))
```

```
This is Latin. \
هذا عربي.
```

This is sans-serif.

This is Latin.  
هذا عربي.

**fallback**

bool

**Settable**

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.

Default: `true`

[View example](#)

```
#set text(font: "Inria Serif")
هذا عربي
```

```
#set text(fallback: false)
هذا عربي
```

هذا عربي  
الله عز

**style**

str

**Settable**

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.

If you want to emphasize your text, you should do so using the [emph](#) function instead. This makes it easy to adapt the style later if you change your mind about how to signify the emphasis.

## Variant

## Details

`"normal"` The default, typically upright style.

`"italic"` A cursive style with custom letterform.

`"oblique"` Just a slanted version of the normal style.

Default: `"normal"`

[View example](#)

```
#text(font: "Libertinus Serif", style: "italic") [Italic]
#text(font: "DejaVu Sans", style: "oblique") [Oblique]
```

## *Italic Oblique*

### **weight**

`int` or `str`

#### *Settable*

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.

If you want to strongly emphasize your text, you should do so using the [strong](#) function instead. This makes it easy to adapt the style later if you change your mind about how to signify the strong emphasis.

Variant	Details
<code>"thin"</code>	Thin weight (100).
<code>"extralight"</code>	Extra light weight (200).
<code>"light"</code>	Light weight (300).
<code>"regular"</code>	Regular weight (400).
<code>"medium"</code>	Medium weight (500).

`"semibold"` Semibold weight (600).

`"bold"` Bold weight (700).

`"extrabold"` Extrabold weight (800).

`"black"` Black weight (900).

Default: `"regular"`

[View example](#)

```
#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**

**stretch**

ratio

*Settable*

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.

If you want to adjust the amount of space between characters instead of stretching the glyphs itself, use the [tracking](#) property instead.

Default: `100%`

[View example](#)

```
#text (stretch: 75%) [Condensed] \
#text (stretch: 100%) [Normal]
```

Condensed  
Normal

**size**length*Settable*

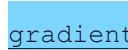
The size of the glyphs. This value forms the basis of the `em` unit: `1em` is equivalent to the font size.

You can also give the font size itself in `em` units. Then, it is relative to the previous font size.

Default: `11pt`

[View example](#)

```
#set text(size: 20pt)
very #text(1.5em) [big] text
```


**fill**
 or  or 
*Settable*

The glyph fill paint.

Default: `luma (0%)`

[View example](#)

```
#set text(fill: red)
This text is red.
```

This text is red.

**stroke**

none or length or color or gradient or stroke or pattern or dictionary

**Settable**

How to stroke the text.

Default: none

[View example](#)

```
#text (stroke: 0.5pt + red) [Stroked]
```

Stroked

**tracking**

length

**Settable**

The amount of space that should be added between characters.

Default: 0pt

[View example](#)

```
#set text (tracking: 1.5pt)
```

Distant text.

Distant text.

**spacing**

relative

**Settable**

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](#) property instead.

Default: `100% + 0pt`

[View example](#)

```
#set text(spacing: 200%)
```

Text with distant words.

Text with distant words.

#### cjk-latin-spacing

[none](#) or [auto](#)

*Settable*

Whether to automatically insert spacing between CJK and Latin characters.

Default: `auto`

[View example](#)

```
#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。

#### baseline

[length](#)

*Settable*

An amount to shift the text baseline by.

Default: `0pt`

[View example](#)

```
A #text(baseline: 3pt) [lowered]
word.
```

A lowered word.

#### **overhang**

`bool`

*Settable*

Whether certain glyphs can hang over into the margin in justified text.

This can make justification visually more pleasing.

Default: `true`

[View example](#)

```
#set par(justify: true)
```

This justified text has a hyphen in the paragraph's first 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 first line. Hanging the hyphen slightly into the margin results in a clearer paragraph edge.

This justified text has a hyphen in the paragraph's first line. Hanging the hyphen slightly into the margin results in a clearer paragraph edge.

This justified text has a hyphen in the paragraph's first line. Hanging the hyphen slightly into the margin results in a clearer paragraph edge.

#### **top-edge**

`length` or `str`

*Settable*

The top end of the conceptual frame around the text used for layout and positioning. This affects the size of containers that hold text.

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.

Default: "cap-height"

[View example](#)

```
#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]
```




**bottom-edge**

length or str

*Settable*

The bottom end of the conceptual frame around the text used for layout and positioning. This affects the size of containers that hold text.

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.

Default: "baseline"

[View example](#)

```
#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]
```

**lang**

str

**Settable**

An [ISO 639-1/2/3 language code](#).

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](#) turns into the correct quotes for the language.

- And all other things which are language-aware.

Default: "en"

View example

```
#set text(lang: "de")
#outline()

= Einleitung
```

In diesem Dokument, ...

## Inhaltsverzeichnis

Einleitung .....	1
------------------	---

### Einleitung

In diesem Dokument, ...

**region**

 [none](#) or  [str](#)

*Settable*

An [ISO 3166-1 alpha-2 region code](#).

This lets the text processing pipeline make more informed choices.

Default: [none](#)

**script**

 [auto](#) or  [str](#)

*Settable*

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.

**Default:** `auto`

[View example](#)

```
#set text(
 font: "Libertinus Serif",
 size: 20pt,
)

#let scedilla = [$]
#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
```

Ş

Ş

Ş

**dir**

`auto` or `direction`

*Settable*

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](#) 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](#), [Discord server](#), or our [contact form](#).

**Default:** `auto`

[View example](#)

```
#set text (dir: rtl)
هذا عربي.
```

هذا عربي.

**hyphenate**

`auto` or `bool`

**Settable**

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](#) ensures that the correct hyphenation patterns are used.

**Default:** `auto`

[View example](#)

```
#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.

### **costs**

#### dictionary

#### **Settable**

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.

Runts are avoided by placing more or fewer words on previous lines, so a higher runt cost can result in more awkward 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.)

**Default:** (`hyphenation: 100%, runt: 100%, widow: 100%, orphan: 100%`, )

[View example](#)

```
#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)
```

**Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do.**

**Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do.**

### **kerning**

bool

**Settable**

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.

Default: `true`

[View example](#)

```
#set text(size: 25pt)
```

Totally

```
#set text(kerning: false)
```

Totally

# Totally

# Totally

**alternates**

bool

*Settable*

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.

Default: `false`

[View example](#)

```
#set text(
 font: "IBM Plex Sans",
 size: 20pt,
)
```

```
0, a, g, ß
```

```
#set text(alternates: true)
0, a, g, ß
```

0, a, g, ß

0, a, g, ß

#### **stylistic-set**

none or int or array

##### *Settable*

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.

Default: ()

[View example](#)

```
#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

#### **ligatures**

bool

##### *Settable*

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.

Default: `true`

[View example](#)

```
#set text(size: 20pt)
A fine ligature.
```

```
#set text(ligatures: false)
A fine ligature.
```

# A fine ligature.

# A fine ligature.

## **discretionary-ligatures**

bool

*Settable*

Whether ligatures that should be used sparingly are active. Setting this to `true` enables the OpenType `dlig` font feature.

Default: `false`

## **historical-ligatures**

bool

*Settable*

Whether historical ligatures are active. Setting this to `true` enables the OpenType `hlig` font feature.

Default: `false`

## **number-type**

auto or str

*Settable*

Which kind of numbers / figures to select. When set to `auto`, the default numbers for the font are used.

Variant	Details
<code>"lining"</code>	Numbers that fit well with capital text (the OpenType <code>lnum</code> font feature).
<code>"old-style"</code>	Numbers that fit well into a flow of upper- and lowercase text (the OpenType <code>onum</code> font feature).

Default: `auto`

[View example](#)

```
#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.

**number-width**

`auto` or `str`

*Settable*

The width of numbers / figures. When set to `auto`, the default numbers for the font are used.

Variant	Details
---------	---------

Numbers with glyph-specific widths (the OpenType `pnum` font "proportional" feature).

`"tabular"` Numbers of equal width (the OpenType `tnum` font feature).

Default: `auto`

[View example](#)

```
#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.

#### `slashed-zero`

bool

*Settable*

Whether to have a slash through the zero glyph. Setting this to

`true` enables the OpenType `zero` font feature.

Default: `false`

[View example](#)

```
0, #text(slashed-zero: true) [0]
```

0, Ø

#### `fractions`

bool**Settable**

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.

Default: `false`

[View example](#)

```
1/2 \
#text (fractions: true) [1/2]
```

1/2

$\frac{1}{2}$

**features**array or dictionary**Settable**

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.

Default: `(:)`

[View example](#)

```
// Enable the `frac` feature manually.
#set text(features: ("frac",))
1/2
```

$\frac{1}{2}$ 

**body**  
content  
*RequiredPositional*

Content in which all text is styled according to the other arguments.

**text**  
str  
*RequiredPositional*

The text.

### 3.7.13 Underline

Underlines text.

#### Example

This is `#underline[important]`.

This is important.

#### Parameters

```
underline (
 stroke: auto length color gradient stroke patterndictionary,
 offset: auto length,
 extent: length,
 evade: bool,
 background: bool,
 content,
) -> content

stroke

auto or length or color or gradient or stroke or patterndictionary
```

*Settable*

How to stroke the line.

If set to `auto`, takes on the text's color and a thickness defined in the current font.

**Default:** `auto`

[View example](#)

```
Take #underline(
 stroke: 1.5pt + red,
 offset: 2pt,
 [care],
)
```

Take care

**offset**

`auto` or `length`

**Settable**

The position of the line relative to the baseline, read from the font tables if `auto`.

**Default:** `auto`

[View example](#)

```
#underline(offset: 5pt) [
 The Tale Of A Faraway Line I
]
```

The Tale Of A Faraway Line I

**extent**

`length`

**Settable**

The amount by which to extend the line beyond (or within if negative) the content.

**Default:** `0pt`

[View example](#)

```
#align(center,
 underline(extent: 2pt) [Chapter 1]
)
```

## Chapter 1

**evade**bool**Settable**

Whether the line skips sections in which it would collide with the glyphs.

Default: `true`[View example](#)

```
This #underline(evade: true) [is great].
This #underline(evade: false) [is less great].
```

This is great. This is less great.

**background**bool**Settable**

Whether the line is placed behind the content it underlines.

Default: `false`[View example](#)

```
#set underline(stroke: (thickness: 1em, paint: maroon, cap: "round"))
#underline(background: true) [This is stylized.] \
#underline(background: false) [This is partially hidden.]
```



**body**  
content  
*RequiredPositional*

The content to underline.

## 3.7.14 Uppercase

Converts a string or content to uppercase.

### Example

```
#upper("abc") \
#upper[*my text*] \
#upper[ALREADY HIGH]
```

ABC  
**MY TEXT**  
 ALREADY HIGH

### Parameters

```
upper(
strcontent
) -> strcontent
text
str or content
```

*RequiredPositional*

The text to convert to uppercase.

## 3.8 Math

Typst has special [syntax](#) 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 $).`

## 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](#).

```
$ A = pi r^2 $
$ "area" = pi dot "radius"^2 $
$ cal(A) :=
 { x in RR | x "is natural" } $
#let x = 5
$ #x < 17 $
```

$$A = \pi r^2$$

$$\text{area} = \pi \cdot \text{radius}^2$$

$$\mathcal{A} := \{x \in \mathbb{R} \mid x \text{ is natural}\}$$

$$5 < 17$$

## Symbols

Math mode makes a wide selection of [symbols](#) like `pi`, `dot`, or `RR` available.

Many mathematical symbols are available in different variants. You can select between different variants by applying [modifiers](#) 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 $
```

$$x < y \Rightarrow x \not\geq y$$

## 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 $
```

$$\sum_{k=0}^n k = 1 + \dots + n \\ = \frac{n(n+1)}{2}$$

## 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, but don't support trailing content blocks and argument spreading.
- 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) $
$ lim_x =
```

```
op("lim", limits: #true) × $
```

$$\frac{a^2}{2}$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\lim_x = \lim_x$$

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.

## 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" $
```

$$\begin{aligned}\frac{3x + y}{7} &= 9 && \text{given} \\ 3x + y &= 63 && \text{multiply by 7} \\ 3x &= 63 - y && \text{subtract y} \\ x &= 21 - \frac{y}{3} && \text{divide by 3}\end{aligned}$$

## Math fonts

You can set the math font by with a [show-set rule](#) as demonstrated below. Note that only special OpenType math fonts are suitable for typesetting maths.

```
#show math.equation: set text(font: "Fira Math")
$ sum_(i in NN) 1 + i $
```

$$\sum_{i \in \mathbb{N}} 1 + i$$

## Math module

All math functions are part of the `math` [module](#), which is available by default in equations. Outside of equations, they can be accessed with the `math.` prefix.

## Definitions

- [`accent`](#) Attaches an accent to a base.
- [`attachSubscript`](#), [`superscripts`](#), and [`limits`](#).
- [`binom`](#) A binomial expression.
- [`cancel`](#) Displays a diagonal line over a part of an equation.

- [cases](#)A case distinction.
- [class](#)Forced use of a certain math class.
- [equation](#)A mathematical equation.
- [frac](#)A mathematical fraction.
- [lrdelimiter](#)Delimiter matching.
- [mat](#)A matrix.
- [op](#)A text operator in an equation.
- [primes](#)Grouped primes.
- [roots](#)Square and non-square roots.
- [sizes](#)Forced size styles for expressions within formulas.
- [stretch](#)Stretches a glyph.
- [styles](#)Alternate letterforms within formulas.
- [underover](#)Delimiters above or below parts of an equation.
- [variants](#)Alternate typefaces within formulas.
- [vec](#)A column vector.

### 3.8.1 Accent

Attaches an accent to a base.

#### Example

```
$grave(a) = accent(a, `)$ \
$arrow(a) = accent(a, arrow)$ \
$tilde(a) = accent(a, \u{0303})$
```

$\grave{a} = \grave{a}$ 
 $\vec{a} = \vec{a}$ 
 $\tilde{a} = \tilde{a}$ 

## Parameters

```
math.accent(
 content,
 strcontent,
 size: autorelative,
) -> content
base
content
RequiredPositional
```

The base to which the accent is applied. May consist of multiple letters.

[View example](#)

`$arrow(A B C)$`

 $\overrightarrow{ABC}$ 

**accent**

str or content

*RequiredPositional*

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.triplet	˙˙˙
Quadruple dot	dot.quad	˙˙˙˙
Circle	circle	◦
Double acute	acute.double	˝
Caron	caron	ˇ
Right arrow	arrow, ->	→
Left arrow	arrow.l, <-	←
Left/Right arrow	arrow.l.r	↔
Right harpoon	harpoon	→
Left harpoon	harpoon.lt	←

**size**auto or relative**Settable**

The size of the accent, relative to the width of the base.

Default: auto

## 3.8.2 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](#) function.

## Example

```
$ sum_(i=0)^n a_i = 2^(1+i) $
```

$$\sum_{i=0}^n a_i = 2^{1+i}$$

## 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.

## Functions

### `attachElement`

A base with optional attachments.

```
math.attach(
 content,
 t: nonecontent,
 b: nonecontent,
 tl: nonecontent,
 bl: nonecontent,
 tr: nonecontent,
 br: nonecontent,
) -> content
$ attach(
 Pi, t: alpha, b: beta,
 tl: 1, tr: 2+3, bl: 4+5, br: 6,
```

) \$

$$\frac{1}{4+5} \prod_{\beta}^{\alpha} 2+3$$

**base**content*RequiredPositional*

The base to which things are attached.

**t**none or content*Settable*

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.

Default: `none`

**b**none or content*Settable*

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.

Default: `none`

**t1**none or content*Settable*

The top-left attachment (before the base).

Default: `none`

**b1**

`none` or `content`

**Settable**

The bottom-left attachment (before base).

Default: `none`

**tr**

`none` or `content`

**Settable**

The top-right attachment (after the base).

Default: `none`

**br**

`none` or `content`

**Settable**

The bottom-right attachment (after the base).

Default: `none`

**scriptsElement**

Forces a base to display attachments as scripts.

```
math.scripts(
 content
) -> content
 $ scripts(sum)_1^2 != sum_1^2 $
```

$$\sum_1^2 \neq \sum_1^2$$

**body**

`content`

**RequiredPositional**

The base to attach the scripts to.

**limitsElement**

Forces a base to display attachments as limits.

```
math.limits(
 content,
 inline: bool,
) -> content
$ limits(A)_1^2 != A_1^2 $
```

$$\overset{2}{A}_1 \neq A_1^2$$

**body**

```
content
```

**RequiredPositional**

The base to attach the limits to.

**inline**

```
bool
```

**Settable**

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.

Default: `true`

### 3.8.3 Binomial

A binomial expression.

#### Example

```
$ binom(n, k) $
$ binom(n, k_1, k_2, k_3, ..., k_m) $
```

$$\binom{n}{k}$$

$$\binom{n}{k_1, k_2, k_3, \dots, k_m}$$

## Parameters

```
math.binom(
 content,
 ..content,
) -> content
upper
content
RequiredPositional
```

The binomial's upper index.

```
lower
content
RequiredPositional
Variadic
```

The binomial's lower index.

## 3.8.4 Cancel

Displays a diagonal line over a part of an equation.

This is commonly used to show the elimination of a term.

### Example

```
Here, we can simplify:
$ (a dot b dot cancel(x)) /
 cancel(x) $
```

Here, we can simplify:

$$\frac{a \cdot b \cdot x}{x}$$

## Parameters

```
math.cancel(
 content,
 length: relative,
 inverted: bool,
 cross: bool,
 angle: autoanglefunction,
 stroke: lengthcolorgradientstrokepatterndictionary,
) -> content
body
content
RequiredPositional
```

The content over which the line should be placed.

**length**  
relative  
**Settable**

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.

**Default:** `100% + 3pt`

[View example](#)

```
$ a + cancel(x, length: #200%)
 - cancel(x, length: #200%) $
```

**inverted**  
bool  
**Settable**

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.

**Default:** `false`

[View example](#)

```
$ (a cancel((b + c), inverted: #true)) /
 cancel(b + c, inverted: #true) $
```

$$\frac{a(b+c)}{b+c}$$

**cross****bool****Settable**

Whether two opposing cancel lines should be drawn, forming a cross over the element. Overrides `inverted`.

**Default:** `false`[View example](#)

```
$ cancel(Pi, cross: #true) $
```

**angle****auto or angle or function****Settable**

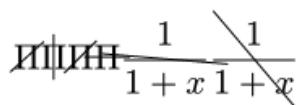
How much to rotate the cancel line.

- 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.

Default: `auto`

[View example](#)

```
$ 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)) $
```



**stroke**

`length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

**Settable**

How to [stroke](#) the cancel line.

Default: `0.5pt`

[View example](#)

```
$ cancel(
 sum x,
 stroke: #(
 paint: red,
 thickness: 1.5pt,
 dash: "dashed",
),
) $
```

$$\sum x$$

## 3.8.5 Cases

A case distinction.

Content across different branches can be aligned with the `&` symbol.

## Example

```
$ f(x, y) := cases(
 1 "if" (x dot y)/2 <= 0,
 2 "if" x "is even",
 3 "if" x in NN,
 4 "else",
) $
```

$$f(x, y) := \begin{cases} 1 & \text{if } \frac{x \cdot y}{2} \leq 0 \\ 2 & \text{if } x \text{ is even} \\ 3 & \text{if } x \in \mathbb{N} \\ 4 & \text{else} \end{cases}$$

## Parameters

```
math.cases(
 delim: nonestrarraysymbol,
 reverse: bool,
 gap: relative,
 ..content,
) -> content
delim

none or str or array or symbol
```

### *Settable*

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.

Default: ("{" , "}")

View example

```
#set math.cases(delim: "[")
$ x = cases(1, 2) $
```

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

**reverse**bool**Settable**

Whether the direction of cases should be reversed.

Default: `false`

[View example](#)

```
#set math.cases(reverse: true)
$ cases(1, 2) = x $
```

$$\begin{cases} 1 \\ 2 \end{cases} = x$$

**gap**relative**Settable**

The gap between branches.

Default: `0% + 0.2em`

[View example](#)

```
#set math.cases(gap: 1em)
$ x = cases(1, 2) $
```

$$x = \begin{cases} 1 \\ 2 \end{cases}$$

**children**content**RequiredPositional****Variadic**

The branches of the case distinction.

## 3.8.6 Class

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](#) and [scripts](#).

### Example

```
#let loves = math.class(
 "relation",
 sym.suit.heart,
)
$ x loves y and y loves 5 $
```

$$x \heartsuit y \wedge y \heartsuit 5$$

### Parameters

```
math.class(
 str,
 content,
) -> content
class
str
RequiredPositional
```

The class to apply to the content.

[View options](#)

#### 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 <code>sum</code> .
"relation"	A relation like = or <code>prec</code> .
"unary"	A unary operator like <code>not</code> .
"binary"	A binary operator like <code>times</code> .
"vary"	An operator that can be both unary or binary like +.

**body****content****RequiredPositional**

The content to which the class is applied.

## 3.8.7 Equation

A mathematical equation.

Can be displayed inline with text or as a separate block.

### Example

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

Let `$a$`, `$b$`, and `$c$` be the side lengths of 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 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)`.

## Syntax

This function also has dedicated syntax: Write mathematical markup within dollar signs to create an equation. Starting and ending the equation with at least one space lifts it into a separate block that is centered horizontally. For more details about math syntax, see the [main math page](#).

## Parameters

```
math.equation(
 block: bool,
 numbering: nonestrfunction,
 number-align: alignment,
 supplement: noneautocontentfunction,
 content,
) -> content
block
bool
Settable
```

Whether the equation is displayed as a separate block.

Default: `false`

**numbering**

`none` or `str` or `function`

***Settable***

How to [number](#) block-level equations.

Default: `none`

View example

```
#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:

$$\phi := \frac{1 + \sqrt{5}}{2} \quad (1)$$

With Equation 1, we get:

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor \quad (2)$$

**number-align**

alignment

***Settable***

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`.

Default: `end + horizon`

View example

```
#set math.equation(numbering: "(1)", number-align: bottom)
```

We can calculate:

```
$ E &= sqrt(m_0^2 + p^2) \
```

```
&approx 125 "GeV" $
```

We can calculate:

$$\begin{aligned} E &= \sqrt{m_0^2 + p^2} \\ &\approx 125 \text{ GeV} \end{aligned} \tag{1}$$

#### **supplement**

`none` or `auto` or `content` or `function`

#### **Settable**

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.

Default: `auto`

[View example](#)

```
#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:

$$\phi := \frac{1 + \sqrt{5}}{2} \tag{1}$$

With Eq. 1, we get:

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor \tag{2}$$

**body**  
content  
*RequiredPositional*

The contents of the equation.

## 3.8.8 Fraction

### Example

```
$ 1/2 < (x+1)/2 $
$ ((x+1)) / 2 = frac(a, b) $
```

$$\frac{1}{2} < \frac{x+1}{2}$$

$$\frac{(x+1)}{2} = \frac{a}{b}$$

### 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 parenthesis. Such parentheses are removed from the output, but you can nest multiple to force them.

### Parameters

```
math.frac(
 content,
 content,
) -> content
num
 content
RequiredPositional
```

The fraction's numerator.

**denom**  
content  
*RequiredPositional*

The fraction's denominator.

## 3.8.9 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.

### Example

```
$ [a, b/2] $
$ lr() sum_(x=1)^n, size: #50% $ x $
$ abs((x + y) / 2) $
```

$$\left[ a, \frac{b}{2} \right]$$

$$\left[ \sum_{x=1}^n x \right]$$

$$\left| \frac{x + y}{2} \right|$$

## Functions

### `lrElement`

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: autorelative,
 content,
) -> content
size
auto or relative
```

**Settable**

The size of the brackets, relative to the height of the wrapped content.

Default: `auto`

```
body
content
RequiredPositional
```

The delimited content, including the delimiters.

**midElement**

Scales delimiters vertically to the nearest surrounding `lr()` group.

```
math.mid(
 content
) -> content
$ { x mid() sum_(i=1)^n w_i|f_i(x)| < 1 } $
```

$$\left\{ x \left| \sum_{i=1}^n w_i |f_i(x)| < 1 \right. \right\}$$

```
body
content
RequiredPositional
```

The content to be scaled.

**abs**

Takes the absolute value of an expression.

```
math.abs(
 size: autorelative,
 content,
) -> content
```

```
$ abs(x/2) $
```

$$\left| \frac{x}{2} \right|$$

**size**

auto or relative

The size of the brackets, relative to the height of the wrapped content.

**body**

content

*RequiredPositional*

The expression to take the absolute value of.

**norm**

Takes the norm of an expression.

```
math.norm(
 size: autorelative,
 content,
) -> content
$ norm(x/2) $
```

$$\left\| \frac{x}{2} \right\|$$

**size**

auto or relative

The size of the brackets, relative to the height of the wrapped content.

**body**

content

*RequiredPositional*

The expression to take the norm of.

**floor**

Floors an expression.

```
math.floor(
 size: autorelative,
 content,
) -> content
$ floor(x/2) $
```

$$\left\lfloor \frac{x}{2} \right\rfloor$$

**size**

auto or relative

The size of the brackets, relative to the height of the wrapped content.

**body**

content

*RequiredPositional*

The expression to floor.

**ceil**

Ceils an expression.

```
math.ceil(
 size: autorelative,
 content,
) -> content
$ ceil(x/2) $
```

$$\left\lceil \frac{x}{2} \right\rceil$$

**size**

auto or relative

The size of the brackets, relative to the height of the wrapped content.

**body**

content

*RequiredPositional*

The expression to ceil.

**round**

Rounds an expression.

```
math.round(
 size: autorelative,
 content,
) -> content
 $ round(x/2) $
```

$$\left\lfloor \frac{x}{2} \right\rfloor$$

**size**

auto or relative

The size of the brackets, relative to the height of the wrapped content.

**body**

content

*RequiredPositional*

The expression to round.

## 3.8.10 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 parameter, or content in cells that are in the same row can be aligned with the & symbol.

## Example

```
$ mat(
 1, 2, ..., 10;
```

```

2, 2, ..., 10;
dots.v, dots.v, dots.down, dots.v;
10, 10, ..., 10;
) $
```

$$\begin{pmatrix} 1 & 2 & \dots & 10 \\ 2 & 2 & \dots & 10 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix}$$

## Parameters

```

math.mat(
 delim: nonestrarraysymbol,
 align: alignment,
 augment: noneintdictionary,
 gap: relative,
 row-gap: relative,
 column-gap: relative,
 ..array,
) -> content
```

**delim**

none or str or array or symbol

*Settable*

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.

**Default:** ("(", ")")

[View example](#)

```
#set math.mat(delim: "[")

$ mat(1, 2; 3, 4) $
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

**align**alignment**Settable**

The horizontal alignment that each cell should have.

Default: center

[View example](#)

```
#set math.mat(alignment: right)
$ mat(-1, 1, 1; 1, -1, 1; 1, 1, -1) $
```

$$\begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix}$$

**augment**none or int or dictionary**Settable**

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:

ohline: 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.

`ovline`: 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.

`ostroke`: How to [stroke](#) the line. If set to `auto`, takes on a thickness of `0.05em` and square line caps.

Default: `none`

[View example](#)

```
$ mat(1, 0, 1; 0, 1, 2; augment: #2) $
// Equivalent to:
$ mat(1, 0, 1; 0, 1, 2; augment: #(-1)) $
```

$$\left( \begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 2 \end{array} \right)$$

$$\left( \begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 2 \end{array} \right)$$

```
$ mat(0, 0, 0; 1, 1, 1; augment: #(hline: 1, stroke: 2pt + green)) $
```

$$\left( \begin{array}{ccc} 0 & 0 & 0 \\ \hline 1 & 1 & 1 \end{array} \right)$$

**gap**  
[relative](#)  
*Settable*

The gap between rows and columns.

This is a shorthand to set `row-gap` and `column-gap` to the same value.

**Default:** `0%` + `0pt`

[View example](#)

```
#set math.mat(gap: 1em)
$ mat(1, 2; 3, 4) $
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

### **row-gap**

[relative](#)

*Settable*

The gap between rows.

**Default:** `0%` + `0.2em`

[View example](#)

```
#set math.mat(row-gap: 1em)
$ mat(1, 2; 3, 4) $
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

### **column-gap**

[relative](#)

*Settable*

The gap between columns.

**Default:** `0%` + `0.5em`

[View example](#)

```
#set math.mat(column-gap: 1em)
$ mat(1, 2; 3, 4) $
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

**rows**  
array  
*RequiredPositional*  
*Variadic*

An array of arrays with the rows of the matrix.

[View example](#)

```
#let data = ((1, 2, 3), (4, 5, 6))
#let matrix = math.mat(..data)
$ v := matrix $
```

$$v := \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

## 3.8.11 Primes

Grouped primes.

```
$ a'''_b = a^'''_b $
```

$$a'''_b = a'''_b$$

## 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.

## Parameters

```
math.primes(
int
) -> content
```

**count**int*RequiredPositional*

The number of grouped primes.

## 3.8.12 Roots

Square and non-square roots.

### Example

```
$ sqrt(3 - 2 sqrt(2)) = sqrt(2) - 1 $
$ root(3, x) $
```

$$\sqrt{3 - 2\sqrt{2}} = \sqrt{2} - 1$$

$$\sqrt[3]{x}$$

## Functions

**root***Element*

A general root.

```
math.root(
 nonecontent,
 content,
) -> content
 $ root(3, x) $
```

$$\sqrt[3]{x}$$

**index**none or content*Positional**Settable*

Which root of the radicand to take.

Default: none

**radicand**content*RequiredPositional*

The expression to take the root of.

**sqrt**

A square root.

`math.sqrt(`content`) -> content``$ sqrt(3 - 2 sqrt(2)) = sqrt(2) - 1 $`

$$\sqrt{3 - 2\sqrt{2}} = \sqrt{2} - 1$$

**radicand**content*RequiredPositional*

The expression to take the square root of.

## 3.8.13 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.

## Functions

**display**

Forced display style in math.

This is the normal size for block equations.

```
math.display(
 content,
 cramped: bool,
) -> content
```

```
$sum_i x_i/2 = display(sum_i x_i/2)$
```

$$\sum_i \frac{x_i}{2} = \sum_i \frac{x_i}{2}$$

**body**content*RequiredPositional*

The content to size.

**cramped**bool

Whether to impose a height restriction for exponents, like regular sub- and superscripts do.

Default: `false`

**inline**

Forced inline (text) style in math.

This is the normal size for inline equations.

```
math.inline(
content,
cramped: bool,
) -> content
$ sum_i x_i/2
= inline(sum_i x_i/2) $
```

$$\sum_i \frac{x_i}{2} = \sum_i \frac{x_i}{2}$$

**body**content*RequiredPositional*

The content to size.

**cramped**bool

Whether to impose a height restriction for exponents, like regular sub- and superscripts do.

Default: `false`

### **script**

Forced script style in math.

This is the smaller size used in powers or sub- or superscripts.

```
math.script(
 content,
 cramped: bool,
) -> content
$sum_i x_i/2 = script(sum_i x_i/2)$
```

$$\sum_i \frac{x_i}{2} = \sum_i \frac{x_i}{2}$$

```
body
content
RequiredPositional
```

The content to size.

```
cramped
bool
```

Whether to impose a height restriction for exponents, like regular sub- and superscripts do.

Default: `true`

### **sscript**

Forced second script style in math.

This is the smallest size, used in second-level sub- and superscripts (script of the script).

```
math.sscript(
 content,
 cramped: bool,
```

```
) -> content
$sum_i x_i/2 = sscript(sum_i x_i/2)$
```

$$\sum_i \frac{x_i}{2} = \Sigma_i \frac{x_i}{2}$$

**body**content*RequiredPositional*

The content to size.

**cramped**bool

Whether to impose a height restriction for exponents, like regular sub- and superscripts do.

Default: `true`

## 3.8.14 Stretch

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 $
```

$$H \overset{\text{define}}{=} U + pV$$

$$f : X \xrightarrow[\text{surjective}]{} Y$$

$$x \leftrightharpoons y[z]$$

## Parameters

```
math.stretch(
 content,
 size: autorelative,
) -> content
body
content
RequiredPositional
```

The glyph to stretch.

**size**

auto or relative

**Settable**

The size to stretch to, relative to the maximum size of the glyph and its attachments.

Default: auto

## 3.8.15 Styles

## 3.8.16 Text Operator

### Example

```
$ tan x = (sin x) / (cos x) $
$ op("custom",
 limits: #true)_(n->oo) n $
```

$$\tan x = \frac{\sin x}{\cos x}$$

$$\underset{n \rightarrow \infty}{\text{custom } n}$$

## Predefined Operators

Typst predefines the operators `arccos`, `arcsin`, `arctan`, `arg`, `cos`, `cosh`, `cot`, `coth`, `csc`, `csch`, `ctg`, `deg`, `det`, `dim`, `exp`, `gcd`, `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`.

## Parameters

```
math.op(
 content,
 limits: bool,
) -> content
text
content
RequiredPositional
```

The operator's text.

**limits**  
bool  
**Settable**

Whether the operator should show attachments as limits in display mode.

Default: `false`

### 3.8.17 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.

## Functions

### **`underlineElement`**

A horizontal line under content.

```
math.underline(
 content
) -> content
$ underline(1 + 2 + ... + 5) $
```

$$\underline{1 + 2 + \dots + 5}$$

### **`body`**

### **content**

### ***RequiredPositional***

The content above the line.

### **`overlineElement`**

A horizontal line over content.

```
math.overline(
 content
) -> content
$ overline(1 + 2 + ... + 5) $
```

$$\overline{1 + 2 + \dots + 5}$$

### **`body`**

### **content**

### ***RequiredPositional***

The content below the line.

### **`underbraceElement`**

A horizontal brace under content, with an optional annotation below.

```
math.underbrace(
```

```
content,
nonecontent,
) -> content
$ underbrace(1 + 2 + ... + 5, "numbers") $
```

$$\underbrace{1 + 2 + \dots + 5}_{\text{numbers}}$$

**body**  
**content**  
*RequiredPositional*

The content above the brace.

**annotation**

**none** or **content**

*Positional*

*Settable*

The optional content below the brace.

Default: **none**

**overbraceElement**

A horizontal brace over content, with an optional annotation above.

```
math.overbrace(
content,
nonecontent,
) -> content
$ overbrace(1 + 2 + ... + 5, "numbers") $
```

$$\overbrace{1 + 2 + \dots + 5}^{\text{numbers}}$$

**body**  
**content**  
*RequiredPositional*

The content below the brace.

**annotation**

`none` or `content`

*Positional*

*Settable*

The optional content above the brace.

Default: `none`

`underbracketElement`

A horizontal bracket under content, with an optional annotation below.

```
math.underbracket(
 content,
 nonecontent,
) -> content
 $ underbracket(1 + 2 + ... + 5, "numbers") $
```

$$\underbrace{1 + 2 + \dots + 5}_{\text{numbers}}$$

`body`

`content`

*RequiredPositional*

The content above the bracket.

`annotation`

`none` or `content`

*Positional*

*Settable*

The optional content below the bracket.

Default: `none`

`overbracketElement`

A horizontal bracket over content, with an optional annotation above.

```
math.overbracket(
 content,
 nonecontent,
) -> content
```

```
$ overbracket(1 + 2 + ... + 5, "numbers") $
```

$$\overbrace{1+2+\dots+5}^{\text{numbers}}$$

**body****content***RequiredPositional*

The content below the bracket.

**annotation**

none or content

*Positional**Settable*

The optional content above the bracket.

Default: none

**underparenElement**

A horizontal parenthesis under content, with an optional annotation

below.

```
math.underparen(
 content,
 nonecontent,
) -> content
$ underparen(1 + 2 + ... + 5, "numbers") $
```

$$\underbrace{1+2+\dots+5}_{\text{numbers}}$$

**body****content***RequiredPositional*

The content above the parenthesis.

**annotation**

none or content

***Positional******Settable***

The optional content below the parenthesis.

Default: `none`

***overparenElement***

A horizontal parenthesis over content, with an optional annotation above.

```
math.overparen(
 content,
 nonecontent,
) -> content
 $ overparen(1 + 2 + ... + 5, "numbers") $
```

$$\overbrace{1 + 2 + \dots + 5}^{\text{numbers}}$$

***body******content******RequiredPositional***

The content below the parenthesis.

***annotation***

`none` or `content`

***Positional******Settable***

The optional content above the parenthesis.

Default: `none`

***undershellElement***

A horizontal tortoise shell bracket under content, with an optional annotation below.

```
math.undershell(
 content,
 nonecontent,
```

```
) -> content
$ undershell(1 + 2 + ... + 5, "numbers") $
```

$$\underbrace{1 + 2 + \dots + 5}_{\text{numbers}}$$

**body**content*RequiredPositional*

The content above the tortoise shell bracket.

**annotation**none or content*Positional**Settable*

The optional content below the tortoise shell bracket.

Default: none

**overshellElement**

A horizontal tortoise shell bracket over content, with an optional

annotation above.

```
math.overshell(
content,
nonecontent,
) -> content
$ overshell(1 + 2 + ... + 5, "numbers") $
```

$$\overbrace{1 + 2 + \dots + 5}^{\text{numbers}}$$

**body**content*RequiredPositional*

The content below the tortoise shell bracket.

**annotation**

`none` or `content`

*Positional*

*Settable*

The optional content above the tortoise shell bracket.

Default: `none`

## 3.8.18 Variants

Alternate typefaces within formulas.

These functions are distinct from the `text` function because math fonts contain multiple variants of each letter.

### Functions

**serif**

Serif (roman) font style in math.

This is already the default.

```
math.serif(
 content
) -> content
```

**body**

`content`

*RequiredPositional*

The content to style.

**sans**

Sans-serif font style in math.

```
math.sans(
 content
) -> content
$ sans(A B C) $
```

*ABC*

**body**

content*RequiredPositional*

The content to style.

**frak**

Fraktur font style in math.

```
math.frak(
content
) -> content
$ frak(P) $
```

$$\mathfrak{P}$$
**body**content*RequiredPositional*

The content to style.

**mono**

Monospace font style in math.

```
math.mono(
content
) -> content
$ mono(x + y = z) $
```

$$x + y = z$$
**body**content*RequiredPositional*

The content to style.

**bb**

Blackboard bold (double-struck) font style in math.

For uppercase latin letters, blackboard bold is additionally available through

[symbols](#) of the form `NN` and `RR`.

```
math.bb(
 content
) -> content
$ bb(b) $
$ bb(N) = NN $
$ f: NN -> RR $
```

b

N = N

$f : \mathbb{N} \rightarrow \mathbb{R}$

**body**  
 content  
*RequiredPositional*

The content to style.

**cal**

Calligraphic font style in math.

```
math.cal(
 content
) -> content
Let $cal(P)$ be the set of ...
```

Let  $\mathcal{P}$  be the set of ...

This corresponds both to LaTeX's `\mathcal` and `\mathscr` as both of these styles share the same Unicode codepoints. Switching between the styles is thus only possible if supported by the font via [font features](#).

For the default math font, the roundhand style is available through the `ss01` feature. Therefore, you could define your own version of `\mathscr` like this:

```
#let scr(it) = text(
 features: ("ss01",),
 box($cal(it)$),
)
```

We establish `$cal(P) != scr(P)$.`

We establish  $\mathcal{P} \neq \mathscr{P}$ .

(The box is not conceptually necessary, but unfortunately currently needed due to limitations in Typst's text style handling in math.)

**body**  
content  
*RequiredPositional*

The content to style.

## 3.8.19 Vector

Content in the vector's elements can be aligned with the [align](#) parameter, or the & symbol.

### Example

```
$ vec(a, b, c) dot vec(1, 2, 3)
 = a + 2b + 3c $
```

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = a + 2b + 3c$$

## Parameters

```
math.vec(
 delim: nonestrarraysymbol,
 align: alignment,
 gap: relative,
 ...content,
```

```
) -> content
```

**delim**

none or str or array or symbol

**Settable**

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.

Default: `("(", ")")`

[View example](#)

```
#set math.vec(delim: "[")

$ vec(1, 2) $
```

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

**align**

alignment

**Settable**

The horizontal alignment that each element should have.

Default: `center`

[View example](#)

```
#set math.vec(align: right)

$ vec(-1, 1, -1) $
```

$$\begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}$$

**gap**

relative

**Settable**

The gap between elements.

Default: `0%` + `0.2em`

[View example](#)

```
#set math.vec(gap: 1em)
$ vec(1, 2) $
```

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

**children**

**content**

**RequiredPositional**

**Variadic**

The elements of the vector.

## 3.9 Symbols

These two 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.

### Definitions

- [sym](#) These two modules give names to symbols and emoji to make them easy to
- [emoji](#) These two modules give names to symbols and emoji to make them easy to
- [symbol](#) A Unicode symbol.

## 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.

### Within Markup Mode

### Within Math Mode

### 3.9.1 General Symbols

For example, `#sym.arrow` produces the  $\rightarrow$  symbol. Within [formulas](#), 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`.

Click on a `symbol` to copy it to the clipboard.

$\mathbb{A}_{AA}$

$A_{Alpha}$

$\mathbb{B}_{BB}$

$B_{Beta}$

$C_{CC}$

$X_{Chi}$

$\mathbb{D}_{DD}$

$\Delta_{Delta}$

$E_{EE}$

$E_{Epsilon}$

$H_{Eta}$

$F_{FF}$

$G_{GG}$

$\Gamma_{Gamma}$

$H_{HH}$

$I_{II}$

$\mathfrak{I}_{Im}$

$I_{Iota}$

$J_{JJ}$

$K_{KK}$

$K_{Kai}$

$K_{Kappa}$

$L_{LL}$

$\Lambda_{Lambda}$

$M_{MM}$

$M_{Mu}$

$N_{NN}$

$N_{Nu}$

$O_{oo}$

$\Omega_{Omega}$

$O_{Omicron}$

$P_{PP}$

$\Phi_{Phi}$

$\Pi_{Pi}$

$\Psi_{Psi}$

$Q_{QQ}$

$R_{RR}$

$\mathfrak{R}_{Re}$

$P_{Rho}$

$S_{SS}$

$\Sigma_{Sigma}$

$T_{TT}$

$T_{Tau}$	$\Theta_{Theta}$	$U_{UU}$
$\Upsilon_{Upsilon}$	$V_{VV}$	$W_{WW}$
$X_{XX}$	$E_{Xi}$	$Y_{YY}$
$Z_{ZZ}$	$Z_{Zeta}$	' acute
" acute.double	$\aleph_{alef}$	$\alpha_{alpha}$
&	$\wp_{amp.inv}$	$j\ddot{A}_{and}$
$\Lambda_{and.big}$	$\lambda_{and.curly}$	$\Lambda_{and.dot}$
$\Lambda_{and.double}$	$j\ddot{\Lambda}_{angle}$	$\langle_{angle.l}$
$\rangle_{angle.r}$	$j\P_{angle.l.double}$	$j^\cdot_{angle.r.double}$
$\angle_{angle.acute}$	$\triangleleft_{angle.arc}$	$\triangleright_{angle.arc.rev}$
$\Delta_{angle.rev}$	$\sqsubset_{angle.right}$	$\square_{angle.right.rev}$
$\triangleleft_{angle.right.arc}$	$\triangleleft_{angle.right.dot}$	$\triangleleft_{angle.right.sq}$
$\swarrow_{angle.spatial}$	$\triangleleft_{angle.spheric}$	$\triangleright_{angle.spheric.rev}$
$\nwarrow_{angle.spheric.top}$	$\text{\AA}_{angstrom}$	$\approx_{approx}$
$\approxeq_{approx.eq}$	$\not\approx_{approx.not}$	$\rightarrow_{arrow.r}$
$\longleftarrow_{arrow.r.long.bar}$	$\rightarrow_{arrow.r.bar}$	$\curvearrowleft_{arrow.r.curve}$
$\cdots\rightarrow_{arrow.r.dashed}$	$\cdots\rightarrow_{arrow.r.dotted}$	$\Rightarrow_{arrow.r.double}$

$\Rightarrow$ arrow.r.double.bar	$\Rightarrow$ arrow.r.double.long	$\Rightarrow$ arrow.r.double.long.b ar
$\not\Rightarrow$ arrow.r.double.not	$\rightarrow$ arrow.r.filled	$\hookleftarrow$ arrow.r.hook
$\longrightarrow$ arrow.r.long	$\rightsquigarrow$ arrow.r.long.squiggly	$\looparrowright$ arrow.r.loop
$\not\rightarrow$ arrow.r.not	$\Rrightarrow$ arrow.r.quad	$\rightsquigarrow$ arrow.r.squiggly
$\rightarrow$ arrow.r.stop	$\Rightarrow$ arrow.r.stroked	$\rightarrowtail$ arrow.r.tail
$\Rrightarrow$ arrow.r.triple	$\Rrightarrow$ arrow.r.twohead.bar	$\rightarrowtail$ arrow.r.twohead
$\rightsquigarrow$ arrow.r.wave	$\leftarrow$ arrow.l	$\leftarrowtail$ arrow.l.bar
$\hookleftarrow$ arrow.l.curve	$\leftarrowtail$ arrow.l.dashed	$\leftarrowtail$ arrow.l.dotted
$\Leftarrow$ arrow.l.double	$\Lleftarrow$ arrow.l.double.bar	$\Leftarrow$ arrow.l.double.long
$\Lleftarrow$ arrow.l.double.long.ba r	$\not\Leftarrow$ arrow.l.double.not	$\leftarrowtail$ arrow.l.filled
$\leftarrowtail$ arrow.l.hook	$\leftarrowtail$ arrow.l.long	$\leftarrowtail$ arrow.l.long.bar
$\leftarrowtail$ arrow.l.long.squiggly	$\leftarrowtail$ arrow.l.loop	$\not\leftarrowtail$ arrow.l.not
$\Lleftarrow$ arrow.l.quad	$\rightsquigarrow$ arrow.l.squiggly	$\leftarrowtail$ arrow.l.stop
$\Lleftarrow$ arrow.l.stroked	$\leftarrowtail$ arrow.l.tail	$\Lleftarrow$ arrow.l.triplet
$\Lleftarrow$ arrow.l.twohead.bar	$\Lleftarrow$ arrow.l.twohead	$\rightsquigarrow$ arrow.l.wave
$\uparrow$ arrow.t	$\uparrow$ arrow.t.bar	$\rightarrowtail$ arrow.t.curve

$\uparrow$ arrow.t.dashed	$\uparrow\uparrow$ arrow.t.double	$\uparrow\uparrow$ arrow.t.filled
$\uparrow\uparrow\uparrow$ arrow.t.quad	$\uparrow$ arrow.t.stop	$\uparrow\uparrow$ arrow.t.stroked
$\uparrow\uparrow\uparrow$ arrow.t.triple	$\uparrow\uparrow$ arrow.t.twohead	$\downarrow$ arrow.b
$\downarrow$ arrow.b.bar	$\curvearrowright$ arrow.b.curve	$\downarrow\downarrow$ arrow.b.dashed
$\downarrow\downarrow$ arrow.b.double	$\downarrow$ arrow.b.filled	$\downarrow\downarrow\downarrow$ arrow.b.quad
$\downarrow\downarrow$ arrow.b.stop	$\downarrow\downarrow$ arrow.b.stroked	$\downarrow\downarrow\downarrow$ arrow.b.triplet
$\downarrow\downarrow$ arrow.b.twohead	$\leftrightarrow$ arrow.l.r	$\leftrightarrow\leftrightarrow$ arrow.l.r.double
$\leftrightarrow\leftrightarrow$ arrow.l.r.double.long	$\not\leftrightarrow$ arrow.l.r.double.not	$\leftrightarrow\leftrightarrow$ arrow.l.r.filled
$\leftrightarrow\leftrightarrow$ arrow.l.r.long	$\not\leftrightarrow$ arrow.l.r.not	$\leftrightarrow\leftrightarrow$ arrow.l.r.stroked
$\leftrightarrow\sim\leftrightarrow$ arrow.l.r.wave	$\uparrow\downarrow$ arrow.t.b	$\uparrow\downarrow\uparrow\downarrow$ arrow.t.b.double
$\updownarrow$ arrow.t.b.filled	$\updownarrow$ arrow.t.b.stroked	$\cdots J$ arrow.tr
$\nearrow\swarrow$ arrow.tr.double	$\nearrow$ arrow.tr.filled	$\nearrow\swarrow$ arrow.tr.hook
$\nearrow\swarrow$ arrow.tr.stroked	$\cdots K$ arrow.br	$\searrow\swarrow$ arrow.br.double
$\searrow\swarrow$ arrow.br.filled	$\searrow$ arrow.br.hook	$\searrow\swarrow$ arrow.br.stroked
$\cdots I$ arrow.tl	$\nwarrow\swarrow$ arrow.tl.double	$\nwarrow\swarrow$ arrow.tl.filled
$\nwarrow\swarrow$ arrow.tl.hook	$\nwarrow$ arrow.tl.stroked	$\cdots L$ arrow.bl
$\swarrow\searrow$ arrow.bl.double	$\swarrow$ arrow.bl.filled	$\swarrow\searrow$ 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.tb
 arrows.bt	 arrows.rrr	 arrows.lll
 ast.basic	 ast.op	 ast.low
 ast.double	 ast.tripple	 ast.small
 ast.circle	 ast.sq	 at
 backslash	 backslash.circle	 backslash.not
 ballot	 ballot.x	 bar.v
 bar.v.double	 bar.v.tripple	 bar.v.broken
 bar.v.circle	 bar.h	 beta.because
 bet	 beta	 beta.alt
 bitcoin	 bot	 brace.l
 brace.r	 brace.t	 brace.b

〔bracket.l	〔〔bracket.l.double	〕bracket.r
〕〕bracket.r.double	〕 bracket.t	〕 bracket.b
^ breve	^ caret	^ caron
✓ checkmark	✓ checkmark.light	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
○ circle.nested	% oco	: colon
::= colon.eq	::= colon.double.eq	, comma
C complement	○ compose	* convolve
© copyright	(P) copyright.sound	† dagger
‡ dagger.double	— dash.en	— dash.em
— dash.fig	~~ dash.wave	—: dash.colon
○ ⊖ dash.circle	~~~ dash.wave.double	° degree
° C degree.c	° F degree.f	δ delta
.. diaer	∅ diameter	◇ diamond.stroked

$\diamond$	diamond.stroked.small	$\diamond$	diamond.stroked.medium	$\diamond$	diamond.stroked.dot
$\blacklozenge$	diamond.filled	$\blacklozenge$	diamond.filled.medium	$\blacklozenge$	diamond.filled.small
$\partial_{\text{diff}}$		$\div_{\text{div}}$		$\bigodot_{\text{div.circle}}$	
$\cdots 0$	divides	$\nmid_{\text{divides.not}}$		$\$_{\text{dollar}}$	
$\cdot_{\text{dot.op}}$		$\cdot_{\text{dot.basic}}$		$\cdot_{\text{dot.c}}$	
$\mathbf{j}\tilde{\mathbf{N}}$	dot.circle	$\bigodot_{\text{dot.circle.big}}$		$\square_{\text{dot.square}}$	
$\cdots$	dot.double	$\cdots_{\text{dot.triple}}$		$\cdots_{\text{dot.quad}}$	
$\cdots\cdots$	dots.h.c	$\cdots_{\text{dots.h}}$		$\vdots_{\text{dots.v}}$	
$\cdots\cdots$	dots.down	$\cdots\cdots_{\text{dots.up}}$		$\ell_{\text{ell}}$	
$\bigcirc$	ellipse.stroked.h	$\bigcirc$	ellipse.stroked.v	$\bullet$	ellipse.filled.h
$\bullet$	ellipse.filled.v	$\mathfrak{E}$	epsilon	$\mathbb{E}$	epsilon.alt
$=_{\text{eq}}$		$\stackrel{*}{=}_{\text{eq.star}}$		$\bigoplus_{\text{eq.circle}}$	
$=:_{\text{eq.colon}}$		$\stackrel{\text{def}}{=}_{\text{eq.def}}$		$\triangleq_{\text{eq.delta}}$	
$\asymp_{\text{eq.equi}}$		$\trianglelefteq_{\text{eq.est}}$		$\geq_{\text{eq.gt}}$	
$\lessgtr_{\text{eq.lt}}$		$\stackrel{m}{=}_{\text{eq.m}}$		$\neq_{\text{eq.not}}$	
$\lessapprox_{\text{eq.prec}}$		$\stackrel{?}{=}_{\text{eq.quest}}$		$\mathbb{C}_{\text{,,eq.small}}$	

$\asymp$ eq.succ	$\eta$ eta	$\text{€}$ euro
$!$ excl	$!!$ excl.double	$!$ excl.inv
$!?$ excl.quest	$\exists$ exists	$\nexists$ exists.not
$\{\!\!$ fence.l	$\{\!\!\!$ fence.l.double	$\{\!\!\!$ fence.r
$\{\!\!\!$ fence.r.double	$\cdot\!\cdot$ fence.dotted	$\circlearrowleft$ floral
$\bullet\!\bullet$ floral.l	$\delta\bullet$ floral.r	$\forall$ forall
$\text{₣}$ franc	$\gamma$ gamma	$\lambda$ gimel
$\grave{`}$ grave	$>$ gt	$\circledgt;$ gt.circle
$\gtdot$	$\gg$ gt.double	$\geq$ gt.eq
$\gtlt$	$\nexists$ gt.eq.not	$\cdots R$ gt.eqq
$\gtlt$	$\nexists$ gt.lt.not	$\ngeq$ gt.neqq
$\hat{U}$ gt.not	$\gttilde$	$\textcircledC f$ gt.small
$\gttilde$	$\nexists$ gt.tilde.not	$\triangleright$ gt.tri
$\gttri$	$\nexists$ gt.tri.eq.not	$\ntriangleright$ gt.tri.not
$\gttriple$	$\gttriplenested$	$\rightarrowtail$ harpoon.rt
$\rightarrowtail$ harpoon.rt.bar	$\rightarrowtail$ harpoon.rt.stop	$\rightarrowtail$ 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	-hyph
-hyph_MINUS	hyph_nobreak	·hyph_point
hyph_soft	≡ident	≢ident_not

$\equiv_{\text{ident.strict}}$	$\hat{\epsilon}_{\text{in}}$	$\notin_{\text{in.not}}$
$\exists_{\text{in.rev}}$	$\nexists_{\text{in.rev.not}}$	$\exists_{\text{in.rev.small}}$
$\epsilon_{\text{in.small}}$	$\infty_{\text{infinity}}$	$\int_{\text{integral}}$
$\oint_{\text{integral.arrow.hook}}$	$\oint_{\text{integral.ccw}}$	$\oint_{\text{integral.cont}}$
$\oint_{\text{integral.cont.ccw}}$	$\oint_{\text{integral.cont.cw}}$	$\oint_{\text{integral.cw}}$
$\iint_{\text{integral.double}}$	$\iiint_{\text{integral.quad}}$	$\oint_{\text{integral.sect}}$
$\oint_{\text{integral.sq}}$	$\oint\!\oint_{\text{integral.surf}}$	$\oint\!\oint_{\text{integral.times}}$
$\iiint_{\text{integral.triple}}$	$\oint\!\oint_{\text{integral.union}}$	$\iiint_{\text{integral.vol}}$
$\text{!interrobang}$	$\text{liota}$	$\bowtie_{\text{join}}$
$\bowtie_{\text{join.r}}$	$\bowtie_{\text{join.l}}$	$\bowtie_{\text{join.l.r}}$
$\mathbb{K}_{\text{kai}}$	$\mathbb{K}_{\text{kappa}}$	$\mathbb{K}_{\text{kappa.alt}}$
$\mathbb{K}_{\text{kelvin}}$	$\lambda_{\text{lambda}}$	$\Delta_{\text{laplace}}$
$\mathbb{L}_{\text{lira}}$	$\lozenge_{\text{lozenge.stroked}}$	$\diamond_{\text{lozenge.stroked.small}}$
$\lozenge_{\text{lozenge.stroked.mediu}}\\m$	$\blacklozenge_{\text{lozenge.filled}}$	$\blacklozenge_{\text{lozenge.filled.small}}$
$\blacklozenge_{\text{lozenge.filled.medium}}$	$<_{\text{lt}}$	$\circlearrowleft_{\text{lt.circle}}$
$\lessdot_{\text{lt.dot}}$	$\ll_{\text{lt.double}}$	$\leq_{\text{lt.eq}}$

$\lt{lt.eq.gt}$	$\not\lt{lt.eq.not}$	$\cdots Q\lt{lt.eqq}$
$\leqslant \lt{lt.gt}$	$\not\leq \lt{lt.gt.not}$	$\not\leq \lt{lt.neqq}$
$\mathfrak{j}\lt{lt.not}$	$\not\leq \lt{lt.ntilde}$	$\mathbb{C}, \lt{lt.small}$
$\lesssim \lt{lt.tilde}$	$\not\lesssim \lt{lt.tilde.not}$	$\Delta \lt{lt.tri}$
$\trianglelefteq \lt{lt.tri.eq}$	$\not\trianglelefteq \lt{lt.tri.eq.not}$	$\not\trianglelefteq \lt{lt.tri.not}$
$\lll \lt{lt.triple}$	$\lll \lt{lt.triple.nested}$	$\overline{\phantom{x}}$ macron
$\maltese \lt{maltese}$	$\overline{-} \lt{minus}$	$\ominus \lt{minus.circle}$
$\cdot \lt{minus.dot}$	$\mp \lt{minus.plus}$	$\boxminus \lt{minus.square}$
$\approx \lt{minus.tilde}$	$\triangleleft \lt{minus.triangle}$	$\models \lt{models}$
$\mu \lt{mu}$	$\multimap \lt{multimap}$	$\nabla \lt{nabla}$
$\neg \lt{not}$	$\text{♪} \lt{notes.up}$	$\text{♪} \lt{notes.down}$
$\emptyset \lt{nothing}$	$\not\emptyset \lt{nothing.rev}$	$\nu \lt{nu}$
$\Omega \lt{ohm}$	$\mathcal{U} \lt{ohm.inv}$	$\omega \lt{omega}$
$\mathbf{O} \lt{omicron}$	$\infty_{\infty}$	$\mathring{\text{A}} \lt{A.or}$
$\mathsf{V} \lt{or.big}$	$\mathsf{Y} \lt{or.curly}$	$\mathsf{V} \lt{or.dot}$
$\mathbb{V} \lt{or.double}$	$\mathring{I} \lt{parallel}$	$\mathbb{II} \lt{parallel.circle}$
$\mathbb{H} \lt{parallel.not}$	$( \lt{paren.l}$	$) \lt{paren.r}$

$\text{paren.t}$	$\text{paren.b}$	$\text{pentastroke}$
$\blacklozenge$	$\%$	$\%0$
$\perp_{\text{perp}}$	$\perp_{\text{perp.circle}}$	$\text{Peso}$
$\varphi_{\text{phi}}$	$\Phi_{\text{phi.alt}}$	$\pi_{\text{pi}}$
$\varpi_{\text{pi.alt}}$	$\P_{\text{pilcrow}}$	$\P_{\text{pilcrow.rev}}$
$h_{\text{planck}}$	$\hbar_{\text{planck.reduce}}$	$+_{\text{plus}}$
$\cdot\cdot_{\text{plus.circle}}$	$\oplus\rightarrow_{\text{plus.circle.arrow}}$	$\bigoplus_{\text{plus.circle.big}}$
$\dot{+}_{\text{plus.dot}}$	$\pm_{\text{plus.minus}}$	$\textcircled{\text{C}}\textcircled{\text{E}}_{\text{plus.small}}$
$\boxplus_{\text{plus.square}}$	$\triangleleft_{\text{plus.triangle}}$	$\text{pound}$
$\prec_{\text{prec}}$	$\approx_{\text{prec.approx}}$	$\prec\prec_{\text{prec.double}}$
$\preccurlyeq_{\text{prec.eq}}$	$\not\prec_{\text{prec.eq.not}}$	$\preccurlyeq\preccurlyeq_{\text{prec.eqq}}$
$\not\approx_{\text{prec.napprox}}$	$\not\approx_{\text{prec.neqq}}$	$\not\prec_{\text{prec.not}}$
$\not\approx_{\text{prec.ntilde}}$	$\not\approx_{\text{prec.tilde}}$	$'_{\text{prime}}$
$\cdot\cdot F_{\text{prime.rev}}$	$\cdot\cdot'_{\text{prime.double}}$	$\cdot\cdot''_{\text{prime.double.rev}}$
$\cdot\cdot\cdot_{\text{prime.tripple}}$	$\cdot\cdot\cdot'_{\text{prime.tripple.rev}}$	$\cdot\cdot\cdot''_{\text{prime.quad}}$
$\prod_{\text{product}}$	$\coprod_{\text{product.co}}$	$\jmath\emptyset_{\text{prop}}$
$\Psi_{\text{psi}}$	$\blacksquare_{\text{qed}}$	$?_{\text{quest}}$

??quest.double	?!quest.excl	?quest.inv
"quote.double	'quote.single	"quote.l.double
'quote.l.single	"quote.r.double	'quote.r.single
«quote.angle.l.double	‹quote.angle.l.single	›quote.angle.r.double
›quote.angle.r.single	"quote.high.double	'quote.high.single
,quote.low.double	,quote.low.single	j̄Aratio
□rect.stroked.h	□ rect.stroked.v	—rect.filled.h
■ rect.filled.v	j̄Urefmark	ρrho
Qrho.alt	₱ruble	₹rupee
∩sect	∩sect.and	∩sect.big
∩sect.dot	∩sect.double	∩sect.sq
∩sect.sq.big	∩∩sect.sq.double	§section
;semi	;semi.rev	SMservicemark
₩shin	Ϭsigma	/slash
// slash.double	/// slash.triple	*smash
space	space.nobreak	space.en

space.quad	space.third	space quarter
space.sixth	space.med	space.fig
space.punct	space.thin	space.hair
□ 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.dot	⊒ subset.double
⊑ subset.eq	⊉ subset.eq.not	⊑⊑ subset.eq.sq
⊉⊉ subset.eq.sq.not	⊉⊉ subset.neq	⊉⊉ subset.not
⊑⊑ subset.sq	⊉⊉ subset.sq.neq	succ
≈≈ succ.approx	≈≈≈ succ.double	≈≈≈ succ.eq
≠≠ succ.eq.not	≈≈≈≈ succ.eqq	≈≈≈≈ succ.napprox
≠≠≠ succ.neqq	≠≠≠ succ.not	≈≈≈≈≈ succ.ntilde
≈≈≈≈≈ succ.tilde	♣ suit.club	♦ suit.diamond
♥ suit.heart	♠ suit.spade	Σ sum

$\sum$ sum.integral	$\supset$ supset	$\supsetdot$ supset.dot
$\supsetdouble$ supset.double	$\supseteq$ supset.eq	$\supsetneq$ supset.eq.not
$\supseteqq$ supset.eq.sq	$\supsetneqq$ supset.eq.sq.not	$\supsetneqq$ supset.neq
$\supsetnot$ supset.not	$\supsetsq$ supset.sq	$\supsetneq$ supset.sq.neq
$\tackr$ tack.r	$\tackrlong$ tack.r.long	$\tackl$ tack.l
$\tackllong$ tack.l.long	$\tacklshort$ tack.l.short	$\tackt$ tack.t
$\tacktbig$ tack.t.big	$\tacktdouble$ tack.t.double	$\tacktshort$ tack.t.short
$\tackb$ tack.b	$\tackbbig$ tack.b.big	$\tackbdouble$ tack.b.double
$\tackbshort$ tack.b.short	$\tacklr$ tack.l.r	$\tau$ tau
$\therefore$ therefore	$\theta$ theta	$\vartheta$ theta.alt
$\sim$ tilde.op	$\tilde{}$ tilde.basic	$\approx$ tilde.eq
$\not\sim$ tilde.eq.not	$\not\approx$ tilde.eq.rev	$\not\approx$ tilde.eqq
$\not\approx$ tilde.eqq.not	$\not\approx$ tilde.neqq	$\not\sim$ tilde.not
$\tilde{x}$ tilde.rev	$\tilde{\Omega}$ tilde.rev.eqq	$\approx$ tilde.triple
$\times$ times	$\times$ times.big	$\otimes$ times.circle
$\otimes$ times.circle.big	$*$ times.div	$\lambda$ times.three.l
$\times$ times.three.r	$\times$ times.l	$\times$ times.r

 times.square	 times.triangle	 top
 triangle.stroked.r	 triangle.stroked.l	 triangle.stroked.t
 triangle.stroked.b	 triangle.stroked.bl	 triangle.stroked.br
 triangle.stroked.tl	 triangle.stroked.tr	 triangle.stroked.small.r
 triangle.stroked.small.b	 triangle.stroked.small.l	 triangle.stroked.small.t
 triangle.stroked.rounded	 triangle.stroked.nested	 triangle.stroked.dot
 triangle.filled.r	 triangle.filled.l	 triangle.filled.t
 triangle.filled.b	 triangle.filled.bl	 triangle.filled.br
 triangle.filled.tl	 triangle.filled.tr	 triangle.filled.small.r
 triangle.filled.small.b	 triangle.filled.small.l	 triangle.filled.small.t
 turtle.l	 turtle.r	 turtle.t
 turtle.b	 union	 union.arrow
 union.big	 union.dot	 union.dot.big
 union.double	 union.minus	 union.or
 union.plus	 union.plus.big	 union.sq
 union.sq.big	 union.sq.double	 upsilon

\without	wj	₩ won
₩wreath	Ξxi	¥ yen
ζzeta	zwj	zwnj
zws		

## 3.9.2 Emoji Symbols

Named emoji.

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.

Click on a symbol to copy it to the clipboard.

🆎 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
🔜 arrow.r.soon	⬅ arrow.l.filled	⬅ arrow.l.hook

◀arrow.l.back	◀ arrow.l.end	↑arrow.t.filled
→arrow.t.curve	↑ <sub>TOP</sub> arrow.t.top	↓arrow.b.filled
↖arrow.b.curve	↔arrow.l.r	↔ <sub>ON!</sub> arrow.l.r.on
↑arrow.t.b	„Larrow.bl	„Karrow.br
„Iarrow.tl	„Jarrow.tr	„arrows.cycle
*ast	*ast.box	✉atm
⚛atom	🍆 aubergine	🥑 avocado
🗡axe	Ⓑb	👶 baby
👼baby.angel	👶 baby.box	🍼 babybottle
🎒backpack	🥓 bacon	+-+- badger
🏸badminton	🥯 bagel	กระเป๋ 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	鼴 beaver
🛏bed	🛏 bed.person	🐝 bee
🍺beer	🍻 beer.clink	🐛 beetle

 beetle.lady	 bell	 bell.ding
 bell.not	 bento	 bicyclist
 bicyclist.mountain	 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