

一份(不太)简短的**Typst**介绍

*Typst*官方文档中文翻译版

中文翻译: *Casea*

版本: *June 20, 2023*

日期: *June 23, 2023*

前言

此文档是我在学习Typst时，翻译官方manual时的一些记录。目前，全文已经翻译完了，但是请时刻关注翻译的版本与官网的版本。因为有一定的翻译滞后性以及官网更新的频繁，所以可能出现有些功能不能使用。之后的工作就是好好优化一下页面的排版。

粗略地制作了文档的封面，该封面是仿照《一份(不太)简短的LaTeX 2e介绍》制作的，在此感谢。

目前来说Typst很对我的胃口，但是还是存在着很多的问题，但我相信随着更新与发展，typst一定会越来越好用。

欢迎加入typst中文社区:<https://typst.cn!>

Casea

2023.04.22

Change logs

- **2023.06.23 Version 0.5.0:**

- 支持更多代码高亮。
- 扩展了大纲格式设定。
- 公式语法规则改进: `$f^abs(3)$` 被解释为 `$f^(abs(3))$` , 可以通过添加空格消除歧义: `$f^zeta (3)$` 。
- 对数学公式添加了强制尺寸命令。
- 添加datetime类型

- **2023.05.20 Version 0.4.0:**

- 添加对footnote的支持。
- 添加对LaTeX用户的指南
- 注: 一些笔者认为的小修改并未加入到此手册中。

- **2023.04.26 Version 0.3.0:**

- 重命名了一些符号: `dot.op->dot` `dot->dpt.basic` `ast` 和 `tlide` 同理。
- 将 `mod` 重命名为 `rem` , 目前 `mod` 仍可使用, 直到版本更新。
- 移除 `query` 函数的 `before` 和 `after` 定位参数, 改用 `selector()` 函数进行定位。 `query(heading,before:loc)` -> `query(selector(heading).before(loc),loc)`
- `bottom` 和 `top` 重命名为 `b` 和 `t` ,支持左右上下标。
- 支持删除线样式。支持pdf大纲显示。

- **2023.04.22:**

- `loops`取消了对`index`和`value`的迭代, 现在可以通过`unpacking`和`enumerating`处理。`map()`方法同理。目前的写法 `(index,value) in keys.enumerate()` 。
- 字典方法现在可以使用插入顺序而不是字母顺序。
- 新增了 `unpacking()`, `enumerate()`, `path`, `layout`, `sorted()` method。

- 添加了 `New Computer Modern` 字体。

- **2023.04.09:**

- 对于源码部分使用等宽字体 `Monospac821 BT`，感谢supernova的建议。
- 对标题进行了部分汉化，但是是缺少一点味道。
- 附录添加了symbol速查表。

- **2023.04.05:**

- 2023/04/04 typst推出了v0.1的正式版本，修正了编号问题（目前从0开始）。在我看来其最大的改进就是中文排版的优化（并不支持斜体加粗等格式）。
- 2023/4/5 14:07:58 已完成官方文档的翻译，但是总觉得还缺点什么。想了想那便是自己的理解吧：单纯翻译文档谁都可以做，但是如何使文档更加通俗易懂以及让人快速上手就需要下一点功夫了。
- 接下来的安排：使用show set规则将typ文件大幅缩短（预计一半篇幅）；文章结构（调整章节顺序，语句修改）；优化排版（力气活 🤖）。

目录

Part I Guide for LaTeX Users	1
前言	1
创建文档	1
如何创建章标题，强调？	1
命令	4
模板	8
导入包	9
数学公式	10
“盗版”LaTeX	13
局限性	14
Part II 简明教程	16
1 内容对齐	16
2 图片插入	16
3 盒子创建	19
4 强制分栏	20
5 设置列数	20
6 有序列表	21
7 网格排版	24
7.1 Figure、Grid结合绘制子图	25
Part III 手册	28
8 自定义格式	28
8.1 set规则	28
8.2 show规则	29

9 Typst脚本语言	31
9.1 表达式	31
9.2 代码块Blocks	31
9.3 Let定义变量	32
9.4 条件判断语句	34
9.5 循环语句	35
9.6 字段访问	36
9.7 方法Methods	37
9.8 模块	37
9.9 运算符	38
10 Typst内置类型	39
10.1 none	39
10.2 auto	40
10.3 boolean	40
10.4 integer	40
10.5 float	40
10.6 length	40
10.7 angle	41
10.8 ratio	41
10.9 relative length	42
10.10 fraction	42
10.11 color	42
10.12 Datetime	45
10.13 符号	46
10.14 字符串	48

10.15 文本	50
10.16 数组	51
10.17 字典	54
10.18 函数	55
10.19 参数	57
10.20 seletor	58
10.21 模块	59
11 文本	59
11.1 lorem	59
11.2 斜体emph	60
11.3 加粗strong emphasis	61
11.4 换行linebreak	61
11.5 小写转换lowercase	62
11.6 大写转换uppercase	63
11.7 上划线overline	63
11.8 下划线underline	65
11.9 代码块构建raw text/code	66
11.10 small capitals	68
11.11 引用smart quote	69
11.12 删除线strikethrough	70
11.13 下标subscript	71
11.14 上标superscript	72
11.15 文本格式text	73
12 数学公式	81
12.1 上下标	84

12.2 分数	86
12.3 方程	86
12.4 向量	88
12.5 矩阵	89
12.6 选择语句	90
12.7 定界符	91
12.8 上下分隔符	93
12.9 字母accent	95
12.10 公式字体替换	96
12.11 字体样式设置	98
12.12 round	99
12.13 运算符	99
12.14 根号运算	100
12.15 cancel	100
12.16 binom	102
13 页面布局	103
13.1 对齐align	103
13.2 块block	104
13.3 盒子box	105
13.4 表格table	106
13.5 列表list	108
13.6 强制分栏colbreak	111
13.7 分列clolumns	111
13.8 enum	112
13.9 grid	113

13.10 水平间距h	113
13.11 垂直间距v	113
13.12 隐藏内容hide	114
13.13 测量measure	115
13.14 移动move	116
13.15 间距padding	116
13.16 页面格式page	117
13.17 换页符pagebreak	122
13.18 段落par	122
13.19 换段符parbreak	124
13.20 内容放置place	125
13.21 重复内容repeat	126
13.22 旋转rotate	126
13.23 水平/垂直排列stack	127
13.24 内容放缩scale	128
13.25 定理terms	129
14 可视化	131
14.1 直线line	131
14.2 矩形rect	132
14.3 正方形square	134
14.4 圆形circle	135
14.5 路径path	136
14.6 多边形polgon	137
14.7 椭圆ellipse	138
14.8 图片image	139

15 Meta	139
15.1 文献库导入bibliography	139
15.2 文献引用cite	140
15.3 文献引用ref	142
15.4 图片figure	143
15.5 脚注footnote	143
15.6 标题heading	147
15.7 布局Layout	148
15.8 编号numbering	150
15.9 链接link	151
15.10 访问locate	152
15.11 大纲outline	153
15.12 计数器counter	155
15.13 查找query	160
15.14 文档document	162
16 计算	162
17 Construct	168
17.1 int	168
17.2 float	168
17.3 range	169
17.4 regex	170
17.5 cmyk	171
17.6 RGB	171
17.7 luma	171
17.8 string	171

17.9 Label	171
17.10 symbol	172
18 导入文件	172
18.1 csv	172
18.2 json	173
18.3 plain text	173
18.4 toml	174
18.5 xml	174
18.6 yaml	176
19 Foundations	176
19.1 Assert	176
19.2 Evaluate	177
19.3 Panic	177
19.4 Representation	178
19.5 Type	178

Part I Guide for LaTeX Users

前言

此部分将会简明的介绍LaTeX和Typst的差异之处，并使LaTeX玩家快速上手Typst。

就像LaTeX，Typst是一个基于标记的排版系统。你可以在文本文件中编写文档，并使用命令和语法进行标记。然后使用编译器将源文件排版为PDF格式。然而，Typst在以下几个方面不同于LaTeX：一方面，Typst 使用更专用的语法（就像您可能从 Markdown 中了解到的那样）用于常见任务。Typst 的命令也更有原则性：它们的工作原理相同，因此与 LaTeX 不同，您只需要了解几个通用概念，而不是学习每个包的不同约定。此外，Typst 的编译速度比 LaTeX 快：编译通常需要几毫秒，而不是几秒钟，因此 Web 应用程序和编译器都可以提供即时预览。

我们将在下文介绍从 LaTeX 转换过来的用户在使用 Typst 编写文档时遇到的一些最常见的问题。如果您更喜欢 Typst 的分步介绍，请查看我们的教程。

创建文档

就像LaTeX创建 `.tex` 文件一样，Typst只需要创建 `.typ` 文件即可。无需提供模板文件。默认情况使用的是A4大小的页面，如果使用web版，只需要单击 `+ Empty document`，就可以创建项目并进入编辑器。和LaTeX一样只需要使用空行就可以实现段落。

如何创建章标题，强调？

LaTeX使用命令 `\section`、`\subsection`、`\subsubsection` 创建章标题与子标题。根据写作内容的不同，还支持 `\part` 和 `\chapter`。

在 Typst 中，标题的表现形式十分简约：`= Introduction`、`== In this paper`。标题的深度取决于你使用了多少个等号 `=`。

内容强调通常以斜体文本的形式呈现，通过将文本括在下划线中表示 (`_underscores_`)，着重强调使用粗体表示 (`*stars*`)。

如下表格为 LaTeX 命令与 Typst 命令的对比。也可以查看完整的 cheatsheet。

Element	LaTeX	Typst	See
Strong emphasis	<code>textbf{strong}</code>	strong	Section 11.3
Emphasis	<code>emph{emphasis}</code>	<i>emphasis</i>	Section 11.2
Monospace code	<code>texttt{print(1)}</code>	<code>print(1)</code>	Section 11.9
Link	<code>url{https://typst.app}</code>	<code>https://typst.app/</code>	Section 15.9
Label	<code>label{intro}</code>	<code><intro></code>	Section 17.9
Reference	<code>ref{intro}</code>	<code>@intro</code>	Section 15.3
Citation	<code>cite{humphrey97}</code>	<code>@humphrey97</code>	Section 15.2
Bullet list	<code>itemize</code> environment	<code>• List</code>	Section 13.5
Numbered list	<code>enumerate</code> environment	<code>+ List</code>	Section 6
Term list	<code>description</code> environment	<code>/ Term: List</code>	Section 13.25
Figure	<code>figure</code> environment	<code>figure</code> function	Section 2
Table	<code>table</code> environment	<code>table</code> function	Section 13.4
Equation	<code>\$x\$</code> , <code>align</code> / <code>equation</code> environments	<code>\$x\$</code> , <code>\$ x = y \$</code>	Section 12.3

Typst 中 Lists 不依赖于环境。相反的，他们具有像标题一样的轻量化语法。如果需要创建无序列表，只需要在列表项前面加连字符。

#Code

To write this list in Typst...

```
\begin{itemize}

  \item Fast

  \item Flexible

  \item Intuitive

\end{itemize}
```

#Code

```
...just type this:
- Fast
- Flexible
- Intuitive
```

#Demo

To write this list in Typst...

```
\begin{itemize}

\item Fast

\item Flexible

\item Intuitive

\end{itemize}
...just type this:
• Fast
• Flexible
• Intuitive
```

通过将它们缩进到连字符之外，您还可以在单个列表项中包含多个段落或嵌套列表。如果列表项变得更长，最好在列表项之间放置空行。这会增加列表项之间的间距。要改为获取编号列表（枚举），请使用 + 而不是连字符。对于术语列表（描述），写 `/ Term: Description` 代替。

命令

LaTeX 严重依赖命令（以反斜杠为前缀）。它使用这些宏来影响排版过程以及插入和操作内容。一些命令接受参数，最常见的是它们被括在花括号中：

```
\cite{rasmus}。
```

Typst 区分标记模式和代码模式。标记模式是默认设置，您可以在其中编写文本并使用语法结构，例如 `*` 星号表示粗体文本。代码模式类似于 Python 等其他编程语言，允许您编写类似 `1 + 2 == 3` 的代码。

在 Typst 的标记中，您可以使用井号 `#` 为单个命令（或者更确切地说，表达式）切换到代码模式。这就是您调用函数和使用标记内导入等功能的方式。在这些命令和函数调用中，代码模式适用。要将内容作为值嵌入，您可以使用方括号返回标记模式：

#Code

First, a rectangle:

```
#rect()
```

Let me show how to do

```
#underline([_underlined_ text])
```

We can also do some maths:

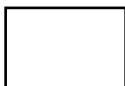
```
#calc.max(3, 2 * 4)
```

And finally a little loop:

```
#for x in range(3) [
  Hi #x.
]
```

#Demo

First, a rectangle:



Let me show how to do underlined text

We can also do some maths: 8

And finally a little loop: Hi 0. Hi 1. Hi 2.

函数调用总是涉及函数的名称（`rect`、`underline`、`calc.max`、`range`），然后是参数列表，即使它是空的。参数列表括在括号中。

一个函数可以有多个参数。一些参数是位置性的，即您只需提供值：函数 `#lower("SCREAM")` 以全小写形式返回其参数。许多函数使用命名参数而不是位置参数来提高易读性。例如，矩形的尺寸和笔划是使用命名参数定义的：

#Code

```
#rect(
  width: 2cm,
  height: 1cm,
  stroke: red,
)
```

#Demo



您可以通过首先输入名称（如上所示，`width`，`height`，and `stroke`）、冒号和价值（`2cm`，`1cm`，`red`）来指定命名参数。您可以在每个函数的参考页或键入时的自动完成面板中找到可用的命名参数。命名参数类似于一些 LaTeX 环境的配置方式，例如，您将键入 `\begin{enumerate}[label={\alph*}]` 以使用标签 `a)`、`b)` 等开始列表。通常，您希望为函数提供一些内容。例如，LaTeX 命令 `\underline{Alternative A}` 在 Typst 中将转换为 `#underline([Alternative A])`。方括号表示一个值是内容。在这些括号内，您可以使用普通标记。然而，对于一个非常简单的构造来说，这是很多括号。这就是为什么您还可以在括号之后移动尾随内容参数（如果括号最终为空，则省略括号）。

#Code

Typst is an `#underline[alternative]`
to LaTeX.

`#rect(fill: aqua)[Get started here!]`

#Demo

Typst is an alternative to LaTeX.

Get started here!

您可能已经注意到参数具有不同的数据类型。Typst 支持多种数据类型。下面是一个表格，其中包含一些最重要的内容以及如何编写它们：

Data type	Example
Content	<code>*fast*</code> typesetting
String	“Pietro S. Author”
Integer	23
Floating point number	1.459
Absolute length	12pt, 5in, 0.3cm, ...
Relative length	65%

内容和字符串之间的区别在于内容可以包含标记，包括函数调用，而字符串实际上只是一个字符序列。您可以像在传统编程语言中那样使用 `+` 运算符求和和 `==` 等运算符来计算这些类型的相等性，而不是使用 `\addtocounter` 或 `\ifnum`。您甚至可以定义变量并使用它们进行计算。

为了指定任何这些类型的值，您必须处于代码模式！

在 LaTeX 中，一些像 `\textbf{bold text}` 这样的命令在花括号中传递它们的参数并且只影响那个参数，而其他像 `\bfseries bold text` 这样的命令充当开关并改变文档或当前范围中所有后续内容的外观（由一组大括号表示）。在 Typst 中，函数可以两种方式使用：效果应用到文档或块的末尾，或者只应用

到它的参数。例如，`#text(weight: "bold")[bold text]` 只会加粗它的论点，而 `#set text(weight: "bold")` 会加粗任何文本，直到当前块结束。一个函数的效果是显而易见的，这取决于它是在调用中使用还是在设置规则中使用。

#Code

I am starting out with small text.

```
#set text(14pt)
```

This is a bit `#text(18pt)[larger,]`
don't you think?

#Demo

I am starting out with small text.

This is a bit **larger**, don't you think?

`set`规则可能出现在文档中的任何地方，并且可以被认为是预先设置其函数的参数：

#Code

```
#set enum(numbering: "I.")
```

Good results can only be obtained by

- + following best practices
- + being aware of current results
of other researchers
- + checking the data for biases

#Demo

Good results can only be obtained by

- I. following best practices
- II. being aware of current results
of other researchers
- I. checking the data for biases

`+` 是调用枚举函数的语法糖（将其视为缩写），我们在上面应用了一组规则。大多数语法都以这种方式链接到函数。如果你需要设置一个元素的样式超出其参数启用的范围，你可以使用显示规则（有点类似于 `\renewcommand`）完全重新定义它的外观。

模板

在 LaTeX 中，您可以使用 `\documentclass{article}` 命令启动您的主 `.tex` 文件，以定义您的文档的外观。在该命令中，您可能已将 `article` 替换为另一个值，例如 `report` 和 `amsart` 以选择不同的外观。使用 Typst 时，您可以使用函数来设计文档的样式。通常，您使用的模板提供了一个功能，可以为整个文档设置样式。首先，您从模板文件导入函数。然后，将其应用于整个文档。这是通过将以下文档包装在给定函数中的显示规则来实现的。下面的例子说明了它是如何工作的：

```
#Code
#import "conf.typ": conf
#show: conf.with(
  title: [
    Towards Improved Modelling
  ],
  authors: (
    (
      name: "Theresa Tungsten",
      affiliation: "Artos Institute",
      email: "tung@artos.edu",
    ),
    (
      name: "Eugene Deklan",
      affiliation: "Honduras State",
      email: "e.deklan@hstate.hn",
    ),
  ),
  abstract: lorem(80),
```

```
)
```

```
Let's get started writing this
article by putting insightful
paragraphs right here!
```

`import` 语句使来自另一个文件的函数（和其他定义）可用。在此示例中，它从 `conf.typ` 文件导入 `conf` 函数。此功能将内容格式化为会议文章。我们使用 `show` 规则将其应用于文档，并配置一些关于文章的元数据。最后，我们可以开始写下面的文章了！

函数是 Typst 的“命令”，可以将它们的参数转换为输出值，包括文档内容。函数是“纯”的，这意味着它们除了创建输出值/输出内容之外不能有任何效果。这与可以对您的文档产生任意影响的 LaTeX 宏形成鲜明对比。

为了让函数为整个文档设置样式，`show` 规则处理它之后的所有内容，并调用冒号后指定的函数，并将结果作为参数。`.with` 部分是一种采用 `conf` 函数并在将其传递给 `show` 规则之前预先配置其参数的方法。

在 Web 应用程序中，您可以从预定义的模板中进行选择，甚至可以使用模板向导创建您自己的模板。您还可以查看 [awesome-typst](#) 存储库以查找社区制作的模板。我们计划构建一个包管理器，使模板在未来更容易共享！您还可以创建自己的自定义模板。它们比相应的 LaTeX `.sty` 文件短几个数量级，可读性更强，所以试一试吧！

导入包

大多数你在 LaTeX 中加载包的东西都只包含在 Typst 中，不需要加载或安装任何东西。下面，我们整理了一张表格，列出了经常加载的包及其对应的 Typst 函数。

如果您需要从另一个文件加载函数和变量，例如使用模板，您可以使用 `import` 语句。如果你想包含另一个文件的文本内容，您可以使用 `include` 语句。它将产生包含文件的内容并将其放入您的文档中。目前，Typst 没有包管理器，但我们计划构建一个，以便您可以轻松地使用来自社区的工具和模板的包，并发布您自己的包。

<code>graphicx, svg</code>	image function
<code>tabularx</code>	table, grid functions
<code>fontenc, inputenc, unicode-math</code>	Just start writing!
<code>babel, polyglossia</code>	text function:
<code>amsmath</code>	Math mode
<code>amsfonts, amssymb</code>	sym module and syntax
<code>geometry, fancyhdr</code>	page function
<code>xcolor</code>	text function:
<code>hyperref</code>	link function
<code>bibtex, biblatex, natbib</code>	cite, bibliography functions
<code>lstlisting, minted</code>	raw function and syntax
<code>parskip</code>	block and par functions
<code>csquotes</code>	Type “ or ‘ and set the text language
<code>caption</code>	figure function
<code>enumitem</code>	list, enum, terms functions

数学公式

要在 Typst 中使用数学模式，只需将您的方程用美元符号括起来。您可以通过在开始和结束美元符号和等式之间放置空格或换行符来进入显示模式。

#Code

The sum of the numbers from

1 to n is:

$\sum_{k=1}^n k = (n(n+1))/2$

#Demo

The sum of the numbers from 1 to n is:

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

数学模式的工作方式与普通标记或代码模式不同。单个字符和任意数量的数字会被显示为数学变量和数值（取决于你的方程式），而多个连续的非数字字符将被解释为Typst变量。

正如你在上面的例子中看到的，Typst在数学模式下预先定义了很多有用的变量。所有的希腊字母和一些希伯来字母都由它们的名字来解决。请参考符号页或使用自动完成面板来检查哪些符号是可用的。符号的替代形式和相关形式通常可以通过在句点后附加一个修饰语来选择。例如，`arrow.l.squiggly` 插入了一个左旋的箭头。如果你想在你的表达式中插入多字母文本，请用双引号将其括起来：

你可以用`<=`、`>=`和`->`等速记符号输入许多符号。同样，定界符也会为其表达式自动缩放，就像LaTeX中隐含插入的 `left` 和 `right` 命令一样。你可以使用`lr`函数自定义定界符的行为。

Typst会自动将斜线/周围的术语设置为分数，同时尊重运算符优先级。所有的圆括号都会出现在输出中，而不会因为分数而出现多余的括号。

下标和上标在 Typst 和 LaTeX 中的工作方式类似。输入 `x^2` 会产生一个上标， `x_2` 会产生一个下标。如果您想在下标或上标中包含多个值，请将它们的内容括在括号中： `$x_{(a \rightarrow \epsilon)}$` 。就像您可以在不键入 `#` 或 `/` 的情况下插入变量一样，您也可以使用“裸”函数。

上面的例子使用 `cases` 函数来描述 f 。在 `cases` 函数中，参数使用逗号分隔，并且参数也被解释为数学。如果您需要将参数解释为 Typst 值，请在它们前面加上 `#`。

您可以在数学模式下使用所有 Typst 函数并插入任何内容。如果您希望它们正常工作，在参数列表中使用代码模式，您可以在它们的调用前加上 `#`。没有人能阻止您再使用矩形或表情符号作为变量。

如果您想直接输入您的数学符号作为 Unicode，那也是可能的！数学调用可以使用二维参数列表；作为分隔符。最常见的用途是创建矩阵的 `mat` 函数。

#Code

```
$ delta "if" x <= 5 $

$ f(x) = (x + 1) / x $

$ f(x, y) := cases(
  1 "if" (x dot y)/2 <= 0,
  2 "if" x "is even",
  3 "if" x in NN,
  4 "else",
) $

$ (a + b)^2
= a^2
+ text(fill: #maroon, 2 a b)
+ b^2 $

$ sum^10_(□=1)
#rect(width: 4mm, height: 2mm)/□
= 🇲🇹 maltese $

$ mat(
  1, 2, ..., 10;
  2, 2, ..., 10;
  dots.v, dots.v, dots.down, dots.v;
```

```
10, 10, ..., 10;
) $
```

#Demo

$$\delta \text{ if } x \leq 5$$

$$f(x) = \frac{x+1}{x}$$

$$f(x, y) := \begin{cases} 1 & \text{if } \frac{x \cdot y}{2} \leq 0 \\ 2 & \text{if } x \text{ is even} \\ 3 & \text{if } x \in \mathbb{N} \\ 4 & \text{else} \end{cases}$$

$$(a+b)^2 = a^2 + 2ab + b^2$$

$$\sum_{=1}^{10} \square = \text{☉} \text{☿}$$

$$\begin{pmatrix} 1 & 2 & \dots & 10 \\ 2 & 2 & \dots & 10 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix}$$

“盗版”LaTeX

在 LaTeX 中设置的论文具有明确无误的外观。这主要是由于它们的字体、Computer Modern、对齐、窄行距和宽边距。这应该是一个很好的起点！如果您想更进一步，为什么不创建一个可重复使用的模板呢？

- sets wide margins
- enables justification, tighter lines and first-line-indent
- sets the font to “New Computer Modern”, an OpenType - derivate of Computer Modern for both text and code blocks
- disables paragraph spacing
- increases spacing around headings

#Code

```
#set page(margin: 1.75in)
#set par(leading: 0.55em, first-line-indent: 1.8em, justify: true)
#set text(font: "New Computer Modern")
#show raw: set text(font: "New Computer Modern Mono")
#show par: set block(spacing: 0.55em)
#show heading: set block(above: 1.4em, below: 1em)
```

局限性

尽管 Typst 可以成为当今许多人的 LaTeX 替代品，但 Typst 仍然（尚）不支持某些功能。以下是它们的列表，其中在适用的情况下包含可能的解决方法。

- 图表绘制Native charts and plots. LaTeX users often create charts along with their documents in PGF/TikZ. Typst does not yet include tools to draw diagrams, but the community is stepping up with solutions such as `typst-canvas`, `typst-plot`, and `circuitytypst`. You can add those to your document to get started with drawing diagrams.
- 页边距Change page margins without a pagebreak. In LaTeX, margins can always be adjusted, even without a pagebreak. To change margins in Typst, you use the `page` function which will force a page break. If you just want a few paragraphs to stretch into the margins, then reverting to the old margins, you can use the `pad` function with negative padding.
- 浮动图片Floating figures. The `figure` command of LaTeX will smartly choose where to place a figure, be it on the top or bottom of the page, or a dedicated figure page. Typst's figure will always appear at the spot where they have been inserted in the markup. While this behavior can save some headache, it is often cumbersome to manually place figures. We will be adding this feature soon!
- 包含PDF Include PDFs as images. In LaTeX, it has become customary to insert vector graphics as PDF or EPS files. Typst supports neither format as an image format,

but you can easily convert both into SVG files with online tools or Inkscape. We plan to add automatic conversion for these file formats to the Typst web app, too!

- 分页优化Page break optimization. LaTeX runs some smart algorithms to not only optimize line but also page breaks. While Typst tries to avoid widows and orphans, it uses less sophisticated algorithms to determine page breaks. You can insert custom page breaks in Typst using `#pagebreak(weak: true)` before submitting your document. The argument `weak` ensures that no double page break will be created if this spot would be a natural page break anyways. You can also use `#v(1fr)` to distribute space on your page. It works quite similar to LaTeX's `vfill`.
- 参考数目格式Bibliographies are not customizable. In LaTeX, the packages `bibtex`, `biblatex`, and `natbib` provide a wide range of reference and bibliography formats. You can also use custom `.bbx` files to define your own styles there. Typst only supports a small set of citation styles at the moment, but we want to build upon this by supporting Citation Style Language (CSL), an XML-based format backed by Zotero that allows you to describe your own bibliography styles.

Part II 简明教程

第 1 章 内容对齐

#Func

```
// 水平/垂直对直内容
align(
  set alignment 2d alignment,
  // 沿两个轴排列 横向排列: start end left center right
  // 竖向排列: top horizon bottom, 使用+号实现横向竖向排列设置
  content,
) -> content
```

#Code

```
#align(center)[#lorem(10)]
#align(right)[#lorem(10)]
#align(left)[#lorem(10)]
#align(center+top)[#lorem(10)]
```

#Demo

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

第 2 章 图片插入

#Func

带有标题的图片及引用

```
figure(
  content, // 图片内容
  set caption: `none` `content`, // 图片标题
  set numbering: `none` `string` `function`, // 图片标号
)
```

```

    set gap: `length`, // 图片与标题之间的距离
  ) -> content

```

#Func

图片

```

image(
  `string`, // 图片路径
  set width: `auto` `relative length`, // 图片宽度
  set height: `auto` `relative length`, // 图片高度
  set alt: `none` `string` // 图片描述
  set fit: `string`,
  // 如何自动调节: cover (默认, 完全覆盖整个区域)、
  // contain (完全包含整个区域)、stretch (拉伸图象以完全填满)
) -> content

```

#Code

```

@gege shows the right pose of playing basketball

#figure(
  image("1.png",height:20%),
  caption: [
    Playing basketball.
  ],
  numbering: "1",
) <gege>

```

#Demo

Figure 1 shows the right pose of playing basketball



Figure 1: Playing basketball.

第 3 章 盒子创建

盒子的创建对于页面布局十分有用。

#Func

内联级的container，除了公式、文字、box之外所有的元素都是block级的，不能出现在一个段落中

box可以用来将元素整合到一个段落中

```
box(
  set width: `auto` `relative length` `fraction`, // 盒子宽度
  set height: `auto` `relative length`,           // 盒子高度
  set baseline: `relative length`,                 // 盒子基线
  set fill: `none` `color`,                       // 背景颜色
  set stroke: `none` `length` `color` `dictionary` `stroke`, // 盒子边界
  set radius: `relative length` `dictionary`,      // 盒子圆角半径
  set inset: `relative length` `dictionary`,        // 内容距离盒子边界距离
  set outset: `relative length` `dictionary`,      // 盒子外扩值
  set `none` `content`,
) -> content
```

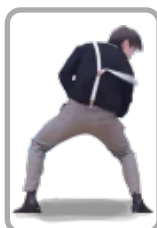
#Code

Image with baseline 50%:

```
#box(width:auto,
  height: 10%,
  baseline: 50%,
  stroke: gray+2pt,
  radius:5pt,
  inset: 1pt,
  outset: 1pt,
  image("1.png")
)
```

#Demo

Image with baseline 50%:



第 4 章 强制分栏

#Func

强制分栏,当在单列布局或页面的最后一列中使用时,该函数将表现得像 `pagebreak()`。否则,分栏后的内容将被放置在下一栏。

在此处可以看到Typst处理中文文字间距时还不是很完美空格判定有缺陷。

```
colbreak(
  set weak: boolean
) -> content
```

#Code

Preliminary findings from
our ..

使用 `\#colbreak()` 强制换列

```
#colbreak()
```

Through rigorous
experimentation ..

需要注意到,此处的省略号只有两个,那是因为如果使用三个省略号中文将会出现乱码情况。

#Demo

Preliminary findings from our ongoing research project have revealed a hitherto unknown phenomenon of extraordinary significance.

使用 `#colbreak()` 强制换列

Through rigorous experimentation and analysis, we have discovered a hitherto uncharacterized process that defies our current understanding of the fundamental laws of nature.

第 5 章 设置列数

#Func

`#pagebreak()` 可以强制换页

`#set page(columns:2)` 可以设置页面列数分栏,将一个区域分割成多个相同大小的列,不会分配列的高度,列可以跨页断开

```
columns(
  set integer, // 列数目
  set gutter: `relative length`,
  // 每列之间的间距
  content,
) -> content
```

#Code

```
#box(height:150pt,columns(2, gutter:
20pt)[
```

```
  #set par(justify: true)
```

This research was funded by the National Academy of Sciences. NAOs provided support for field tests and interviews with a grant of up to USD 40.000 for a period of 6 months.

```
]
```

```
)
```

In recent years, deep learning

has increasingly been used to solve a variety of problems.

#Demo

This research for field tests and was funded by interviews with a the National grant of up to Academy of Sci- USD 40.000 for ences. NAoS a period of 6 provided support months.

In recent years, deep learning has increasingly been used to solve a variety of problems.

第 6 章 有序列表

`enum`用于创建有序无序列表以及连续编号,`enum`函数也有专门的语法糖: 以`+`起行,创建一个自动编号的枚举项目. 以数字和`.`开头的一行将创建一个明确编号的枚举项目. 枚举项目可以包含多个段落和其他块级内容.所有缩进超过一个项目的`+`或`.`的内容都成为该项目的一部分。

#Code

```
enum(
  set tight: `boolean`,
  // 紧凑显示
  set numbering: `string`
`function`,
  // 如何编号
  set start: `integer`,
  // 编号起始值
  set full: `boolean`,
  // 是否显示全部编号
  set indent: `length`,
  // 每个元素的缩进
  set body-indent: `length`,
  // 编号与内容之间的间距
  set spacing: `auto`
`relative length` `fraction`,
  // 行间距
  ..contentarray,
) -> content
```

#Code

```
#block(fill:gray,width:100%,
  inset:5pt,radius:5pt)[
  Automatically numbered:
  + Preparations
  + Analysis
  Manually numbered:
  2. What is the first step?
  5. I am confused.
  + Moving on ...
  Function call.
  #enum[First][Second]
]
```


#Demo

Automatically numbered:

1. Preparations
2. Analysis

Manually numbered:

2. What is the first step?
5. I am confused.
6. Moving on ...

Function call.

1. First
2. Second

```
a) ")
+ Different
+ Numbering
  + Nested
  + Items
+ Style

#set enum(numbering: n =>
super[#n])
+ Superscript
+ Numbering!

#enum(
  start: 3,
  numbering: "a.",
  [Skipping],
  [Ahead],
)
]
```

#Code

```
#block(fill:green,width:100%,
  radius:5pt)[
  #set enum(numbering: "a)")

  + Starting off ...
  + Don't forget step two
]
```

#Demo

- a) Starting off ...
- b) Don't forget step two

#Code

```
#block(fill:rgb("#b1f2eb"),
  width:100%,
  inset:5pt,
  radius:5pt)[
  #set enum(numbering: "1.
```

#Demo

- 1) Different
- 2) Numbering
 - a) Nested
 - b) Items
- 3) Style
 - ¹ Superscript
 - ² Numbering!
- c. Skipping
- d. Ahead

#Code

```
#block(fill:rgb("#b1f2eb"),
      width:100%,inset:5pt,
      radius:5pt)[
  #enum(
    start: 3,
    numbering: "a.",
    [Skipping],
    [Ahead],
  )
]
```

c. Skipping

d. Ahead

#Code

```
#block(fill:rgb("#b122eb"),
      width:100%,inset:5pt,
      radius:5pt)[
  #set enum(numbering: "1.a)",
  full: true)
  + Cook
    + Heat water
    + Add integredients
  + Eat
]
```

1) Cook

1.a) Heat water

1.b) Add integredients

2) Eat

第 7 章 网格排版

在网格中排版内容. `grid` 允许将内容安排在一个 `grid` 中. 可以定义行和列的数量, 以及它们之间的间距. 有多种列和行的模式, 可以用来创建复杂的布局.

#Func

```
grid(  
  set columns: `auto` `integer` `relative length` `fraction` `array`,  
  // 列数  
  set rows: `auto` `integer` `relative length` `fraction` `array`,  
  // 行数  
  set gutter: `auto` `integer` `relative length` `fraction` `array`,  
  // 行或列间距  
  set column-gutter: `auto` `integer` `relative length` `fraction` `array`,  
  // 列间距, 优先级高于 gutter  
  set row-gutter: `auto` `integer` `relative length` `fraction` `array`,  
  // 行间距, 优先级高于 gutter  
  ..content,  
) -> content
```

#Code

```
#let cell = rect.with(  
  inset: 8pt,  
  fill: rgb("e4e5ea"),  
  width: 100%,  
  radius: 6pt  
)  
#grid(  
  columns: (60pt, 1fr, 60pt),  
  rows: (60pt, auto),  
  gutter: 3pt,  
  cell(height: 100%)[Easy to learn],  
  cell(height: 100%)[Great output],  
  cell(height: 100%)[Intuitive],  
  cell[Our best Typst yet],  
  cell[  
    Responsive \ design in \ print  
    for everyone
```

```
],
cell[One more thing...],
)
```

#Demo

Easy to
learn

Great output

Intuitive

Our best
Typst yet

Responsive
design in
print for everyone

One more
thing...

7.1 Figure、Grid结合绘制子图

#Code

```
#let subfigure(body, caption: "", numbering: "(a)") = {
  let figurecount = counter(figure)
  let subfigurecount = counter("subfigure")
  let subfigurecounterdisply = counter("subfigurecounter")
  let number = locate(loc => {
    let fc = figurecount.at(loc)
    let sc = subfigurecount.at(loc)
    if fc == sc.slice(0, -1) {
      subfigurecount.update(
        fc + (sc.last()+1,)
      )
      subfigurecounterdisply.update((sc.last()+1,))
    } else {
      subfigurecount.update( fc + (1,))
      subfigurecounterdisply.update((1,))
    }
    subfigurecounterdisply.display(numbering)
  })
  body
```

```
v(-.65em)
if not caption == none {
  align(center)[#number #caption]
}
}
#figure(
  grid(columns: 3, gutter: 2em,
    subfigure(image("1.png", width: 50%)),
    subfigure(image("1.png", width: 50%)),
    subfigure(image("1.png", width: 50%), caption: "Test Caption")
  ),
  caption: "Test caption"
)
#v(2em)
#figure(
  grid(
    columns: (1fr, 1fr, 1fr),
    rows: (auto, auto),
    gutter: 1pt,
    image("1.png",width:50%),
    image("1.png",width:50%),
    image("1.png",width:50%),
    image("1.png",width:50%),
    image("1.png",width:50%),
    image("1.png",width:50%),
  ),
  numbering: "1",
  caption: [
    SubFigures.
  ]
)
```

#Demo



(a)



(b)



(c) Test Caption

Figure 2: Test caption



Figure 3: SubFigures.

Part III 手册

第 8 章 自定义格式

Typst 有灵活的排版格式. 通过使用 `set` 规则, 可以设置元素的基本性质. 但是对于稀奇古怪的要求, 内置的性质可能无法达到要求, 因此 Typst 可以使用 `show` 规则重新定义元素的显示效果. 如果只想设定在某些位置有效, 可以将其置于块内 `#[]`. 有时像重复使用一组规则, 此时可以使用 `set-if` 规则.

8.1 `set` 规则

#Code

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

This one is not:
- Bullet

\

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

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

#Demo

This list is affected:

- Dash

This one is not:

- Bullet
- Food today?
- Work deadline

8.2 show规则

使用show规则, 可以深度定义一种元素的外观. show规则的最基本形式是show-set规则. show 函数: set规则. 这使设置规则仅适用于所选元素. 在下面的示例中, 标题变为深蓝色, 而所有其他文本保持黑色. 使用 show-set 规则, 您可以混合和匹配来自不同函数的属性以实现许多不同的效果. 但它们仍然限制您使用Typst 中预定义的内容. 为了获得最大的灵活性, 您可以编写一个show规则来定义如何从头开始格式化元素. 要编写这样的show规则, 请将:后面的 set 规则替换为任意函数. 此函数接收元素并可以返回任意内容. 传递给函数的元素上有不同的字段. 下面, 定义一个显示规则, 用于格式化标题. show规则和set规则一样, 一旦设定, 一直使用到结束, 可以使用#[]限定使用范围. 除了函数之外, show右边还可以使用直接替换原色的文字字符串或者内容块. 除了函数之外, show左侧也可使用其他的选择器定义这些转换的适用范围:

#Func

- Everything: show: rest => ..转换所有内容, 有助于将更复杂的布局应用于整个文档
- Text: show "Text": ..
文本样式、转换、替换
- Regex: show regex("\w+"): .
使用正则灵活选择和转换文本
- Function with fields: show heading.where(level: 1): ..
转换指定fields的袁术, 举个例子: 只改变一级标题
- Label: show <intro>: ..
选择和转换指定标签的元素

#Code


```

#[
  #show heading: set text(navy)
  === This is navy-blue
  But this stays black
]
#[
  #set heading(numbering: "(I)",outlined:false)
  #show heading: it => block[
    #set align(center)
    #set text(font: "Inria Serif")
    \~ #emph(it.body)
    #counter(heading).display() \~
  ]
  = Dragon
  With a base health of 15, the dragon is the most powerful
  creature.
  = Manticore
  While less powerful than the dragon, the manticore gets extra
  style points.
]
#[
  We started Project in 2019 and are still working on it. Project
  is progressing badly. #parbreak()
  #show "Project":smallcaps
  #show "badly": "great"
  We started Project in 2019 and are still working on it. Project
  is progressing badly.
]

```

8.2.1 This is navy-blue

But this stays black

#Demo

~ **Dragon (IX)** ~

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

~ **Manticore (X)** ~

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

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.

第 9 章 Typst 脚本语言

Typst 拥有强大的脚本语言(应该是得益于 rust). 可以使用代码自动格式化文档以及创建更加复杂的样式.

9.1 表达式

在 Typst 中, 标记和代码合而为一. 除了最常见的元素外, 其余所有元素都是用函数创建的. 为了尽可能方便, Typst 提供了紧凑的语法来将代码表达式嵌入到标记中: 表达式以井号 (#) 引入, 表达式完成后恢复正常的标记解析. 如果字符将继续表达式但应解释为文本, 则可以强制以分号 (;) 结束表达式.

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

#Demo

Hello



5

一些表达式与主题标签语法不兼容(例如二元运算符表达式). 要将它们嵌入到标记中, 您可以使用括号, 如 `$(1 + 2)$`.

注意空格的存在!!

9.2 代码块 Blocks

Typst 提供了两个 blocks:

- 代码块: { let x = 1; x + 2 }

编写代码时，希望拆分为多个语句、创建一些中间变量等。代码块让您可以在需要一个表达式的地方编写多个表达式。代码块中的各个表达式应该用换行符或分号分隔。代码块中各个表达式的输出值被连接在一起以确定块的值。

- 内容块: [Hey there!] 使用内容块，您可以将标记/内容作为编程值处理，将其存储在变量中并将其传递给函数。内容块由方括号分隔并且可以包含任意标记。内容块产生内容类型的值。可以将任意数量的内容块作为尾随参数传递给函数。也就是说，list([A], [B]) 等价于 list[A][B]。

内容和代码块可以任意嵌套。在下面的示例中，[hello] 与 a + [the] + b 的输出相结合产生 [hello from the **world**]。

#Code

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

#Demo

hello from the **world**

9.3 Let定义变量

使用 let 绑定来定义变量。变量被赋予 = 符号后的表达式的值。赋值是可选的，如果没有赋值，变量将被初始化为none。let 关键字也可用于创建自定义命名函数。可以为包含块或文档的其余部分访问 Let 绑定。

#Code

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

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

#Demo

This is Typst's documentation. It explains Typst.

Sum is 5.

Let可以用于解构数组以及字典

#Code

```
#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.
]
```

#Demo

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.

也可以使用下划线丢弃解构模式中的元素

#Code

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

#Demo

The y coordinate is 2.

9.4 条件判断语句

判断语句，可以根据条件来显示和计算不同的内容。目前 Typst 支持 `if`，`else if` 和 `else` 语句。

#Code

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

#Demo

This is shown

对于判读语句来说，每个分支都有一个代码块或者内容块作为主体。

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

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

9.5 循环语句

使用 `loops` 可以重复计算或者显示内容。Typst 支持两种格式 `for` 和 `while`。前者遍历指定的集合，而后者只要满足条件就进行迭代。就像块一样，循环将每次迭代的结果连接成一个值。下例中，`for` 循环创建的三个句子连接在一起成为一个内容值，而 `while` 循环中长度为 1 的数组连接在一起成为一个更大的数组。

#Code

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

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

#Demo

A is a letter.

B is a letter.

C is a letter.

(3, 5, 9, 17)

- `for letter in "abc" {...}`

遍历字符串的字符。（从技术上讲，迭代字符串的字素簇。大多数时候，一个字素簇只是一个字符/代码点。但是，由多个代码点组成的标志表情符号等一些结构仍然只是一个簇。）

- `for value in array {...}`

迭代数组中的值。还可以提供每个值的索引。

- for pair in dict {...}

for (key, value) in dict {...}

迭代字典的键值对

为了控制循环的执行，Typst 提供了 `break` 和 `continue` 语句。前者提前退出循环，而后者跳到循环的下一迭代。

#Code

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

  letter
}
```

#Demo

abc

循环函数的主体可以是代码块或者内容块。

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

9.6 字段访问

可以使用 `.` 访问值上的字段：

- 拥有指定键的字典 `dictionary`
- 具有指定修饰符的符号 `symbol`
- 特殊定义的模块 `module`
- 指定域的内容 `content`

#Code

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

#Demo

Hello



9.7 方法Methods

Method是一类与特定类型耦合的函数。它使用相同的.表示法对其类型的值进行调用：`value.method(..)`。Type文档列出了每个内置类型的可用method。目前还不能定义自己的方法。

#Code

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

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

#Demo

4

3

a — b — c

Methods是Typst中唯一可以修改调用值的函数。

9.8 模块

可以将Typst项目拆分为多个modules文件，同时module可以使用多种方式调用：

- Including: include “bar.typ”

判断bar.typ是否存在，并返回结果内容

- Import: import “bar.typ”

判断文件是否存在，并将module当作bar导入当前scope

- Import items: import “bar.typ”: a, b

判断bar.typ是否存在，提取变量 a 和 b 的值（需要在 bar.typ 中定义，例如通过 let 绑定）并在当前文件中定义它们。用 * 导入模块中定义的所有变量。

除了导入路径，还可以使用module值

#Code

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

#Demo



9.9 运算符

下表列出了所有可用的一元和二元运算符，具有效果、元数（一元、二元）和优先级。

操作符	优先级
#	7
+	7
*	6
/	6
-	5
==	4

<code>!=</code>	4
<code><</code>	4
<code><=</code>	4
<code>></code>	4
<code>>=</code>	4
<code>in</code>	4
<code>not in</code>	4
<code>not</code>	3
<code>and</code>	3
<code>or</code>	2
<code>=</code>	1
<code>+=</code>	1
<code>-=</code>	1
<code>*=</code>	1
<code>/=</code>	1

第 10 章 Typst 内置类型

Typst 使用不同类型的值设置文档样式：指定元素大小的长度、文本和形状的颜色等等。除了非常基本的数值类型和编程语言中已知的典型类型之外，Typst 还提供了一种特殊的内容类型。这种类型的值可以包含您可以输入到文档中的任何内容：文本、标题和形状等元素以及样式信息。在 Typst 的某些地方使用了更专业的数据类型。这里没有列出所有这些，而是在相关的地方进行了解释。

10.1 none

缺省值, `none` 类型只有一个值: `none`。当插入到文档中时, 它是不可见的。这也是空代码块产生的值。它可以与任何值结合, 产生另一个值。

10.2 auto

自动识别类型

10.3 boolean

布尔值, `true` or `false`

10.4 integer

整数。该数字可以是负数、零或正数。由于 Typst 使用 64 位存储整数, 因此整数不能小于 -9223372036854775808 或大于 9223372036854775807。

10.5 float

浮点数。有限精度表示实数。Typst 使用 64 位来存储浮点数。在需要浮点数的地方, 您也可以传递一个整数。

#Code

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

#Demo

```
3.14  
10000  
2.5
```

10.6 length

大小或距离, 可能用上下文单位表示。Typst 支持以下长度单位:

- Points: 72pt
- Millimeters: 254mm

- Centimeters: 2.54cm
- Inches: 1in
- Relative to font size: 2.5em

#Code

```
#rect(width: 20pt)
#rect(width: 2em)
#rect(width: 1in)
```

#Demo

10.7 angle

角度，支持如下单位：

- Degrees: 180deg
- Radians: 3.14rad

#Code

```
#rotate(10deg)[Hello there!]
```

#Demo

Hello there!

10.8 ratio

整体的比例。数字+百分号。

#Code

```
#set align(center)
#scale(x: 150%)[
  Scaled apart.
]
```

#Demo

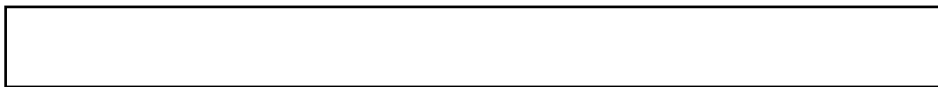
Scaled apart.

10.9 relative length

与某个已知长度相关的长度。这种类型是长度与比值的组合。它是长度和比率的加减法结果。在需要相对长度的地方，您也可以使用裸长度或比率。

```
#Code
#rect(width: 100% - 50pt)
```

#Demo



10.10 fraction

定义布局中剩余空间的分布方式。

```
#Code
Left #h(1fr) Left-ish #h(2fr) Right
```

#Demo

Left

Left-ish

Right

10.11 color

颜色空间的设定，支持三个空间：

- sRGB
- CMYK

- D65

同时也内嵌了如下色彩: black, gray, silver, white, navy, blue, aqua, teal, eastern, purple, fuchsia, maroon, red, orange, yellow, olive, green, and lime.

10.11.1 Methods

10.11.1.1 lighten

增亮颜色

#Code

```
value.lighten(ratio) -> color
```

10.11.1.2 darken

使颜色变暗

#Code

```
value.darken(ratio) -> color
```

10.11.1.3 negate

反色

#Code

```
value.negate() -> color
```

10.11.2 sRGB

创建RGB色彩，颜色在sRGB空间中指定

#Func

```
rgb(
  hex: string, // 16进制色彩表示，与以下参数不同时出现
  red: integer ratio, // 红色比率
  green: integer ratio, // 绿色比率
  blue: integer ratio, // 蓝色比率
  alpha: integer ratio, // 透明度
) -> color
```

#Code

```
#square(fill: rgb("#b1f2eb"))
#square(fill: rgb(87, 127, 230))
```

```
#square(fill: rgb(25%, 13%, 65%))
#text(16pt, rgb("#239dad"))[ *Typst* ]
```

#Demo



Typst

10.11.3 CMYK

创建 CMYK 颜色。如果您想针对特定打印机，这很有用。为显示预览转换为 RGB 可能与您的打印机再现颜色的方式不同。

```
#Func
cmyk(
  cyan: ratio,
  magenta: ratio,
  yellow: ratio,
  key: ratio,
) -> color
```

#Code

```
#square(
  fill: cmyk(27%, 0%, 3%, 5%)
)
```

#Demo



10.11.4 D65

创建灰度图

#Func

```
luma(integer ratio) -> color
```

#Code

```
#for x in range(250, step: 50) {  
  box(square(fill: luma(x)))  
}
```

#Demo

10.12 Datetime

`datetime` 表示日期、时间或两者的组合。可以通过使用 `datetime` 函数指定自定义日期时间或使用 `datetime.today` 获取当前日期来创建。（仅翻译部分）

#Code

```
#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]"
)
```

#Demo

2020-10-04

y:20

18:02:23

h:06PM

10.13 符号

Unicode符号, Typst定义了常用符号, 从而轻松输入符号。这些符号在模块中定义, 可以使用字段访问。

- 通用符号在`sym module`中定义
- Emoji在`emoji module`中定义

更进一步可以使用`symbol`函数自定义符号

文末附录列出了Typst内置的通用符号。

#Func

```
// 自定义符号
// 可以只是一个由单个字符组成的字符串, 表示无修饰符变体, 也可以是一个数组, 其中包含两个指定修饰符和符号的字符串。
// 各个修饰符应该用点分隔。显示符号时, Typst 从具有所有附加修饰符和最少数量的其他修饰符的变体中选择第一个。
symbol(..string array) -> symbol
```

#Code

```
#sym.arrow.r \
#sym.gt.eq.not \
$gt.eq.not$ \
#emoji.face.halo
```

#Demo

→

≥

≠

**#Code**

```
#let envelope = symbol(
  "✉",
  ("stamped", "📧"),
  ("stamped.pen", "✍"),
  ("lightning", "⚡"),
  ("fly", "🐝"),
)
```

#envelope

#envelope.stamped

#envelope.stamped.pen

#envelope.lightning

#envelope.fly

#Demo

许多符号有不同的变体，可以通过在修饰符后面附加点符号来选择。修饰符的顺序无关紧要。访问符号模块的文档页面并单击符号以查看其可用变体。

#Code

```
$arrow.l$ \
$arrow.r$ \
$arrow.t.quad$
```

#Demo

←

→

⌵

10.14 字符串

您可以使用 `for` 循环遍历字符串。字符串可以用 `+` 运算符相加、连接在一起并与整数相乘。Typst 提供了用于字符串操作的实用方法。（`split`, `trim`, `replace`）所有长度和索引均以 UTF-8 字节表示。

#Code

```
#"hello world!" \
#"\"hello\n  world\!"\" \
#"1 2 3".split() \
#"1,2;3".split(regex("[,;]")) \
#(regex("\\d+") in "ten euros") \
#(regex("\\d+") in "10 euros")
#str(10) \
#str(2.7) \
#str(1e8) \
```

#Demo

hello world!

"hello

world"!

("1", "2", "3")

("1", "2", "3")

false

true

10

2.7

100000000

一些转义序列：

- \\ 空格
- \" 引用
- \n 新行
- \r 回车
- \t tab
- \u{1f600} 16进制转义序列

10.14.1 Methods

#Func

```
// 用法和编程语言相似
// 获取字符串长度
value.len() -> integer
// 获取第一个字符
value.first() -> any
// 获取最后一个字符
value.last() -> any
// 获取指定index的字符
value.at(integer) -> string
// 获取字符串切片
value.slice(start:integer,end:integer,count: integer,) -> string
// 将字符串的单字符作为子字符串数组返回。
value.clusters() -> array
// 将字符串的 Unicode 代码点作为子字符串数组返回。
value.codepoints() -> array
// 是否包含某些字符，可以使用正则
value.contains(string regex) -> boolean
// 是否以指定字符开始
value.starts-with(string regex) -> boolean
// 是否以指定字符结束
value.ends-with(string regex) -> boolean
// 在字符串中搜索指定的字符并返回第一个匹配项作为字符串，如果没有匹配项则返回无。
value.find(string regex) -> stringnone
// 搜寻指定字符并返回第一个匹配项的索引值
value.position(string regex) -> integer none
// 字符串匹配
```

```

value.match(string regex) -> dictionary none
value.matches(string regex) -> array
// 替换字符串
value.replace(pattern: string|regex,replacement: string,count: integer,) -> string
// 去除匹配项
value.trim(pattern: string | regex, at: alignment,repeat: boolean,) -> string
// 拆分字符串
value.split(string|regex) -> array
#str.from-unicode(97)
str.from-unicode(
  integer
) -> string
str.to-unicode(
  string
) -> integer

```

#Code

```

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

```

#Demo

97

(97, 768)

10.15 文本

文档内容是 Typst 的核心。编写的所有标记和您调用的大多数函数都会产生内容值。可以通过在方括号[]中来创建内容值。这也是将内容传递给函数的方式。

#Code

```

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

```

#Demo

Type of **Hello!** is content

10.15.1 Methods

#Func

// func()函数。此函数可用于创建此内容中包含的元素。

// 它可以用于元素的设置和显示规则。可以与全局函数进行比较以检查您是否具有特定种类的元素。

```
value.func() -> function
```

// 内容是否含有指定字段

```
value.has(string) -> boolean
```

// 访问指定字段

```
value.at(string) -> any
```

// 查询内容的位置。

```
value.location() -> location
```

// 返回此内容的字段。

```
value.fields()
```

#Code

```
#rect(
  width: 10cm,
  height: 10cm,
).fields()
```

#Demo

```
(width: 283.46pt, height: 283.46pt)
```

10.16 数组

创建圆括号包围，逗号分隔的数组。数组内的值不需要具有相同的类型。使用 `.at()` 方法访问和更新数组项。索引从 `0` 开始，同时支持负索引。可以使用 `loop` 遍历数组，数组可以使用 `+` 相加（类似于rust语法）。空数组写作 `()`

#Code

```
#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 "))
```

#Demo

1

2

4

```
(3, 7, -3)
```

```
(3, 7, 4, 3, 2)
```

```
(2, -3, 4, 7, 3)
```

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

A, B and C

10.16.1 Methods

#Func

```
// rust语法!
// 数组长度
value.len() -> integer
// 数组第一项
value.first() -> any
// 数组最后一项
value.last() -> any
// 数组指定位置值
value.at(index: integer) -> any
// 在最后添加一项
value.push(value: any)
// 删除最后一项并返回
value.pop() -> any
// 在指定位置插入
value.insert(index: integer, value: any,)
```

```
// 移除指定位置值
value.remove(index: integer) -> any
// 获得数组切片
value.slice(start: integer, end: integer, count: integer,) -> array
// 是否包含指定值
value.contains(value: any) -> boolean
// 根据函数搜寻值并返回第一个匹配项
value.find(function) -> anynone
// 根据函数搜寻值并返回index
value.position(function) -> integer none
// 过滤数组并创建为新数组
value.filter(function) -> array
// 根据目标函数创建新数组
value.map(function) -> array
// 返回新的数组包含值和index
// 返回的数组有长度为2的数组(index,value)对组成。可以使用let或者for函数解构
value.enumerate() -> array
// 用另一个数组压缩数组。如果两个数组的长度不等，它只会压缩到较小数组的最后一个元素，其余元素将被忽略。
value.zip(array) -> array
// 使用累加将所有项合并为一个值
value.fold(any,function,) -> any
// 数组加和
value.sum(default:any) -> any
// Calculates the product all items (works for any types that can be multiplied)
value.product(default:any) -> any
// 只要一个值满足函数返回true就返回true
value.any(function) -> boolean
// 所有值满足函数返回true就返回true
value.all(function) -> boolean
// 将数组展开
value.flatten() -> array
// 将数组反向排列
value.rev() -> array
// 将所有项合并为一个数组
value.join(separator: any,last: any,) -> any
// 排序
// key:如果给定，则将函数用于数组中的元素以筛选排序的键
value.sorted(key:function) -> array
```


10.17 字典

字典：键值对。通过在大括号中使用逗号分隔的键：值来构造字典。这些值不必是相同的类型。字典在概念上类似于数组，但它是由字符串索引而不是整数索引的。可以使用`.at()`方法访问和创建字典条目。如果知道`key`，那么您也可以使用字段访问表示法`.key`来访问对应`value`。字典可以使用`+`运算符添加并连接在一起。要检查字典中是否存在关键字，请使用`in`关键字。可以使用`for`循环来迭代字典中的键值对。字典总是按键排序。由于空括号已经产生了一个空数组，因此必须使用特殊的`(:)`语法来创建一个空字典。

#Code

```
#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)
```

#Demo

Typst

3

```
("name", "born", "launch")
```

```
("Typst", 2019, 20)
```

2019

```
true
```

10.17.1 Methods

#Func

```
// 字典长度
value.len() -> integer
// 返回与字典中指定键关联的值。
value.at(string) -> any
// 插入新的键值对
value.insert(string,any,)
// 返回排序后的所有键
value.keys() -> array
// 返回值
value.values() -> array
// 以成对数组的形式返回字典的键和值。每一对都表示为一个长度为2的数组。
value.pairs() -> array
// 按键名删除键值对
value.remove(key: string) -> any
```

10.18 函数

函数 函数调用是typst的特色，用户可以定义并调用函数来自定义输出格式。

可以通过指定参数列表后跟 `=>` 和函数体来创建匿名函数。如果您的函数只有一个参数，则参数列表周围的括号是可选的。匿名函数主要用于显示规则。

#Code

```
// Call a function.
#list([A], [B])

// Named arguments and trailing
// content blocks.
#enum(start: 2)[A][B]

// Version without parentheses.
#list[A][B]
```

#Demo

- A

- B
- 2. A
- 3. B
- A
- B

使用`#let`设置变量，通过`#`调用

#Code

```
#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
]

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

#Demo

Warning:
Danger is imminent!

Warning:**KEEP OFF TRACKS**

once? once?

10.18.1 Methods

#Func

// 返回一个预先应用了给定参数的新函数。

value.with(..any) -> function

// 返回一个选择器，用于过滤属于此函数的元素，其字段具有给定参数的值。

value.where(..fields:any) -> selector

10.19 参数

捕获函数的参数。与内置函数一样，自定义函数也可以采用可变数量的参数。可以指定一个参数接收器sink，它将所有多余的参数收集为 `..sink`。sink值属于arguments类型。它公开了访问位置参数和命名参数的方法，并且可以使用 `for` 循环进行迭代。相反，您可以使用展开运算符将参数、数组和字典展开到函数调用中：`func(..args)`。

#Code

```
#let format(title, ..authors) = [
  *#title* \
  _Written by #(authors
    .pos()
    .join(", ", last: " and "));._
]

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

#Demo**ArtosFlow***Written by Jane and Joe.*

10.19.1 Methods

#Func

```
// 返回位置参数数组。  
value.pos() -> array  
// 返回命名参数字典。  
value.named() -> dictionary
```

10.20 selector

选择文档的元素。

可以使用如下方法构建选择器：

- use an element function
- filter for an element function with specific fields
- use a string or regular expression
- use a `<label>`
- use a location
- call the selector function to convert any of the above types into a selector value and use the methods below to refine it

选择器用于将样式规则应用于元素。您还可以使用选择器在文档中查询某些类型的元素。

此外，您可以将选择器传递给几个 Typst 的内置函数来配置它们的行为。一个这样的例子是大纲，它可以用来改变大纲中列出的元素。

可以使用下面显示的方法组合多个选择器。但是，目前并非所有地方都支持所有类型的选择器。

#Code

```
#locate(loc => query(  
  heading.where(level: 1)  
  .or(heading.where(level: 2)),  
  loc,  
))
```

```
= This will be found  
== So will this  
=== But this will not.
```

#Func

```
value.or(..selector)  
value.and(..selector)  
value.before(selector,inclusive: boolean,)  
value.after(selector,inclusive: boolean,)
```

10.21 模块

#Code

```
#import "utils.typ"  
#utils.add(2, 5)  
  
#import utils: sub  
#sub(1, 4)
```

第 11 章 文本

11.1 lorem

随机生成指定数量的blind text，对于用来排版占位特别有效。

#Func

```
lorem(words: integer) -> string
```

#Code

```
= Blind Text  
#lorem(30)  
  
= More Blind Text  
#lorem(15)
```

#Demo

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

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

11.2 斜体**emph**

将内容设置为斜体以示强调，同时提供了语法糖，使用下划线(`_`)，只对单词有效。

- 如果目前text格式为 `normal`，则变为 `italic`
- 如果已经是 `italic` 或者 `oblique`，则变为 `normal`

#Func

```
emph(content) -> content
```

#Code

```
This is _emphasized._ \
This is #emph[too.]
```

```
#show emph: it => {
  text(blue, it.body)
}
```

```
This is _emphasized_ differently.
```

#Demo

This is *emphasized.*

This is *too.*

This is *emphasized* differently.

11.3 加粗strong emphasis

字体加粗，语法糖: *

#Func

```
strong(
  // 加粗对于字体weight的添加量
  set delta: integer,
  content,
) -> content
```

#Code

```
This is *strong.* \
This is #strong[too.] \

#show strong: set text(red)
And this is *evermore.*

#set strong(delta: 0)
No *effect!*
```

#Demo

This is **strong.**

This is **too.**

And this is **evermore.**

No **effect!**

11.4 换行linebreak

插入换行符，将段落切至下一行，段落末尾的换行符会被忽略。语法糖: \

#Func

```
// justify 是否在断行前对齐行。
// 如果在对齐的文本中发现比 Typst 更好的换行机会，这将很有用。
linebreak(set justify:boolean) -> content
```


#Code

```

*Date:* 26.12.2022 \
*Topic:* Infrastructure Test \
*Severity:* High \

```

#Demo

Date: 26.12.2022

Topic: Infrastructure Test

Severity: High

#Code

```

#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

#Demo

I have manually tuned the
line breaks in this paragraph
for an *interesting* result.

11.5 小写转换lowercase

大小写转换

#Func

```
lower(string content) -> string content
```

#Code

```

#lower("ABC") \
#lower[*My Text*] \
#lower[already low]

```

#Demo

abc

my text

already low

11.6 大写转换uppercase

大小写转换

#Func

upper(string content) -> string content

#Code

```
#upper("abc") \
#upper[*my text*] \
#upper[ALREADY HIGH]
```

#Demo

ABC

MY TEXT

ALREADY HIGH

11.7 上划线overline

文本上划线

#Func

```
overline(
  // 设置上划线样式
  set stroke: auto length color stroke,
  // 设置上划线距离基线距离
  set offset: auto length,
  // 设置上划线长度
  set extent: length,
  // 上划线是否跳过会与字形发生冲突的部分。
```

```

    set evade: boolean,
    content,
  ) -> content

```

#Code

```
#overline[A line over text.]
```

#Demo

A line over text.

#Code

```

#set text(fill: olive)
#overline(
  stroke: green.darken(20%),
  offset: -12pt,
  [The Forest Theme],
)

```

#Demo

The Forest Theme

#Code

```

#overline(offset: -1.2em)[
  The Tale Of A Faraway Line II
]

```

#Demo

The Tale Of A Faraway Line II

#Code

```

#set overline(extent: 4pt)
#set underline(extent: 4pt)
#overline(underline[Typography Today])

```

#DemoTypography Today**#Code**

```
#overline(
  evade: false,
  offset: -7.5pt,
  stroke: 1pt,
  extent: 3pt,
  [Temple],
)
```

#DemoTemple

11.8 下划线underline

下划线用法与上划线一致

#Func

```
underline(
  // 设置下划线样式
  set stroke: auto length color stroke,
  // 设置下划线距离基线距离
  set offset: auto length,
  // 设置下划线长度
  set extent: length,
  // 下划线是否跳过会与字形发生冲突的部分。
  set evade: boolean,
  content,
) -> content
```

#Code

```
Take #underline(
  stroke: 1.5pt + red,
  offset: 2pt,
  [care],
```

```
)

#underline(offset: 5pt)[
  The Tale Of A Faraway Line I
]

#align(center,
  underline(extent: 2pt)[Chapter 1]
)

This #underline(evade: true)[is great].
This #underline(evade: false)[is less great].
```

#DemoTake careThe Tale Of A Faraway Line IChapter 1This is great. This is less great.

11.9 代码块构建raw text/code

可以使用一个`或者三个`构建代码块，三个`组成的代码块后面可以添加指定的语言tag，以自动语法高亮。

#Func

```
raw(
  // 文本
  text: string,
  // 原始文本是否显示为单独的块。
  set block: boolean,
  // 语法高亮显示的语言。
  // 除了 Markdown 中已知的典型语言标签外，它还分别支持 Typst 标记和 Typst 代
  码的“typ”和“typc”标签。
  set lang: nonestring,
) -> content
```

#Code

Adding ``rbx`` to ``rcx`` gives the desired result.

```
\\`\\`rust
fn main() {
    println!("Hello World!");
}
```

#Demo

Adding `rbx` to `rcx` gives the desired result.

```
fn main() {
    println!("Hello World!");
}
```

#Code

// 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
\\`\\`
```

#Demo

15

#Code

```
// 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.

\\`\\`bash
rg "Hello World"
\\`\\`
```

#Demo

With `rg`, you can search through your files quickly.

```
rg "Hello World"
```

11.10 small capitals

以小写字母显示文本。

注意：这会为字体启用 OpenType smcp 功能。并非所有字体都支持此功能。有时小型大写字母是专用字体的一部分，有时它们根本不可用。未来该功能将支持选择专用smallcaps字体，以及从普通字母合成smallcaps，但目前尚未实现。

#Func

```
smallcaps(content) -> content
```

#Code

```
#set par(justify: true)
#set heading(numbering: "I.")

#show heading: it => {
  set block(below: 10pt)
  set text(weight: "regular")
  align(center, smallcaps(it))
}

= Introduction
#lorem(40)
```

11.11 引用smart quote

根据活动文本语言使用适当的引用符号。语法糖: '和"

#Func

```
smartquote(
  // 双引号
  set double: boolean,
  // 是否使用智能引号
  set enabled: boolean,
) -> content
```

#Code

```
"This is in quotes."

#set text(lang: "de")
"Das ist in Anführungszeichen."

#set text(lang: "fr")
"C'est entre guillemets."
```


#Demo

“This is in quotes.”

„Das ist in Anführungszeichen.“

« C’est entre guillemets. »

#Code

```
#set smartquote(enabled: false)
```

These are "dumb" quotes.

#Demo

These are "dumb" quotes.

11.12 删除线~~strikethrough~~

删除线

#Func

```
strike(
  // 删除线样式
  set stroke: auto length color stroke,
  // 删除线基于基线位置
  set offset: auto length,
  // 删除线是否比文本更长或者更短
  set extent: length,
  content,
) -> content
```

#Code

This is ~~#strike~~[not] relevant.

#Demo

This is ~~not~~ relevant.

#Code

```

This is #strike(stroke: 1.5pt + red)[very stricken through]. \
This is #strike(stroke: 10pt)[redacted].\
#set text(font: "Inria Serif")\
This is #strike(offset: auto)[low-ish]. \
This is #strike(offset: -3.5pt)[on-top].\
This #strike(extent: -2pt)[skips] parts of the word.\
This #strike(extent: 2pt)[extends] beyond the word.

```

#Demo

This is ~~very stricken through~~.

This is REDACTED.

This is ~~low-ish~~.

This is ~~on-top~~.

This ~~skips~~ parts of the word.

This ~~extends~~ beyond the word.

11.13 下标subscript

设置下标

#Func

```

sub(
  // 是否使用偏好字体的专用下标字符。
  // 如果启用，Typst 首先尝试将文本转换为下标代码点。如果失败，它会退回到渲染降
  // 低和缩小的正常字母。
  set typographic: boolean,
  set baseline: length,
  // 下标字体大小
  set size: length,
  content,
) -> content

```

#Code

```
Revenue#sub[yearly]
N#sub(typographic: true)[1]
N#sub(typographic: false)[1]
```

#Demo

Revenue_{yearly}

N₁

N₁

11.14 上标superscript

设置上标，与下标相同

#Func

```
super(
  // 是否使用偏好字体的专用上标字符。
  // 如果启用，Typst 首先尝试将文本转换为上标代码点。如果失败，它会退回到渲染降低和缩小的正常字母。
  set typographic: boolean,
  set baseline: length,
  // 上标字体大小
  set size: length,
  content,
) -> content
```

#Code

```
1#super[st] try! \
N#super(typographic: true)[1] \
N#super(typographic: false)[1] \
```

#Demo

1st try!

N¹

N¹

11.15 文本格式text

以多种方式自定义文本的外观和布局。

#Func

```
text(
  // 字体系列的优先顺序。
  // 处理文本时，Typst 按顺序尝试所有指定的字体系列，直到找到具有必要字形的字体。
  set font: string array,
  // 当主要字体列表不包含匹配项时是否允许其他字体。这使 Typst 可以在所有可用字体中搜索具有必要字形的最相似字体。
  // 注意：当回退被禁用并且没有找到字形时，没有警告。相反，您的文本以“tofus”的形式显示：表示缺少适当字形的小方框。
  set fallback: boolean,
  // 字型 normal italic oblique
  set style: string,
  // 字体粗细。接受 100 到 900 之间的整数或预定义的权重名称之一。
  // thin(100) extralight(200) light(300) regular(400) medium(500)
  // semibold(600) bold(700) extrabold(800) black(900)
  set weight: integer string,
  // 字型宽度
  set stretch: ratio,
  // 字号
  set size: length,
  // 字体颜色
  set fill: color,
  // 字符间距
  set tracking: length,
  // 单词空格
  set spacing: relative length,
  // 文本基线
  set baseline: length,
  // 对齐
  set overhang: boolean,
  // 文本框上边距
  // ascender cap-height x-height baseline descender
  set top-edge: length string,
  // 文本框下边距
  // ascender cap-height x-height baseline descender
  set bottom-edge: length string,
  // 语言
```

```
set lang: string,  
set region: none string,  
// 方向  
// auto ltr(左到右) rtl(右到左)  
set dir: auto direction,  
// 是否对文本断字以换行  
set hyphenate: auto boolean,  
// 自动字距调整  
set kerning: boolean,  
// 文本替代  
set alternates: boolean,  
set stylistic-set: none integer,  
// 是否启用连字显示  
set ligatures: boolean,  
set discretionary-ligatures: boolean,  
set historical-ligatures: boolean,  
set number-type: auto string,  
set number-width: auto string,  
// 是否有一条斜线穿过0  
set slashed-zero: boolean,  
// 是否将数字转换为分数。将此设置为 true 可启用 OpenType frac 字体功能。  
set fractions: boolean,  
set features: array dictionary,  
content,  
) -> content
```

#Code

```
#set text(18pt)  
With a set rule.  
  
#emph(text(blue)[  
  With a function call.  
])
```

#Demo

With a set rule.

With a function call.

#Code

```
#set text(font: (
  "Inria Serif",
  "Noto Sans Arabic",
))
```

This is Latin. \

هذا عربي.

#Demo

This is Latin.

هذا عربي.

#Code

```
#set text(font: "Inria Serif")
```

هذا عربي

```
#set text(fallback: false)
```

هذا عربي

#Demo

هذا عربي

#Code

```
#text(font: "Linux Libertine", style: "italic")[Italic]
```

```
#text(font: "DejaVu Sans", style: "oblique")[Oblique]
```

#Demo

Italic Oblique

#Code

```
#text(weight: "light")[Light] \
```

```
#text(weight: "regular")[Regular] \
```

```
#text(weight: "medium")[Medium] \
```

```
#text(weight: 500)[Medium] \  
#text(weight: "bold")[Bold]
```

#Demo

Light

Regular

Medium

Medium

Bold

#Code

```
#text(stretch: 75%)[Condensed] \  
#text(stretch: 100%)[Normal]
```

#Demo

Condensed

Normal

#Code

```
#set text(size: 20pt)  
very #text(1.5em)[big] text
```

#Demo

very **big** text

#Code

```
#set text(tracking: 1.5pt)  
Distant text.
```

#Demo

Distant text.

#Code

```
#set text(spacing: 200%)
```

Text with distant words.

```
A #text(baseline: 3pt)[lowered]  
word.
```

```
#set par(justify: true)
```

In this particular text, the justification produces a hyphen in the first line. Letting this hyphen hang slightly into the margin makes for a clear paragraph edge.

```
#set text(overhang: false)
```

In this particular text, the justification produces a hyphen in the first line. This time the hyphen does not hang into the margin, making the paragraph's edge less clear.

#Demo

Text with distant words.

A lowered word.

In this particular text, the justification produces a hyphen in the first line. Letting this hyphen hang slightly into the margin makes for a clear paragraph edge.

In this particular text, the justification produces a hyphen in the first line. This time the hyphen does not hang into the margin, making the paragraph's edge less clear.

#Code


```
#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]

#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]
```

#Demo

Typst

Typst

Typst

Typst

#Code

```
#set text(dir: rtl)
هذا عربي.

#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.

```
#set text(size: 25pt)
```

Totally

```
#set text(kerning: false)
```

Totally

```
#set text(size: 20pt)
```

0, a, g, ß

```
#set text(alternates: true)
```

0, a, g, ß

```
#set text(size: 20pt)
```

A fine ligature.

```
#set text(ligatures: false)
```

A fine ligature.

#Demo

هذا عربي.

.This text illustrates how enabling hyphenation can improve justification

.This text illustrates how enabling hyphenation can improve justification

Totally

Totally

a, g, ß ,0

a, g, ß ,0

.A fine ligature

.A fine ligature

#Code

```

#set text(font: "Noto Sans", 20pt)
#set text(number-type: "lining")
Number 9.

#set text(number-type: "old-style")
Number 9.

#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.

0, #text(slashed-zero: true)[0]

1/2 \
#text(fractions: true)[1/2]

// Enable the `frac` feature manually.
#set text(features: ("frac",))
1/2

```

#Demo

Number 9.

Number 9.

A 12 B 34.

A 56 B 78.

$A_{12} B_{34}.$
 $A_{56} B_{78}.$
 $0, 0$
 $\frac{1}{2}$
 $\frac{1}{2} \frac{1}{2}$

第 12 章 数学公式

Typst 有特殊的语法和库函数来排版数学公式。数学公式可以与文本一起显示，也可以作为单独的块显示。如果它们以至少一个空格开始和结束（例如 `$ x^2 $`），它们将被排版到自己的块中。在数学中，单个字母总是按原样显示。然而，多个字母被解释为变量和函数。要逐字显示多个字母，可以将它们放在引号中，要访问单个字母变量，您可以使用主题标签语法。

#Code

```
$ A = pi r^2 $
$ "area" = pi dot.op "radius"^2 $
$ cal(A) :=
  { x in RR | x "is natural" } $
#let x = 5
$ #x < 17 $
```

#Demo

$$A = \pi r^2$$

$$\text{area} = \pi \cdot \text{radius}^2$$

$$\mathcal{A} := \{x \in \mathbb{R} \mid x \text{ is natural}\}$$

$$5 < 17$$

数学模式提供多种符号选择，例如 `pi`、`dot.op` 或 `RR`。许多数学符号有不同的变体。您可以通过对符号应用修饰符来在不同的变体之间进行选择。Typst 进一步识别许多速记序列，如 `=>` 近似于一个符号。当存在这样的速记时，符号的文档会列出它。

#Code

```
$ x < y => x gt.eq.not y $
```

#Demo

$$x < y \Rightarrow x \not\leq y$$

公式也可以包含换行符。每行可以包含一个或多个对齐点 (`&`)，然后对齐。

#Code

```
$ sum_(k=0)^n k
  &= 1 + ... + n \
  &= (n(n+1)) / 2 $
```

#Demo

$$\sum_{k=0}^n k = 1 + \dots + n$$

$$= \frac{n(n+1)}{2}$$

数学模式支持不带主题标签前缀的特殊函数调用。在这些“数学调用”中，参数列表的工作方式与代码中的略有不同：

- 在它们内部，Typst 仍处于“数学模式”。因此，您可以将数学直接写入其中，但需要使用 `#` 来传递代码表达式（字符串除外，数学语法中提供）。
- 它们支持位置和命名参数，但不支持尾随内容块和参数传播。

- 它们为二维参数列表提供额外的语法。分号 (;) 将前面用逗号分隔的参数合并到一个数组参数中。

#Code

```
$ frac(a^2, 2) $
$ vec(1, 2, delim: "[") $
$ mat(1, 2; 3, 4) $
$ lim_x =
  op("lim", limits: #true)_x $
```

#Demo

$$\frac{a^2}{2}$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\lim_x = \lim_x$$

要在数学公式中使用逗号或分号，请使用反斜杠将其转义。另一方面，如果冒号前面直接有标识符，则只能以特殊方式识别冒号，因此在这些情况下要逐字显示它，您可以在它之前插入一个空格。以#开头的函数调用是正常的代码函数调用，不受这些规则的影响。所有数学函数都是数学模块的一部分，默认情况下在方程式中可用。在方程式之外，可以通过数学来访问它们。字首。

#Code

```
#show math.equation: set text(font: "Fira Math")
$ sum_(i in NN) 1 + i $
```

#Demo

$$\sum_{i \in \mathbb{N}} 1 + i$$

12.1 上下标

上标, 下标和极限, 下标、上标和极限。`attach` 函数支持 `a_b^c` 语法, 该语法将顶部和底部附件添加到方程的一部分。`attach` 可以显示为下标/上标或极限。Typst 会根据基础自动决定哪个更合适, 但您也可以使用脚本和限制功能手动控制它。

#Func

// 语法糖: 使用下划线 (_) 表示底部附件, 使用帽子 (^) 表示顶部附件。

```
attach(
  // 文本
  content,
  // 上标
  t: none content,
  // 下标
  b: none content,
  // 左上标
  tl: none content,
  // 左下标
  bl: none content,
  // 右上标
  tr: none content,
  // 右下标
  br: none content,
) -> content
```

#Code

```
// With syntax.
$ sum_(i=0)^n a_i = 2^(1+i) $

// With function call.
$ attach(
  Pi, t: alpha, b: beta,
  tl: 1, tr: 2, bl: 3, br: 4,
) $
```

#Demo

$$\sum_{i=0}^n a_i = 2^{1+i}$$

$$\frac{1}{3} \prod_{\beta}^{\alpha} 2$$

#Code

```
$ sum_(i=0)^n a_i = 2^(1+i) $
```

#Demo

$$\sum_{i=0}^n a_i = 2^{1+i}$$

#Func

```
scripts(
  // Force a base to display attachments as scripts.
  content
) -> content
```

#Code

```
$ scripts(sum)_1^2 != sum_1^2 $
```

#Demo

$$\sum_1^2 \neq \sum_1^2$$

#Func

```
// Force a base to display attachments as limits.
limits(
  content
) -> content
```

#Code

```
$ limits(A)_1^2 != A_1^2 $
```

#Demo

$$A_1^2 \neq A_1^2$$

12.2 分数

分数。语法糖：使用/将相邻表达式转换为分数。可以使用圆括号将多个原子分组到一个表达式中。括号会在输出中删除，但您可以嵌套多个以强制使用它们，或者使用转义符号。

#Func

```
frac(
  // 分子
  num: content,
  // 分母
  denom: content,
) -> content
```

#Code

```
$ 1/2 < (x+1)/2 $
$ ((x+1)) / 2 = frac(a, b) $
```

#Demo

$$\frac{1}{2} < \frac{x+1}{2}$$

$$\frac{(x+1)}{2} = \frac{a}{b}$$

12.3 方程

一个数学方程式。可以与文本内联显示或作为单独的块显示。语法糖：在\$内写入数学标记以创建方程式。以至少一个空格开始和结束等式将其提升到水平居中的单独块中。

#Func

```
equation(
  // 方程式是否显示为单独的块。
  set block: boolean,
  // 如何对块级方程进行编号。
```

```

set numbering: nonestringfunction,
// 对于方程的引用
set supplement: none auto content function
content,
) -> content

```

#Code

```
#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:

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

#Demo

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:

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

#Code

```
#set math.equation(numbering: "(1)")
```

We define:

$$\phi := \frac{1 + \sqrt{5}}{2} \quad \text{<ratio>}$$

With `@ratio`, we get:

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor$$

#Demo

We define:

$$\phi := \frac{1 + \sqrt{5}}{2} \quad (1)$$

With Equation 1, we get:

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor \quad (2)$$

#Code

```
#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) $
```

#Demo

We define:

$$\phi := \frac{1 + \sqrt{5}}{2} \quad (3)$$

With Eq. 3, we get:

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor \quad (4)$$

12.4 向量

列向量。矢量元素中的内容可以与 & 符号对齐。

#Func

```
vec(
  // 向量括号形式: "(" "[" "{" "|" "||"
  set delim: nonestring,
  ..content,
) -> content
```

#Code

```
$ vec(a, b, c) dot.op vec(1, 2, 3)
    = a + 2b + 3c $

#set math.vec(delim: "[")
$ vec(1, 2) $
```

#Demo

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = a + 2b + 3c$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

12.5 矩阵

创建矩阵。一行的元素应该用逗号分隔，而行本身应该用分号分隔。分号语法将前面用逗号分隔的参数合并到一个数组中。您还可以使用这种特殊的数学函数调用语法来定义采用二维数据的自定义函数。同一行单元格中的内容可以用 & 符号对齐。

#Func

```
mat(
  // 矩阵括号形式: "(" "[" "{" "|" "||"
  set delim: nonestring,
  // 具有矩阵行数组的数组。
  ..array,
) -> content
```

#Code

```
$ mat(
  1, 2, ..., 10;
  2, 2, ..., 10;
  dots.v, dots.v, dots.down, dots.v;
  10, 10, ..., 10;
) $
```

#Demo

$$\begin{pmatrix} 1 & 2 & \dots & 10 \\ 2 & 2 & \dots & 10 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix}$$

#Code

```
#set math.mat(delim: "[ ")
$ mat(1, 2; 3, 4) $
```

#Demo

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

#Code

```
#let data = ((1, 2, 3), (4, 5, 6))
#let matrix = math.mat(..data)
$ v := matrix $
```

#Demo

$$v := \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

12.6 选择语句

选择语句，区分大小写，对于跨越不同分支的内容可以使用&对齐。

#Func

```
cases(
  // 选择括号形式: "(" "[" "{" "|" "||"
  set delim: string,
  ..content,
) -> content
```

#Code

```
$ f(x, y) := cases(
  1 "if" (x dot.op y)/2 <= 0,
```

```

2 "if" x "is even",
3 "if" x in NN,
4 "else",
) $

```

#Demo

$$f(x, y) := \begin{cases} 1 & \text{if } \frac{x \cdot y}{2} \leq 0 \\ 2 & \text{if } x \text{ is even} \\ 3 & \text{if } x \in \mathbb{N} \\ 4 & \text{else} \end{cases}$$

#Code

```

#set math.cases(delim: "[")
$ x = cases(1, 2) $

```

#Demo

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

12.7 定界符

分隔符匹配。lr 函数允许匹配两个定界符并根据它们包含的内容缩放它们。虽然这对于语法匹配的定界符也会自动发生，但 lr 允许您匹配两个任意定界符并精确控制它们的大小。除了 lr 函数之外，Typst 还提供了一些函数来为绝对值、上限值和下限值以及范数创建定界符对。

#Func

// 刻度定界符。虽然匹配的定界符默认缩放，但这可用于缩放不匹配的定界符并更精确地控制定界符缩放。

```

lr(
  // 括号的大小，相对于包裹内容的高度。默认为 100%。
  set size: auto relative length,
  content,
) -> content

```

#Code

```
$ \r([a, b/2]) $
$ \r(\sum_{x=1}^n x, size: #50%) $
```

#Demo

$$\left[a, \frac{b}{2} \right]$$

$$\left[\sum_{x=1}^n x \right]$$

#Func

```
// 绝对值符号
abs(
  content
) -> content
```

#Code

```
$ abs(x/2) $
```

#Demo

$$\left| \frac{x}{2} \right|$$

#Func

```
// 范数符号
norm(
  content
) -> content
```

#Code

```
$ norm(x/2) $
```

#Demo

$$\left\| \frac{x}{2} \right\|$$

#Func

```
// 上半封顶括号
floor(
  content
) -> content
```

#Code

```
$ floor(x/2) $
```

#Demo

$$\left\lceil \frac{x}{2} \right\rceil$$

#Func

```
// 下半封顶括号
ceil(
  content
) -> content
```

#Code

```
$ ceil(x/2) $
```

#Demo

$$\left\lfloor \frac{x}{2} \right\rfloor$$

12.8 上下分隔符

等式部分上方或下方的分隔符。大括号和方括号进一步允许您在其下方或上方添加可选注释。

#Func

```
// 公式下划线
underline(content) -> content
// 公式上划线
overline(content) -> content
```



```

// 可加注释的下水平括号
underbrace(
  content,
  // 括号下方注释
  set none content,
) -> content
// 可加注释的上水平括号
overbrace(
  content,
  // 括号上方注释
  set none content,
) -> content
// 可加注释的下水平方括号
underbracket(
  content,
  // 注释
  set none content,
) -> content
// 可加注释的上水平方括号
overbracket(
  content,
  // 注释
  set nonecontent,
) -> content

```

#Code

```

$ underline(1 + 2 + ... + 5) $
$ overline(1 + 2 + ... + 5) $
$ underbrace(1 + 2 + ... + 5, "numbers") $
$ overbrace(1 + 2 + ... + 5, "numbers") $
$ underbracket(1 + 2 + ... + 5, "numbers") $
$ overbracket(1 + 2 + ... + 5, "numbers") $

```

#Demo

$$\begin{array}{c}
 \underline{1 + 2 + \dots + 5} \\
 \overline{1 + 2 + \dots + 5} \\
 \underbrace{1 + 2 + \dots + 5}_{\text{numbers}}
 \end{array}$$

$$\overbrace{1 + 2 + \dots + 5}^{\text{numbers}}$$

$$\underbrace{1 + 2 + \dots + 5}_{\text{numbers}}$$

$$\overline{\overbrace{1 + 2 + \dots + 5}^{\text{numbers}}}$$

12.9 字母accent

给字母添加"帽子"。

#Func

```
accent(
  // 加帽子的字母或者字符串
  base: content,
  // 帽子类型，见下表
  accent: string content,
) -> content
```

#Code

```
$grave(a) = accent(a, `)`$ \
$arrow(a) = accent(a, arrow)$ \
$tilde(a) = accent(a, \u{0303})$
```

#Demo

$\grave{a} = \grave{a}$

$\vec{a} = \vec{a}$

$\tilde{a} = \tilde{a}$

#Code

```
$arrow(A B C)$
```

#Demo

\overrightarrow{ABC}

Accent	Name	Codepoint
Grave	grave	`
Acute	acute	'
Circumflex	hat	^
Tilde	tilde	~
Macron	macron	-
Breve	breve	˘
Dot	dot	.
Diaeresis	diaer	¨
Circle	circle	◦
Double acute	acute.double	¨
Caron	grave	ˇ
Right arrow	arrow, ->	→
Left arrow	arrow.l, <-	←

12.10 公式字体替换

替换公式中的字体。这些函数与文本函数不同，因为数学字体包含每个字母的多种变体。

#Func

```
// 数学中的衬线（罗马）字体样式。默认。
serif(content) -> content
```

#Func

```
// Sans-serif字体
sans(content) -> content
```

#Code

```
$ sans(A B C) $
```

#Demo

ABC

#Func

```
// Fraktur 字体样式。
frak(content) -> content
```

#Code

```
$ frak(P) $
```

#Demo

℘

#Func

```
// 数学中的等宽字体样式。
mono(content) -> content
```

#Code

```
$ mono(x + y = z) $
```

#Demo

$x + y = z$

#Func

```
// 数学中的黑板粗体（双打）字体样式。对于大写拉丁字母，还可以通过 NN 和 RR 形式
的符号使用黑板粗体。
bb(content) -> content
```

#Code

```
$ bb(b) $
$ bb(N) = NN $
$ f: NN -> RR $
```

#Demo

$$\mathbb{b}$$

$$\mathbb{N} = \mathbb{N}$$

$$f : \mathbb{N} \rightarrow \mathbb{R}$$

#Func

// 数学中的书法字体样式。

```
cal(content) -> content
```

#Code

Let $\text{\textcolor{blue}{cal}}(P)$ be the set of ...

#Demo

Let \mathcal{P} be the set of ...

12.11 字体样式设置

设置公式中的字母形式。这些函数与文本函数不同，因为数学字体包含每个字母的多种变体。

#Func

// 数学中的直立（非斜体）字体样式。

```
upright(content) -> content
```

// 数学中的斜体字体样式。对于罗马字母和希腊小写字母，这已经是默认值。

```
italic(content) -> content
```

// 数学中的粗体字体样式。

```
bold(content) -> content
```

#Code

```
$ upright(A) != A $
$ bold(A) := B^+ $
```

#Demo

$$A \neq A$$

$$A := B^+$$

12.12 round

使用半括号包围表达式

#Func

round(content) -> content

#Code

```
$ round(x/2) $
```

#Demo

$$\left\lfloor \frac{x}{2} \right\rfloor$$

12.13 运算符

文本运算符。Typst 预定义了运算符 arccos, arcsin, arctan, arg, cos, cosh, cot, ctg, coth, csc, deg, det, dim, exp, gcd, hom, mod, inf, ker, lg, lim, ln, log, max, min, Pr, sec, sin, sinc, sinh, sup, tan, tg, tanh, liminf 和 limsup.

#Func

```
op(
  // 文本
  text: string,
  // Whether the operator should force attachments to display as limits. Defaults to
  false.
  set limits: boolean,
) -> content
```

#Code

```
$ tan x = (sin x) / (cos x) $
$ op("custom",
  limits: #true)_(n->oo) n $
```

#Demo

$$\tan x = \frac{\sin x}{\cos x}$$

custom n
 $n \rightarrow \infty$

12.14 根号运算

平方根和多次方根

现在 `$root(x+y)$` 等价于 `$\sqrt{x+y}$` .

#Func

```
// 多次方根
root(
  // 几次根
  index: none content,
  // 开根表达式
  radicand: content,
) -> content
// 平方根
sqrt(
  // 对其求平方根的表达式。
  radicand: content
) -> content
```

#Code

```
$ root(3, x) $
$ sqrt(x^2) = x = sqrt(x)^2 $
```

#Demo

$$\sqrt[3]{x}$$

$$\sqrt{x^2} = x = \sqrt{x}^2$$

12.15 cancel

在方程的一部分上显示对角线。这通常用于表示一个术语的消除。

#Func

```
cancel(
  content,
  length: relative length,
```

```

inverted: boolean,
cross: boolean,
rotation: angle,
stroke: length color dictionary stroke,
) -> content

```

#Code

Here, we can simplify:

```

$ (a dot b dot cancel(x)) /
  cancel(x) $

```

#Demo

Here, we can simplify:

$$\frac{a \cdot b \cdot x}{x}$$

#Code

```

$ a + cancel(x, length: #200%)
  - cancel(x, length: #200%) $

```

#Demo

$$a + x - x$$

#Code

```

$ (a cancel((b + c), inverted: #true)) /
  cancel(b + c, inverted: #true) $

```

#Demo

$$\frac{a(\cancel{b+c})}{\cancel{b+c}}$$

#Code

```

$ cancel(Pi, cross: #true) $

```

#Demo

\mathbb{H}

#Code

```
$ cancel(Pi, rotation: #30deg) $
```

#Demo

 \mathbb{H}

#Code

```
$ cancel(
  sum x,
  stroke: #(
    paint: red,
    thickness: 1.5pt,
    dash: "dashed",
  ),
) $
```

#Demo

 $\sum x$

12.16 binom

二项式表达式?

#Func

```
binom(
  // 二项式上索引
  upper: content,
  // 二项式下索引
  lower: content,
) -> content
```

#Code

```
$ binom(n, k) $
```

#Demo

$$\binom{n}{k}$$

第 13 章 页面布局

以不同方式排列页面上的元素。通过组合布局功能，您可以创建复杂的自动布局。

13.1 对齐align

水平或者垂直对齐内容

#Func

```
align(
  // 水平排列 start end left center right
  // 竖直排列 top horizon bottom
  // 同时沿两个轴对齐，使用 + 运算符添加两个对齐以获得 2d 对齐。例如，top + right
  // 将内容对齐到右上角。
  set alignment2d alignment,
  content,
) -> content
```

#Code

```
#set align(center)

Centered text, a sight to see \
In perfect balance, visually \
Not left nor right, it stands alone \
A work of art, a visual throne
```

#Demo

Centered text, a sight to see

In perfect balance, visually

Not left nor right, it stands alone

A work of art, a visual throne

13.2 块block

使用block可以分割内容，给内容添加背景，也可以跨页展示。Block可以强制本来内联的元素变为block层级，特别是编写show规则时

#Func

```
block(
  set width: auto relative length, // 设置block宽度
  set height: auto relative length,
  // 设置block的高度。当设置高度大于该页剩余空间,
  // breakable为true时, block将在下一个页面上继续。
  set breakable: boolean, // block是否可打断
  set fill: nonecolor, // 背景颜色
  set stroke: none length color dictionary stroke, // 边界颜色
  set radius: relative length dictionary, // block圆角半径
  set inset: relative length dictionary, // 内容与block边界距离
  set outset: relative length dictionary, // block外扩多少距离
  set spacing: relative length fraction,
  // block距离上下文的间距 可以使用above, below代替
  set above: content,
  // block与上一个block的间距。优先于spacing。
  // 可以与show结合使用, 以调整任意块级元素周围的间距。
  set below: content,
  // block与下一个block的间距。优先于spacing。
  // 可以与show结合使用, 以调整任意块级元素周围的间距。
  set clip: boolean,
  // 是否裁剪内容 (v0.1新增)
  set `none` content, // block内容
) -> content
```

#Code

```
#align(center)[
  #block(
    width:50%,
    height: 15em,
    breakable: true,
    fill: gray,
    stroke: black + 2pt,
    radius:5pt,
```

```

    inset: 15pt,
    outset: 1pt,
    spacing: 50%,
    lorem(20),
  )
]

```

#Demo

Lorem ipsum dolor sit amet, con-
 sectetur adipiscing elit, sed do
 eiusmod tempor incididunt ut labore
 et dolore magna aliqua quaerat.

13.3 盒子box

#Func

内联级的container，除了公式、文字、box之外所有的元素都是block级的，不能出现在一个段落中

box可以用来将元素整合到一个段落中

```

box(
  set width: `auto` `relative length` `fraction`, // 盒子宽度
  set height: `auto` `relative length`,           // 盒子高度
  set baseline: `relative length`,                 // 盒子基线
  set fill: `none` `color`,                         // 背景颜色
  set stroke: `none` `length` `color` `dictionary` `stroke`, // 盒子边界
  set radius: `relative length` `dictionary`,      // 盒子圆角半径
  set inset: `relative length` `dictionary`,        // 内容距离盒子边界距离
  set outset: `relative length` `dictionary`,       // 盒子外扩值
  set clip: boolean,                                // 是否裁剪内容 (v0.1新增)
  set `none` `content`,
) -> content

```

#Code

Line in `#box(width: 1fr, line(length: 100%))` between.

An inline

```
#box(
  fill: luma(235),
  inset: (x: 3pt, y: 0pt),
  outset: (y: 3pt),
  radius: 2pt,
)[rectangle].
```

#Demo

Line in _____ between.

An inline `rectangle`.

13.4 表格table

表格用于排列单元格中的内容。单元格可以包含任意内容，包括多个段落，并以行优先顺序指定。

#Func



```
table(
  // 列
  set columns: auto integer relative length fraction array,
  // 行
  set rows: auto integer relative length fraction array,
  // 行列间距
  set gutter: auto integer relative length fraction array,
  // 列间距
  set column-gutter: auto integer relative length fraction array,
  // 行间距
  set row-gutter: auto integer relative length fraction array,
  // 填充样式
  set fill: none color function,
  // 对齐规则
  set align: auto function alignment2d alignment,
  // 盒子格式
  set stroke: none length color stroke,
```

```
// 内容与盒子边距
set inset: relative length,
  ..content,
) -> content
```

#Code

```
#table(
  columns: (auto, auto, auto),
  inset: 10pt,
  align: horizon,
  [], [*Area*], [*Parameters*],
  image("cylinder.svg",height:5%),
  $ pi h (D^2 - d^2) / 4 $,
  [
    $h$: height \
    $D$: outer radius \
    $d$: inner radius
  ],
  image("tetrahedron.svg",height:5%),
  $ sqrt(2) / 12 a^3 $,
  [$a$: edge length]
)
```

#Demo

	Area	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

#Code

```
#table(
  fill: (col, _) => if calc.odd(col) { luma(240) } else { white },
  align: (col, row) =>
    if row == 0 { center }
    else if col == 0 { left }
    else { right },
  columns: 4,
  [], [*Q1*], [*Q2*], [*Q3*],
  [Revenue:], [1000 €], [2000 €], [3000 €],
  [Expenses:], [500 €], [1000 €], [1500 €],
  [Profit:], [500 €], [1000 €], [1500 €],
)
```

#Demo

	Q1	Q2	Q3
Revenue:	1000 €	2000 €	3000 €
Expenses:	500 €	1000 €	1500 €
Profit:	500 €	1000 €	1500 €

#Code

```
#table(
  columns: 3,
  align: (x, y) => (left, center, right).at(x),
  [Hello], [Hello], [Hello],
  [A], [B], [C],
)
```

#Demo

Hello	Hello	Hello
A	B	C

13.5 列表list

项目符号列表。垂直显示一系列项目，每个项目由一个标记引入。

#Func

```
list(  
  // 是否紧凑布局  
  set tight: boolean,  
  // 列表标记  
  set marker: contentarrayfunction,  
  // 缩进  
  set indent: length,  
  // 标记与主体间的距离  
  set body-indent: length,  
  // 列表间间距  
  set spacing: autorelative lengthfraction,  
  ..content,  
) -> content
```

#Code

```
- *Content*  
  - Text  
  - Math  
  - Layout  
  - Visualize  
  - Meta  
  - Symbols  
  
- *Compute*  
  #list(  
    [Foundations],  
    [Calculate],  
    [Construct],  
    [Data Loading],  
  )
```

#Demo

- **Content**
 - Text
 - Math
 - Layout

- Visualize
- Meta
- Symbols
- **Compute**
- Foundations
- Calculate
- Construct
- Data Loading

#Code

```
#set list(marker: [--])  
- A more classic list  
- With en-dashes  
  
#set list(marker: ([·], [--]))  
- Top-level  
  - Nested  
  - Items  
- Items  
  
#for letter in "ABC" [  
  - Letter #letter  
]
```

#Demo

- A more classic list
- With en-dashes
- Top-level
- Nested
- Items
- Items
- Letter A

- Letter B
- Letter C

13.6 强制分栏 `colbreak`

强制分栏。在单列布局或页面的最后一列中使用时，该函数的行为类似于分页符。否则，分栏后的内容将放在下一栏中。

#Func

```
// 如果为 true，则在当前列已经为空时跳过分栏符。
colbreak(set weak:boolean) -> content
```

#Code

```
#set page(columns: 2)
Preliminary findings from our
ongoing research project have
revealed a hitherto unknown
phenomenon of extraordinary
significance.

#colbreak()
Through rigorous experimentation
and analysis, we have discovered
a hitherto uncharacterized process
that defies our current
understanding of the fundamental
laws of nature.
```

13.7 分列 `clolumns`

将一个区域分成多个大小相同的列。列功能允许将任何容器的内部分成多个列。它不会使列的高度相等，相反，列将占用其容器的高度或页面上的剩余高度。如有必要，列功能可以跨页。

#Func

```
columns(
  // 列数
  set integer,
  // 列间距
  set gutter: relative length,
  content,
) -> content
```

#Code

```
#box(height: 68pt,
  columns(2, gutter: 11pt)[
    #set par(justify: true)
    This research was funded by the
    National Academy of Sciences.
    NAoS provided support for field
    tests and interviews with a
    grant of up to USD 40.000 for a
    period of 6 months.
  ]
)
```

In recent years, deep learning has increasingly been used to solve a variety of problems.

#Demo

This research was funded by the National Academy of Sciences. NAoS provided support for field tests and interviews with a grant of up to USD 40.000 for a period of 6 months.

support for field tests and interviews with

In recent years, deep learning has increasingly been used to solve a variety of problems.

13.8 enum

见之前章节 Section 6 描述

13.9 grid

见之间章节 Section 7 描述

13.10 水平间距h

在段落中插入水平间距。间距可以是绝对的、相对的或分数。在最后一种情况下，线上的剩余空间根据它们的相对分数分布在所有分数间距中。

#Func

```
h(  
  // 插入多少间距  
  amount: relative length fraction,  
  set weak: boolean,  
) -> content
```

#Code

```
First #h(1cm) Second \  
First #h(30%) Second \  
First #h(2fr) Second #h(1fr) Third
```

#Demo

First	Second	
First	Second	
First	Second	Third

13.11 垂直间距v

将垂直间距插入块流中。间距可以是绝对的、相对的或分数。在最后一种情况下，页面上的剩余空间根据它们的相对分数分布在所有分数间距中。

#Func

```
v(  
  // 插入多少间距  
  amount: relative length fraction,
```

```
    set weak: boolean,
  ) -> content
```

#Code

```
#grid(
  rows: 3cm,
  columns: 6,
  gutter: 1fr,
  [A #parbreak() B],
  [A #v(0pt) B],
  [A #v(10pt) B],
  [A #v(0pt, weak: true) B],
  [A #v(40%, weak: true) B],
  [A #v(1fr) B],
)
```

#Demo

A	A	A	$\frac{A}{B}$	A	A
B	B	B		B	
					B

13.12 隐藏内容hide

隐藏内容而不影响布局。隐藏功能允许您在布局仍“看到”内容时隐藏内容。这对于创建与某些内容一样大的空白很有用。编辑内容也可能有用，因为它的参数不包含在输出中。

#Func

```
hide(content) -> content
```

#Code

```
Hello Jane \
#hide[Hello] Joe
```

#Demo

Hello Jane

Joe

13.13 测量measure

衡量内容的布局大小。`measure` 函数可让您确定内容的布局大小。相同的内容可以有不同的大小，具体取决于布局时处于活动状态的样式。

#Func

```
// measure 函数返回一个字典，其中包含条目 width 和 height，两者都是 length 类型。
measure(
  content,
  styles,
) -> dictionary
```

#Code

```
#let content = [Hello!]
#content
#set text(14pt)
#content
```

#Demo

Hello! Hello!

因此，要进行有意义的measure，您首先需要使用 `style` 函数检索活动样式。然后您可以将它们传递给测量函数。

#Code

```
#let thing(body) = style(styles => {
  let size = measure(body, styles)
  [Width of "#body" is #size.width]
})
```

```
#thing[Hey] \
#thing[Welcome]
```

#Demo

Width of “Hey” is 19.99pt

Width of “Welcome” is 45.01pt

13.14 移动move

移动内容而不影响布局。move允许您移动内容，而layout仍然在原始位置“看到”它。容器仍将调整大小，就好像内容没有被移动一样。

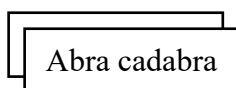
#Func

```
move(
  // 水平移动距离
  set dx: relative length,
  // 垂直移动距离
  set dy: relative length,
  content,
) -> content
```

#Code

```
#rect(inset: 0pt, move(
  dx: 6pt, dy: 6pt,
  rect(
    inset: 8pt,
    fill: white,
    stroke: black,
    [Abra cadabra]
  )
))
```

#Demo



13.15 间距padding

`pad` 函数在内容周围添加间距。可以为每一边单独指定间距，或者通过指定位置参数一次为所有边指定间距。

#Func

```
pad(
  // 左边距
  set left: relative length,
  // 上边距
  set top: relative length,
  // 右边距
  set right: relative length,
  // 下边距
  set bottom: relative length,
  // 水平间距，优先级低于左右边距设置
  set x: relative length,
  // 垂直间距，优先级低于上下边距
  set y: relative length,
  // 所有边的填充。所有其他参数优先于此。
  set rest: relative length,
  content,
) -> content
```

#Code

```
#set align(center)

#pad(x: 16pt, image("typing.jpg"))
_Typing speeds can be
measured in words per minute._
```

#Demo



Typing speeds can be measured in words per minute.

13.16 页面格式page

设置页面格式

#Func

```

page(
  // 设置页面大小，默认为a4
  set paper: string,
  // 页面宽度
  set width: auto length,
  // 页面高度
  set height: auto length,
  // 页面是否翻转为横向
  set flipped: boolean,
  // 页边距
  // 可以使用字典{ }单独设置边距,
  // (top: value,right: value,bottom: value,left: value,x: value,y: value,z: value,rest: value)
  set margin: auto relative length dictionary,
  // 列数
  set columns: integer,
  // 背景颜色
  set fill: none color,
  // 给页码编号。如果给出了明确的页脚，则编号将被忽略。
  set numbering: none string function,
  // 页码对齐方式
  set number-align: alignment2d alignment,
  // 页眉
  set header: none content,
  // 页眉距离上边距的量
  set header-ascent: relative length,
  // 页面的页脚。填充每页的底部边距。对于页码，编号属性通常就足够了。
  // 如果要创建自定义页脚，但仍显示页码，则可以直接访问页码计数器。
  set footer: none content,
  // 页脚距离下边距的距离
  set footer-descent: relative length,
  // 页面背景中的内容。此内容将放置在页面正文的后面。它可用于放置背景图像或水印。
  set background: none content,
  // 页面前景中的内容。此内容将覆盖页面的主体。
  set foreground: none content,
  content,
) -> content

```

#Code

```
#set page("us-letter")
```

There you go, US friends

#Code

```
#set page(  
  width: 3cm,  
  margin: (x: 0cm),  
)  
  
#for i in range(3) {  
  box(square(width: 1cm))  
}
```

#Code

```
#set page(  
  "us-business-card",  
  flipped: true,  
  fill: rgb("f2e5dd"),  
)  
  
#set align(bottom + end)  
#text(14pt)[*Sam H. Richards*] \  
_Procurement Manager_  
  
#set text(10pt)  
17 Main Street \  
New York, NY 10001 \  
+1 555 555 5555
```

#Code

```
#set page(  
  width: 3cm,  
  height: 4cm,  
  margin: (x: 8pt, y: 4pt),  
)
```

```
#rect(  
  width: 100%,  
  height: 100%,  
  fill: aqua,  
)
```

#Code

```
#set page(columns: 2, height: 4.8cm)  
Climate change is one of the most  
pressing issues of our time, with  
the potential to devastate  
communities, ecosystems, and  
economies around the world. It's  
clear that we need to take urgent  
action to reduce our carbon  
emissions and mitigate the impacts  
of a rapidly changing climate.
```

#Code

```
#set page(fill: rgb("444352"))  
#set text(fill: rgb("fdfdfd"))  
*Dark mode enabled.*
```

#Code

```
#set page(  
  height: 100pt,  
  margin: (top: 16pt, bottom: 24pt),  
  numbering: "1 / 1",  
)  
  
#lorem(48)
```

#Code

```
#set page(  
  margin: (top: 16pt, bottom: 24pt),  
  numbering: "1",
```

```
    number-align: right,  
  )  
  
#lorem(30)
```

#Code

```
#set par(justify: true)  
#set page(  
  margin: (top: 32pt, bottom: 20pt),  
  header: [  
    #set text(8pt)  
    #smallcaps[Typst Academy]  
    #h(1fr) _Exercise Sheet 3_  
  ],  
)  
  
#lorem(19)
```

#Code

```
#set par(justify: true)  
#set page(  
  height: 100pt,  
  margin: 20pt,  
  footer: [  
    #set align(right)  
    #set text(8pt)  
    #counter(page).display(  
      "1 of I",  
      both: true,  
    )  
  ]  
)  
  
#lorem(48)
```

#Code

```
#set page(background: rotate(24deg,
  text(18pt, fill: rgb("FCBC4"))[
    *CONFIDENTIAL*
  ]
))
```

= Typst's secret plans

In the year 2023, we plan to take
over the world (of typesetting).

#Code

```
#set page(foreground: text(24pt)[])
```

Reviewer 2 has marked our paper
"Weak Reject" because they did
not understand our approach...

13.17 换页符 `pagebreak`

手动分页符

#Func

```
// if true, 如果当前页为空，则不使用分页符
pagebreak(set weak:boolean) -> content
```

#Code

```
The next page contains
more details on compound theory.
#pagebreak()
```

== Compound Theory

In 1984, the first ...

13.18 段落 `par`

将文本、间距和行内级元素排列到一个段落中。尽管此函数主要用于设置规则以影响段落属性，但它也可用于将其参数显式呈现到其自己的段落中。

#Func

```

par(
  // 行间距 default 0.65em
  set leading: length,
  // 是否在行中对其文本
  set justify: boolean,
  // 如何确定换行符。auto simple optimized
  set linebreaks: auto string,
  // 连续段落的第一行应该有的缩进。页面上的第一段永远不会缩进。按照排版惯例，段落分隔符由段落之间的一些空格或首行缩进表示。使用此属性时请考虑关闭段落间距（例如使用#show par: set block(spacing: 0pt)）。
  set first-line-indent: length,
  set hanging-indent: length,
  set body: content,
) -> content

```

#Code

```

#set par(first-line-indent: 1em, justify: true)
#show par: set block(spacing: 0.65em)

```

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 ...

#Demo

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 ...

#Code

```
#set page(width: 190pt)
#set par(linebreaks: "simple")
```

Some texts are frustratingly
challenging to break in a
visually pleasing way. This
very aesthetic example is one
of them.

```
#set par(linebreaks: "optimized")
```

Some texts are frustratingly
challenging to break in a
visually pleasing way. This
very aesthetic example is one
of them.

#Demo

Some texts are frustratingly challenging to break in a visually pleasing way. This
very aesthetic example is one of them.

Some texts are frustratingly challenging to break in a visually pleasing way. This
very aesthetic example is one of them.

13.19 换段符`parbreak`

段落终断符

#Func

```
parbreak() -> content
```

#Code

```
#for i in range(3) {
  [Blind text #i: ]
  lorem(5)
  parbreak()
}
```

#Demo

Blind text 0: Lorem ipsum dolor sit amet.

Blind text 1: Lorem ipsum dolor sit amet.

Blind text 2: Lorem ipsum dolor sit amet.

13.20 内容放置 **place**

将内容放在绝对位置。放置的内容不会影响其他内容的位置。Place 始终相对于其父容器，并且将位于容器中所有其他内容的前台。

#Func

```
place(
  set alignment2d alignment,
  // 水平位移
  set dx: relative length,
  // 垂直位移
  set dy: relative length,
  content,
) -> content
```

#Code

```
Hello, world!

#place(
  top + right,
  square(
    width: 20pt,
    stroke: 2pt + blue
  ),
)
```

#Demo

Hello, world!



#Code

```
#for i in range(16) {
  let amount = i * 4pt
```



```
place(center, dx: amount - 32pt, dy: amount / 8)[A]
}
```

#Demo

AAAAAAAAAAAA

13.21 重复内容repeat

重复内容。这在实现自定义索引、引用或大纲时很有用。

#Func

```
repeat(content) -> content
```

#Code

```
Sign on the dotted line:
#box(width: 1fr, repeat[.])

#set text(10pt)
#v(8pt, weak: true)
#align(right)[
  Berlin, the 22nd of December, 2022
]
```

#Demo

Sign on the dotted line:
Berlin, the 22nd of December, 2022

13.22 旋转rotate

按给定角度旋转元素。

#Func

```
rotate(
  // 旋转角
  set angle,
  // 旋转原点，默认情况下，原点是旋转元素的中心。旋转元素的左下角与基线保持对
  齐，将原点设置为 bottom + left。
  set origin: alignment2d alignment,
```

```
content,
) -> content
```

```
#Code
#stack(
  dir: ltr,
  spacing: 1fr,
  ..range(16)
    .map(i => rotate(24deg * i)[X]),
)
```

```
#Demo
X  ↗  ↖  ↘  ↙  ↗  ↖  ↘  ↙  ↗  ↖  ↘  ↙  ↗  ↖  ↘  ↙
```

```
#Code
#rotate(-1.571rad)[Space!]
```

```
#Demo
Space!
```

```
#Code
#set text(spacing: 8pt)
#let square = square.with(width: 8pt)

#box(square())
#box(rotate(30deg, origin: center, square()))
#box(rotate(30deg, origin: top + left, square()))
#box(rotate(30deg, origin: bottom + right, square()))
```

```
#Demo
□ ◊ ◊ ◊
```

13.23 水平/垂直排列stack

水平或垂直排列内容和间距。stack沿轴放置项目列表，每个项目之间的间距可选。

#Func

```

stack(
  // 方向: ltr rtl ttb btt
  set dir: direction,
  // 间距
  set spacing: none relative length fraction,
  ..relative length fraction content,
) -> content

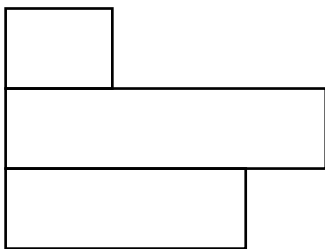
```

#Code

```

#stack(
  dir: ttb,
  rect(width: 40pt),
  rect(width: 120pt),
  rect(width: 90pt),
)

```

#Demo**13.24 内容放缩scale**

在不影响布局的情况下缩放内容。缩放功能允许在不影响布局的情况下缩放和镜像内容。

#Func

```

scale(
  // 水平放缩因子, 如果为负, 水平镜像
  set x: ratio,
  // 垂直放缩因子, 如果为负, 垂直镜像
  set y: ratio,
  // 放缩原点
  set origin: alignment2d alignment,

```

```
content,
) -> content
```

#Code

```
#set align(center)
#scale(x: -100%)[This is mirrored.]
```

#Demo

.beronim si sidT

#Code

```
A#box(scale(75%)[A])A \
B#box(scale(75%, origin: bottom + left)[B])B
```

#Demo

AA A

BB B

13.25 定理terms

术语的使用及其描述。垂直显示一系列术语及其描述。当描述跨越多行时，他们使用悬挂缩进来传达视觉层次结构。

语法糖：以/开始一行，后跟术语、:和描述，创建术语列表项。

#Func

```
terms(
  // false: 项目以术语列表间距隔开。true: 改用正常行距。这使得术语列表更紧凑
  set tight: boolean,
  // 分隔符
  set separator: content,
  // 缩进
  set indent: length,
  // 悬挂缩进
  set hanging-indent: length,
  // 间距
  set spacing: auto relative length fraction,
```

```
..content array,
) -> content
```

#Code

```
/ Ligature: A merged glyph.
/ Kerning: A spacing adjustment between two adjacent letters.
```

#Demo

Ligature A merged glyph.

Kerning A spacing adjustment between two adjacent letters.

#Code

```
#set terms(separator: [: ])

/ Colon: A nice separator symbol.
```

#Demo

Colon: A nice separator symbol.

#Code

```
#set terms(separator: h(2cm,weak:true))

/ Colon: A nice separator symbol.
```

#Demo

Colon A nice separator symbol.

#Code

```
#set terms(hanging-indent: Opt)

/ Term: This term list does not
  make use of hanging indents.
```

#Demo

Term This term list does not

make use of hanging indents.

#Code

```
#for year, product in (
  "1978": "TeX",
  "1984": "LaTeX",
  "2019": "Typst",
) [ / #product: Born in #year.]
```

#Demo

TeX Born in 1978.

LaTeX Born in 1984.

Typst Born in 2019.

第 14 章 可视化

目前Typst还不支持绘制图表!

14.1 直线line

绘制直线

#Func

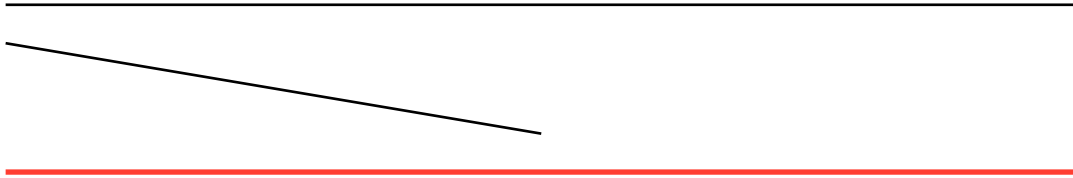
```
line(
  set start: array,
  set end: none array,
  set length: relative length,
  set angle: angle,
  set stroke: length color stroke,
) -> content
```

#Code

```
#set page(height: 100pt)
#line(length: 100%)
```

```
#line(end: (50%, 5%))
#line(length: 100%, stroke: 2pt + red)
```

#Demo



14.2 矩形rect

绘制矩形

#Func

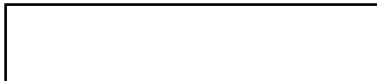
```
rect(
  // 宽
  set width: auto relative length,
  // 高
  set height: auto relative length,
  // 内部填充
  // 设置填充时，默认stroke失效。要创建同时具有填充和描边的矩形，您必须同时配置
  // 两者。
  set fill: none color,
  // 圆圈格式
  // none 不使用描边 auto 1pt黑色描边
  set stroke: none auto length color stroke,
  // 边框弧度
  set radius: relative length dictionary,
  // 内边距
  set inset: relative length dictionary,
  // 外边距
  set outset: relative length dictionary,
  set none content,
) -> content
```

#Code

```
// Without content.
#rect(width: 35%, height: 30pt)
```

```
// With content.  
#rect[  
  Automatically sized \  
  to fit the content.  
]
```

#Demo



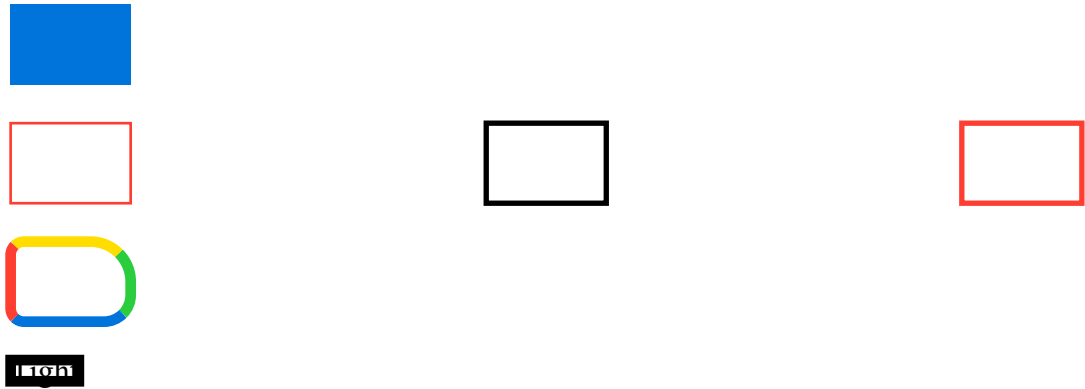
Automatically sized
to fit the content.

#Code

```
#rect(fill: blue)  
#stack(  
  dir: ltr,  
  spacing: 1fr,  
  rect(stroke: red),  
  rect(stroke: 2pt),  
  rect(stroke: 2pt + red),  
)  
  
#set rect(stroke: 4pt)  
#rect(  
  radius: (  
    left: 5pt,  
    top-right: 20pt,  
    bottom-right: 10pt,  
  ),  
  stroke: (  
    left: red,  
    top: yellow,  
    right: green,  
    bottom: blue,  
  ),  
)
```



```
)  
  
#rect(inset: 0pt)[Tight]
```

#Demo

14.3 正方形square

正方形，参数解释同上

#Func

```
square(  
  set size: auto length,  
  set width: auto relative length,  
  set height: auto relative length,  
  set fill: none color,  
  set stroke: none auto length color dictionary stroke,  
  set radius: relative length dictionary,  
  set inset: relative length dictionary,  
  set outset: relative length dictionary,  
  set none content,  
) -> content
```

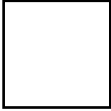
#Code

```
// Without content.  
#square(size: 40pt)  
  
// With content.  
#square[  
  Automatically \
```

```
sized to fit.
```

```
]
```

#Demo



```
Automatically  
sized to fit.
```

14.4 圆形circle

绘制带有内容的圆圈

#Func

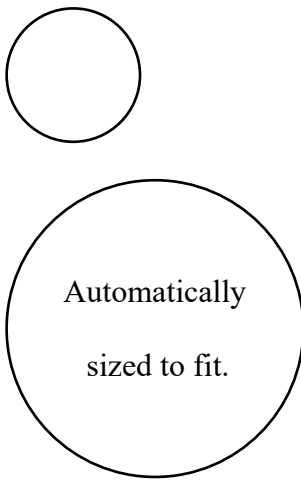
```
circle(  
  // 圆半径，与weight和height互斥  
  set radius: length,  
  // 相对于父容器的宽度，互斥  
  set width: auto relative length,  
  // 相对于父容器的高度，互斥  
  set height: auto relative length,  
  // 内部填充  
  set fill: none color,  
  // 圆圈格式  
  set stroke: none auto length color stroke,  
  // 内边距  
  set inset: relative length dictionary,  
  // 外边距  
  set outset: relative length dictionary,  
  set none content,  
) -> content
```

#Code

```
// Without content.
#circle(radius: 25pt)

// With content.
#circle[
  #set align(center + horizon)
  Automatically \
  sized to fit.
]
```

#Demo



14.5 路径path

使用路径绘制图形，通过点列表绘制，曲线为贝塞尔曲线 (v0.2新增)

#Func

```
path(
  // 如何填充路径
  fill: none | color,
  // 路径格式设置
  stroke: none | auto | length | color | stroke,
  // 是否闭合路径。该曲线将考虑相邻的控制点。如果想用一条直线结束，只需添加一个
  // 与起点相同的最后一个点。
  closed: boolean,
  // 路径顶点设置，可以用三种方式定义：
  // - 给定线或多边形函数的常规点。
  // - 两个点的数组，第一个是顶点，第二个是控制点。控制点相对于顶点表示，镜像得
```

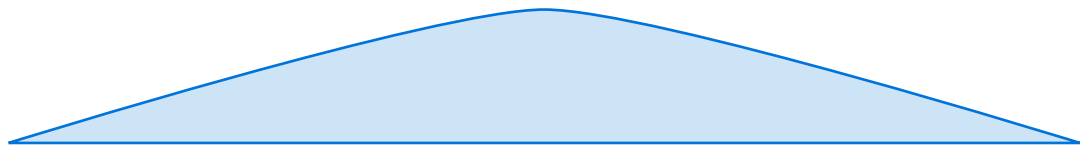
到第二个控制点。给定的控制点是影响进入该顶点的曲线的点（即使是第一个点）。镜像控制点影响从该顶点出来的曲线。

```
// - 三个点的数组，第一个是顶点，下一个是控制点（分别是曲线进出的控制点）
..array,
) -> content
```

#Code

```
#path(
  fill: blue.lighten(80%),
  stroke: blue,
  closed: true,
  (0pt, 50pt),
  (100%, 50pt),
  ((50%, 0pt), (40pt, 0pt)),
)
```

#Demo



14.6 多边形polygon

一个封闭的多边形。多边形由其角点定义并自动闭合。

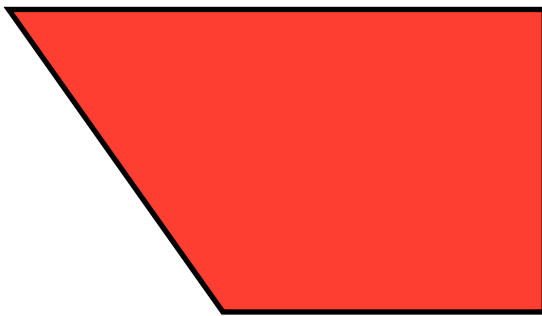
#Func

```
polygon(
  // 如何填充多边形。目前所有的多边形都是根据non-zero winding rule填充的。
  set fill: none color,
  // 多边形边界格式设置
  set stroke: none length color stroke,
  // 多边形的顶点。每个点都指定为两个相对长度的数组。
  ..array,
) -> content
```

#Code

```
#polygon(  
  fill: red,  
  stroke: 2pt + black,  
  (0pt, 0pt),  
  (50%, 0pt),  
  (50%, 4cm),  
  (20%, 4cm),  
)
```

#Demo



14.7 椭圆ellipse

绘制椭圆，参数同上

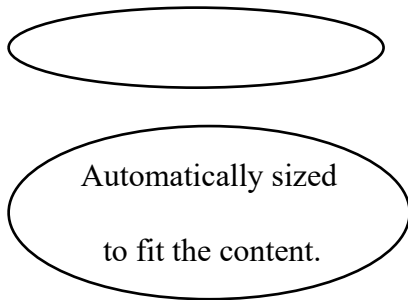
```
#Func  
  
ellipse(  
  set width: auto relative length,  
  set height: auto relative length,  
  set fill: none color,  
  set stroke: none auto length color stroke,  
  set inset: relative length dictionary,  
  set outset: relative length dictionary,  
  set none content,  
) -> content
```

#Code

```
// Without content.  
#ellipse(width: 35%, height: 30pt)  
  
// With content.
```

```
#ellipse[
  #set align(center)
  Automatically sized \
  to fit the content.
]
```

#Demo



14.8 图片image

参考之前章节 Section 2 的描述

第 15 章 Meta

文档结构化、内省和元数据配置。在这里，可以找到构建文档并与该结构交互的函数。这包括章节标题和图表、书目管理、交叉引用等。此外，此类别描述了 Typst 内省功能：使用计数器功能，您可以访问和操作页面、部分、图形和方程式计数器或创建自定义计数器。查询功能让您可以搜索文档中的元素，以构建图表列表或显示当前章节标题的标题等内容。

15.1 文献库导入bibliography

导入参考文献库

可以通过使用以下两种格式之一的参考书目文件的路径调用此函数来创建新的参考书目：

- Hayagriva `.yaml`

Hayagriva 是一种新的参考书目文件格式，专为与 Typst 一起使用而设计。

- BibLaTeX `.bib`

只要在文档中的某处添加参考书目，就可以开始使用引用语法 (`@key`) 或显式调用 `citation` 函数 (`#cite("key")`) 来引用内容。参考书目将仅显示文档中引用的作品的条目。

#Func

```
bibliography(
  // 路径
  path: string,
  // 设置参考文献
  set title: none auto content,
  // 设置引用格式 当前支持: apa author-date ieee mla
  // v0.1版本"author-date"和"author-title"改名为"chicago-author-date"和"chicago-author-title"
  set style: string,
) -> content
```

#Code

```
This was already noted by
pirates long ago. @arrgh
```

```
Multiple sources say ...
#cite("arrgh", "netwok").

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

15.2 文献引用cite

引用参考书目中的作品。在开始引用之前，您需要在文档的某处添加参考书目。

#Func

```
cite(
  // 标识应在参考书目中引用的元素的引用键。参考语法仅支持单个键。
  ..string,
  // 引文的补充，例如页码或章节编号。在参考语法中，可以在方括号中添加补充：
  set none content,
  // 引文是否应包括括号。
```

```

    set brackets: boolean,
    // 引用样式 numerical alphanumerical chicago-author-date chicago-notes chicago-author-
    title keys
    // v0.1版本"author-date"和"author-title"改名为"chicago-author-date"和"chicago-author-title"
    set style: auto string,
  ) -> content

```

#Code

This was already noted by
pirates long ago. @arrgh

Multiple sources say ...
#cite("arrgh", "netwok").

#bibliography("works.bib")

#Code

This has been proven over and
over again. @distress[p.~7]

#bibliography("works.bib")

#Code

```

#set cite(brackets: false)

@network follow these methods
in their work ...

#bibliography(
  "works.bib",
  style: "author-date",
)

```

#Code

```

#set cite(style: "alphanumerical")
Alphanumerical references.
@network

```



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

15.3 文献引用ref

对标签或参考书目的引用。引用函数生成对标签的文本引用。例如，对标题的引用将产生一个适当的字符串，例如“Section 1”，用于对第一个标题的引用。参考文献也是指向相应元素的链接。参考语法也可用于引用参考书目。

#Func

```
ref(
  label,
  set supplement: none auto content function,
) -> content
```

#Code

```
#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.
$ 0(n) = 2^n $ <slow>
#bibliography("works.bib")
```

#Code

```
#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.

```

15.4 图片 figure

见之前章节 Section 2 描述

15.5 脚注 footnote

使用脚注，可以在当前页面提供额外的标记与参考。脚注将会插入一个上标数字，用于链接到底部的注释。注释在文档中是按顺序编号的，并且可以跨越多个页面。

下文描述了如何自定义脚注列表中的条目外观。脚注是以普通的上标实现的，因此可以基于 `super` 函数自定义规则。

#Code

Check the docs for more details.
[#footnote\[https://typst.app/docs\]](https://typst.app/docs)

#Demo

Check the docs for more details.¹

脚注会自动以上标的形式依附在前面的单词。即使在编标记前面还有空格。如果有强制使用空格的地方，可以使用字符 `#" "` 或者显示声明水平间距 `#h()`。

¹<https://typst.app/docs>

#Func

```

footnote(
  // 如何编号
  // 默认情况下, 脚注编号是连续的。如果需要自定义每页的脚注编号, 可以在每页开头
  // 设置脚注`counter`
  numbering: string | function,
  content,
) -> content

```

#Code

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

Footnotes:

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

#Demo

Footnotes:[†],[‡]

15.5.1 footnote.entry

自定义脚注列表中的条目。

此函数不能直接调用。但是可以用于设置和显示规则以自定义脚注列表。

此文档编写中, 发现该函数只有在文章开头定义才会生效, 原因未知!

#Func

```

footnote.entry(
  // 脚注内容, 他的位置可以用于确定counter状态
  note: content,
  // 设置脚注与正文之间的分隔符
  // 默认: #line(length: 30%, stroke: 0.5pt)
  separator: content,
  // 正文与分隔符之间的间距
  // Default: 1em
  clearance: length,
  // 脚注之间的间距
  // Default: 0.5em
)

```

[†]Star

[‡]Dagger

```
// gap: length,
// 脚注缩进
// Default: 1em
indent: length,
) -> content
```

#Code

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

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

#Demo

My footnote listing⁴ has red text!

#Code

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

Customized [#footnote](#)[Hello]
 listing [#footnote](#)[World! 🌍]

#Demo

Customized⁵ listing⁶

#Code

⁴It's down here

⁵Hello

⁶World! 🌍

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

Testing a different separator.

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

#Demo

Testing a different separator.⁷

#Code

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

Footnotes also need ...

```
#footnote[  
  ... some space to breathe.  
]
```

#Demo

Footnotes also need ...⁸

#Code

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

Footnotes:

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

#Demo

Footnotes:⁹ ¹⁰

⁷Unconventional, but maybe not that bad?

⁸... some space to breathe.

⁹Spaced

¹⁰Apart

#Code

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

Footnotes:

```
#footnote[No],
#footnote[Indent]
```

#Demo

Footnotes:^{11, 12}

15.6 标题heading

章节标题。通过使用标题，可以将文档组织成多个部分。每个标题都有一个级别，从一级开始，向上无界。此级别指示以下内容（部分、小节等）的逻辑作用。顶级标题表示文档的顶级部分（不是文档的标题）。Typst 可以自动为您的标题编号。要启用编号，请指定您希望如何使用编号模式或功能对标题进行编号。除了编号之外，Typst 还可以自动为您生成所有标题的大纲。要从此大纲中排除一个或多个标题，您可以将 `outlined` 参数设置为 `false`。语法糖：通过以一个或多个 `=` 开始一行，后跟一个空格来创建。等号的数量决定了标题的逻辑嵌套深度。

#Func

```
heading(
  // 标题深度
  set level: integer,
  // 编号格式
  set numbering: nonestringfunction,
  // 是否在目录显示
  set outlined: boolean,
  content,
) -> content
```

#Code

¹¹No

¹²Indent

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

```
= Introduction
```

In recent years, ...

```
== Preliminaries
```

To start, ...

```
#Code
```

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

```
= A section
```

```
== A subsection
```

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

```
#Code
```

```
#outline()
```

```
#heading[Normal]
```

This is a normal heading.

```
#heading(outlined: false)[Hidden]
```

This heading does not appear
in the outline.

15.7 布局Layout

对当前外部容器大小(如果none则设置页面)进行修改（宽度和高度）

给定函数必须接受单一参数，也即高和宽使用字典的形式传递。

```
#Code
```

```
#let text = lorem(30)
```

```
#layout(size => style(styles => [
  #let (height,) = measure(
    block(width: size.width, text),
    styles,
  )
])
```

```

This text is #height high with
the current page width: \
#text
1))

```

#Demo

This text is 63.84pt high with the current page width:

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

如果布局调用放置在宽度为 800pt、高度为 400pt 的框内，则指定的函数将被赋予参数（宽度：800pt，高度：400pt）。如果它直接放在页面中，它会收到页面的尺寸减去边距。这在与测量结合时最有用。

您还可以使用此函数将比率解析为固定长度。如果您正在构建自己的布局抽象，这可能会派上用场。

#Code

```

#layout(size => {
  let half = 50% * size.width
  [Half a page is #half wide.]
})

```

#Demo

Half a page is 200.78pt wide.

请注意，如果页面宽度或高度之一分别为自动，则此函数将提供无限宽度或高度。

#Func

用外部容器的大小来调用的函数。它的返回值显示在文档中。

容器的尺寸是以字典的形式给出的，有宽度和高度两个键。

布局返回的内容每次出现在文档中时，都会调用这个函数一次。这使得生成的内容有可能取决于它所在的容器的大小。

```
layout(function) -> content
```


15.8 编号 numbering

编号定义应如何将数字序列显示为内容。它通过模式字符串或任意函数定义。编号模式由计数符号、它们的前缀和一个后缀组成，实际数字被替换为计数符号。前缀和后缀按原样重复。

#Func

```
numbering(
  // 定义编号。计数符号是 1、a、A、i、I 和 *。在给定的情况下，它们被序列中的数字
  // 替换。
  // * 字符表示应使用符号进行计数，顺序为*、†、‡、§、¶、||。如果项目超过六项，则
  // 使用多个符号表示数量。
  numbering: string function,
  // 要应用编号的数字。必须是积极的。
  // 如果编号是一种模式并且给出的数字多于计数符号，则重复最后一个计数符号及其
  // 前缀。
  numbers: ..integer,
) -> any
```

#Code

```
#numbering("1.1)", 1, 2, 3) \
#numbering("1.a.i", 1, 2) \
#numbering("I - 1", 12, 2) \
#numbering(
  (..nums) => nums
  .pos()
  .map(str)
  .join(".") + ")",
  1, 2, 3,
)
```

#Demo

```
1.2.3)
1.b
XII - 2
1.2.3)
```

15.9 链接link

超链接 `link` 函数使其位置主体参数可点击并将其链接到 `dest` 参数指定的目的地。默认情况下，链接的样式与普通文本没有任何不同。但是，您可以使用显示规则轻松应用您选择的样式。语法糖：以 `http://` 或 `https://` 开头的文本会自动变成链接。

#Func

```
link(
  // 指向目的地
  // 要链接到网页，dest 应该是一个有效的 URL 字符串。
  // 要链接到文档的另一部分，dest 可以采用以下两种形式之一：locate函数或具有整数
  // 类型的页面键和长度类型的 x 和 y 坐标的字典。页数从一页开始，坐标相对于页面的左上
  // 角。
  dest: string dictionary location,
  // 链接的显示内容。
  body: content,
) -> content
```

#Code

```
#show link: underline

https://example.com \
#link("https://example.com") \
#link("https://example.com") [
  See example.com
]
```

#Demo

```
https://example.com

https://example.com

See example.com
```

#Code

```
#link("mailto:hello@typst.app") \
#link((page: 1, x: 0pt, y: 0pt))[
  Go to top
]
```

#Demo

hello@typst.app

Go to top

15.10 访问locate

提供对文档内容位置的访问。与queries、counters、state和links结合使用很有用。

#Func

locate(function) -> content

#Code

```
#locate(loc => [
  My locatation: \
  #loc.position()!
])
```

#Demo

My locatation:

(page: 163, x: 108.2pt, y: 560.3pt) !

15.10.1 methods

#Func

// 返回当前页码。不会返回此位置的页面计数器的值，而是返回真实的页码（从一开始）。
 // 如果想知道页面计数器的值，请改用 counter(page).at(loc)。
 value.page() -> integer
 // 返回包含页码和此位置的 x、y 位置的字典。页码从 1 开始，坐标从页面的左上角开始测量。

// 只需要页码，请改用 `page()`，因为它允许 Typst 跳过不必要的工作。
`value.position()` -> dictionary

15.11 大纲outline

生成大纲/目录，此函数生成文档中所有标题的列表，直到给定深度。

#Func

```
outline(
  // 设置目录标题，none表示无目录标题
  set title: none auto content,
  // 设置包含在大纲中的元素类型
  set target: label function selector
  // 显示标题的最大深度
  set depth: none integer,
  // 是否缩进副标题以将其编号的开头与其父母的标题对齐。这仅在设置标题编号时有
  效。
  set indent: boolean,
  // 标题和页码之间的空间。可以设置为 none 以禁用填充。默认是重复[.]。
  set fill: none content,
) -> content
```

#Code

```
#outline()

= Introduction
#lorem(5)

= Prior work
#lorem(10)
```

#Code

```
#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)
```

```
#Code  
#outline(  
  title: [List of Figures],  
  target: figure.where(kind: image),  
)  
  
#figure(  
  image("tiger.jpg"),  
  caption: [A nice figure!],  
)
```

```
#Code  
#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],
)

```

#Code

```

#set heading(numbering: "1.")
#outline(depth: 2)

```

= Yes

Top-level section.

== Still

Subsection.

=== Nope

Not included.

#Code

```

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

```

= A New Beginning

15.12 计数器counter

count 页面、元素等。使用 counter func，可以访问和修改页面、标题、图表等的计数器。此外，您可以为其他要计数的事物定义自定义计数器。

15.12.1 显示counter

要显示 heading counter 的当前值，可以调用 counter func 并将 key 设置为 heading，然后调用 counter 的 display method。要查看任何输出，您还必须启用 heading number。

display 函数可选地接受一个参数，告诉它如何格式化计数器。

#Code

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

= Introduction

Some text here.

= Background

The current value is:

```
#counter(heading).display()
```

Or in roman numerals:

```
#counter(heading).display("I")
```

15.12.2 修改counter

要修改counter，可以使用 `step` 和 `update` 方法：

- `step` 方法将counter的值增加 1。因为counter可以有多个级别（在节、小节等的标题的情况下），所以 `step` 方法可以选择使用级别参数。如果给定，则counter在给定深度步进。
- `update` 方法允许您任意修改counter。可以给它一个整数（或多个级别的倍数）。为了获得更大的灵活性，您还可以为其提供一个获取当前值并返回新值的函数。

#Code

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

= Introduction

```
#counter(heading).step()
```

= Background

```
#counter(heading).update(3)
```

```
#counter(heading).update(n => n * 2)
```

= Analysis

Let's skip 7.1.

```
#counter(heading).step(level: 2)
```

== Analysis

Still at `#counter(heading).display()`.

15.12.3 如何步进 (v0.1新增)

当您定义和使用自定义counter时，一般情况下，您应该先step counter然后display it。这样，计数器的step可以取决于它步进的元素。如果您正在为定理编写一个计数器，那么您的定理定义将首先包括计数器步骤，然后才显示计数器和定理的内容。

#Code

```
#let c = counter("theorem")
#let theorem(it) = block[
  #c.step()
  *Theorem #c.display():* #it
]

#theorem[$1 = 1$]
#theorem[$2 < 3$]
```

#Demo

Theorem 1: $1 = 1$

Theorem 2: $2 < 3$

机理解释：heading counter的更新取决于heading的层级。通过在heading之前设置heading的step，当到二级heading时，我们可以将step从 1 修正为 1.1。如果我们在heading之后使用step，那么程序不会知道step的值。这是因为counters必须在计数元素之前step，并且从0开始。

15.12.4 页面计数器pagecounter

pagecounter是特殊的。它会在每个分页符处自动步进。但与其他counter一样，也可以手动步进。例如，您可以为序言使用罗马页码，然后为主要内容切换为阿拉伯文页码并将pagecounter重置为 1。

#Code

```
#set page(numbering: "(i)")

= Preface
```


The preface is numbered with roman numerals.

```
#set page(numbering: "1 / 1")
#counter(page).update(1)
```

= Main text

Here, the counter is reset to one.
We also display both the current page and total number of pages in Arabic numbers.

15.12.5 自定义counters

自定义counter，使用字符串作为key调用counter函数。

#Code

```
#let mine = counter("mycounter")
#mine.display() \
#mine.step()
#mine.display() \
#mine.update(c => c * 3)
#mine.display() \
```

#Demo

0

1

3

15.12.6 time travel

Counters can travel through time! You can find out the final value of the counter before it is reached and even determine what the value was at any particular location in the document.

#Code

```
#let mine = counter("mycounter")
```

```

= Values
#locate(loc => {
  let start-val = mine.at(loc)
  let elements = query(<intro>, loc)
  let intro-val = mine.at(
    elements.first().location()
  )
  let final-val = mine.final(loc)
  [Starts as: #start-val \
    Value at intro is: #intro-val \
    Final value is: #final-val \ ]
})

#mine.update(n => n + 3)

= Introduction <intro>
#lorem(10)

#mine.step()
#mine.step()

```

上述代码做了如下几件事：

- 调用 `locate` 函数获取当前位置。然后将该位置传递给 `counter`。 `at` 方法总是返回一个数组，因为计数器可以有多个级别。由于计数器从 1 开始，因此第一个值为 (1,)。
- 查询文档中所有带有 `intro` 标签的元素。结果是一个数组，从中提取第一个元素的位置。然后我们在该位置查找计数器的值。计数器的第一次更新将其设置为 $1 + 3 = 4$ 。因此在介绍标题处，该值为 (4,)。
- 最后，在 `counter` 上调用 `final` 方法。它告诉我们在文档末尾 `counter` 的值是多少。还需要给它一个位置来证明我们在一个 `locate` 调用中，但哪个并不重要。标题之后是对 `step()` 的两次调用，因此最终值为 (6,)。

#Func

```
counter(key: string | label | function) -> counter
```

15.12.7 Methods**#Func**

```
// 显示当前counter值
```

```
value.display(string | function) -> content
```

// 将计数器的值增加一。更新将在返回内容插入文档的位置生效。如果您不将输出放入文档中，则什么也不会发生！例如，如果您编写 `let _ = counter(page).step()`，就会出现这种情况。

```
value.step(level: integer) -> content
```

```
// 更新计数器的值
```

```
value.update(integer | array | function) -> content
```

// 获取给定位置的计数器值。始终返回一个整数数组，即使计数器只有一个数字。

```
value.at(location) -> array
```

// 获取文档末尾的计数器值。始终返回一个整数数组，即使计数器只有一个数字。

```
value.final(location) -> array
```

15.13 查找query

在文档中查找元素。`query`功能使您可以在文档中搜索特定类型或具有特定标签的元素。要使用它，您首先需要使用 `locate` 函数检索当前文档位置。然后您可以决定是否要查找所有元素，仅查找该位置之前的元素，还是仅查找该位置之后的元素。

15.13.1 查找元素find elements

在下面的示例中，我们创建了一个自定义页眉，以小写大写形式显示文本“-Typst Academy”和当前部分标题。在第一页上，节标题被省略，因为标题在第一个节标题之前。为了实现这种布局，我们调用 `locate` 然后查询当前位置之后的所有标题。在这种情况下，我们传递给 `locate` 的函数被调用两次：每页调用一次。

- 在第一页上，查询当前位置之前的所有标题会产生一个空数组：没有以前的标题。我们检查这种情况，然后只显示“-Typst Academy”。

- 对于第二页，我们从查询结果中检索最后一个元素。这是当前位置之前的最新标题，因此，它是我们当前所在部分的标题。我们通过 `body` 字段访问其内容并将其显示在“Typst Academy”旁边。

#Code

```
#set page(header: locate(loc => {
  let elems = query(
    selector(heading).before(loc),
    loc,
  )
  let academy = smallcaps[
    Typst Academy
  ]
  if elems == () {
    align(right, academy)
  } else {
    let body = elems.last().body
    academy + h(1fr) + emph(body)
  }
}))
```

= Introduction

```
#lorem(23)
```

= Background

```
#lorem(30)
```

= Analysis

```
#lorem(15)
```

#Func

```
query(
  // heading figure equation reference label
  target: label function,
  location,
) -> content
```

为了解决您的所有查询，Typst 多次评估和布局文档的各个部分。但是，不能保证您的查询实际上可以完全解决。如果您不小心，查询可能会影响自身——导致结果永远不稳定。在下面的示例中，我们查询文档中的所有标题。然后我们生成尽可能多的标题。一开始，只有一个标题，名为 **Real**。这样 `count` 为 1，生成一个 **Fake heading**。Typst 看到查询的结果已经改变并再次处理它。这次，`count` 为 2，生成了两个 **Fake headings**。这种情况一直在继续。正如我们所见，输出有五个标题。这是因为 Typst 在五次尝试后就放弃了。一般来说，你应该尽量不要写影响自己的查询。同样的警告也适用于其他内省功能，如计数器和状态。

```
#Code

= Real
#locate(loc => {
  let elems = query(heading, loc)
  let count = elems.len()
  count * [= Fake]
})
```

15.14 文档document

文档的根元素及其元数据。所有文档都自动包装在文档元素中。该元素的主要用途是在设置规则中使用它来指定文档元数据。使用此功能设置的元数据不会在文档中呈现。相反，它嵌入在已编译的 PDF 文件中。

```
#Func

document(
  // 设置文档标题
  set title: none string,
  // 文档作者
  set author: string array,
) -> content
```

第 16 章 计算

Typst因为支持函数，所以支持数值的计算和处理。这些函数是calc module的组成，默认情况下是不会导入的。除了如下的函数以外，calc还定义了常量 `pi`、`e`、`inf`、`nan`。

#Func

```
// 计算给定数的绝对值
abs(integer | float | length | angle | ratio | fraction) -> any
// 计算sin值 当使用整数或浮点数调用时，它们将被解释为弧度。
sin(integer | float | angle) -> float
// 计算cos值 当使用整数或浮点数调用时，它们将被解释为弧度。
cos(integer | float | angle) -> float
// 计算tan值 当使用整数或浮点数调用时，它们将被解释为弧度。
tan(integer | float | angle) -> float
// 计算arccos值
acos(integer | float) -> angle
// 计算arcsin值
asin(integer | float) -> angle
// 计算arctan值
atan(integer | float) -> angle
// 计算角度的双曲正弦值 当使用整数或浮点数调用时，它们将被解释为弧度
sinh(integer | float | angle) -> float
// 计算角度的双曲余弦值 当使用整数或浮点数调用时，它们将被解释为弧度
cosh(integer | float | angle) -> float
// 计算角度的双曲正切值 当使用整数或浮点数调用时，它们将被解释为弧度
tanh(integer | float | angle) -> float
// 计算对数 如果未指定底数，则以 10 为底数计算对数。 base:底数
log(integer | float,base: float,) -> float
// 最大值
max(..any) -> any
// 最小值
min(..any) -> any
// 计算二次式系数
binom(integer,integer,) -> integer
// 阶乘
fact(integer) -> integer
// 求两个数的模
mod(dividend: integer | float,divisor: integer | float,) -> integer | float
// 同上，v0.3新增用于替换mod
rem(dividend: integer | float,divisor: integer | float,) -> integer | float
```

```

// 返回数字的整数部分。如果数字已经是整数，则返回原样。
trunc(integer float) -> integer
// 返回数字的小数部分。如果数字是整数，则返回 0。
fract(integer float) -> integerfloat
// 指数计算
pow(base: integer | float,exponent: integer | float,) -> integer | float
// 判断整数是否为奇数
odd(integer) -> boolean
// 判断整数是否为偶数
even(integer) -> boolean
// 将数字舍入到最接近的整数。可以指定小数位数。
round(integer | float,digits: integer,) -> integer | float
// 将数字向下取整
floor(integer | float) -> integer
// 将数字向上取整。如果数字已经是整数，则返回原样。
ceil(integer | float) -> integer
// 计算两个整数的最小公倍数。
lcm(integer,integer,) -> integer
// 计算一个排列。
perm(integer,integer,) -> integer
// 平方根
sqrt(integer | float) -> float
// 将数字夹在最大值与最小值之间？
clamp(integer | float,integer | float,integer | float,) -> integer | float

```

#Code

```

#calc.abs(-5) \
#calc.abs(5pt - 2cm) \
#calc.abs(2fr) \
#calc.sqrt(16) \
#calc.sqrt(2.5) \
#calc.pow(2, 3) \
#calc.log(100) \
#calc.rem(20, 6) \
#calc.rem(1.75, 0.5) \
#calc.fact(5) \
#calc.fract(-3.1)\
#calc.lcm(96, 13) \
#calc.perm(10, 5)\

```

```
#calc.trunc(3) \  
#calc.trunc(-3.7) \  
#calc.trunc(15.9) \  
#calc.clamp(5, 0, 4)
```

#Demo

5

51.69pt

2fr

4

1.5811388300841898

8

2

2

0.25

120

-0.100000000000000009

1248

30240

3

-3

15

4

#Code

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

```



```
#calc.cos(1.5) \
#calc.cos(90deg) \
#calc.tan(1.5) \
#calc.tan(90deg)
#calc.acos(0) \
#calc.acos(1) \
#calc.asin(0) \
#calc.asin(1) \
#calc.atan(0) \
#calc.atan(1) \
#calc.sinh(0) \
#calc.sinh(45deg) \
#calc.cosh(0) \
#calc.cosh(45deg) \
#calc.tanh(0) \
#calc.tanh(45deg) \
```

#Demo

0.9974949866040544

1

0.00000000000000006123233995736766

0.0707372016677029

0.00000000000000006123233995736766

14.101419947171719

16331239353195370 90deg

0deg

0deg

90deg

0deg

45deg

0

0.8686709614860095

1

1.324609089252006

0

0.6557942026326724

#Code

```
#calc.max(1, -3, -5, 20, 3, 6) \  
#calc.max("typst", "in", "beta") \  
#calc.min(1, -3, -5, 20, 3, 6) \  
#calc.min("typst", "in", "beta") \  
#calc.odd(4) \  
#calc.odd(5) \  
#range(10).filter(calc.odd) \  
#calc.even(4) \  
#calc.even(5) \  
#range(10).filter(calc.even) \  
#calc.round(3.1415, digits: 2) \  
#calc.floor(500.1) \  
#calc.ceil(500.1) \  

```

#Demo

20

typst -5

beta

false

true

(1, 3, 5, 7, 9)

true

false

(0, 2, 4, 6, 8)

3.14

500

501

第 17 章 Construct

Typst支持不同类型值的构造和转换。

17.1 int

将值转换为整数

- 布尔值转换为0或1
- 浮点数向下取整
- 字符串以10为基数解析

#Func

```
int(boolean | integer | float | string) -> integer
```

#Code

```
#int(false) \  
#int(true) \  
#int(2.7) \  
#{ int("27") + int("4") }
```

#Demo

0
1
2
31

17.2 float

将值转为浮点数

- 布尔值转为0.0或1.0
- 整数向下取整为64位浮点数
- 字符串以 10 为基数解析为最接近的 64 位浮点数。支持指数符号。

#Func

```
float(boolean | integer | float | string) -> float
```

#Code

```
#float(false) \
#float(true) \
#float(4) \
#float("2.7") \
#float("1e5")
```

#Demo

```
0
1
4
2.7
100000
```

17.3 range

创建一个由数字序列组成的数组。如果你只传递一个位置参数，它被解释为范围的结束。如果你传递两个，它们描述了范围的开始和结束。

#Func

```
range(
  // 起始, 包含当前位置
  start: integer,
  // 结束, 不包含当前位置
  end: integer,
```

```
// 步长
step: integer,
) -> array
```

#Code

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

#Demo

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

17.4 regex

从字符串创建正则表达式。结果可用作show 规则选择器，或者string方法 (find, split, replace)。

#Func

```
regex(string) -> regex
```

#Code

```
// Works with show rules.
#show regex("\\d+"): set text(red)

The numbers 1 to 10.

// Works with string methods.
```

```
#("a,b;c"
  .split(regex("[,;]")))
```

#Demo

The numbers 1 to 10. ("a", "b", "c")

17.5 cmyk

见之前章节 Section 10.11.3 描述。

17.6 RGB

见之前章节 Section 10.11.2 描述

17.7 luma

见之前章节 Section 10.11.4 描述

17.8 string

见之前章节 Section 10.14 描述

17.9 Label

从字符串创建label。将label插入到内容中会将其附加到最近的不是空格的前一个元素。然后，可以通过标签引用该元素并设置其样式。语法糖：可以通过将其名称括在尖括号中来创建标签。这适用于标记和代码。

#Func

```
label(string) -> label
```

#Code

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

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

#Demo

Heading

Strong

17.10 symbol

见之前章节 Section 10.13 描述

第 18 章 导入文件

从外部读取数据文件

18.1 csv

从 CSV 文件中读取结构化数据。CSV 文件将被读取并解析为一个二维字符串数组：CSV 文件中的每一行将表示为一个字符串数组，所有行将被收集到一个数组中。标题行不会被删除。

#Func

```
csv(  
  // csv路径  
  path:string,  
  // 分隔符  
  delimiter: string,  
) -> array
```

#Code

```
#let results = csv("data.csv")  
  
#table(  
  columns: 2,  
  [*Condition*], [*Result*],  
  ..results.flatten(),  
)
```

18.2 json

从 JSON 文件中读取结构化数据。该文件必须包含有效的 JSON 对象或数组。JSON 对象将被转换为 Typst 字典，JSON 数组将被转换为 Typst 数组。字符串和布尔值将被转换为 Typst 等价物，null 将被转换为 none，数字将被转换为浮点数或整数，具体取决于它们是否为整数。该函数返回字典或数组，具体取决于 JSON 文件。

#Func

```
json(path: string) -> array dictionary
```

#Code

```
#let forecast(day) = block[
  #box(square(
    width: 2cm,
    inset: 8pt,
    fill: if day.weather == "sunny" {
      yellow
    } else {
      aqua
    },
    align(
      bottom + right,
      strong(day.weather),
    ),
  ))
  #h(6pt)
  #set text(22pt, baseline: -8pt)
  #day.temperature °#day.unit
]

#forecast(json("monday.json"))
#forecast(json("tuesday.json"))
```

18.3 plain text

读取文本文件并返回字符串

#Func

```
read(path: string) -> string
```

#Code

```
#let text = read("data.html")
```

An example for a HTML file:\

```
#raw(text, lang: "html")
```

18.4 toml

从 TOML 文件中读取结构化数据。该文件必须包含有效的 TOML 表。TOML 表将转换为 Typst 字典，TOML 数组将转换为 Typst 数组。字符串和布尔值将被转换为 Typst 等价物，数字将被转换为浮点数或整数，具体取决于它们是否为整数。目前，日期时间将被转换为字符串，因为 Typst 还没有内置的日期时间。示例中的 TOML 文件包含一个表，其中包含键标题、版本和作者。

#Func

```
toml(string) -> dictionary
```

#Code

```
#let details = toml("details.toml")
```

```
Title: #details.title \
```

```
Version: #details.version \
```

```
Authors: #(details.authors
```

```
  .join(", ", last: " and "))
```

18.5 xml

从 XML 文件中读取结构化数据。XML 文件被解析为字典和字符串数组。XML 节点可以是元素或字符串。元素表示为具有以下键的字典：

- **tag**: 作为字符串的元素名称。

- **attrs**: 元素属性的字典，作为字符串。
- **children**: 元素的子节点数组。

示例中的 XML 文件包含一个带有多个 `article` tags 的 root `news` tag。每篇文章都有 `title`, `author`, and `content` tag。 `content` tag 包含一个或多个段落，表示为 `p` 标签。

#Func

```
xml(path: string) -> array
```

#Code

```
#let findChild(elem, tag) = {
  elem.children
    .find(e => "tag" in e and e.tag == tag)
}

#let article(elem) = {
  let title = findChild(elem, "title")
  let author = findChild(elem, "author")
  let pars = findChild(elem, "content")

  heading(title.children.first())
  text(10pt, weight: "medium")[
    Published by
    #author.children.first()
  ]

  for p in pars.children {
    if (type(p) == "dictionary") {
      parbreak()
      p.children.first()
    }
  }
}

#let data = xml("example.xml")
#for child in data.first().children {
  if (type(child) == "dictionary") {
    article(child)
  }
}
```

```
}
}
```

18.6 yaml

从 YAML 文件中读取结构化数据。该文件必须包含有效的 YAML 对象或数组。YAML 映射将转换为 Typst 字典，YAML 序列将转换为 Typst 数组。字符串和布尔值将被转换为 Typst 等价物，空值（`null`、或空）将被转换为无，数字将被转换为浮点数或整数，具体取决于它们是否为整数。请注意，不是字符串的映射键会导致条目被丢弃。自定义 YAML 标签将被忽略，但加载的值仍然存在。该函数返回字典或值或数组，具体取决于 YAML 文件。

#Func

```
yaml(path: string) -> arrayvaluedictionary
```

#Code

```
#let bookshelf(contents) = {
  for author, works in contents {
    author
    for work in works [
      - #work.title (#work.published)
    ]
  }
}

#bookshelf(yaml("scifi-authors.yaml"))
```

第 19 章 Foundations

19.1 Assert

如果不满足条件，则失败并出现错误。不在文档中产生任何输出。

#Func

```
assert(
  // 判断语句
  condition: boolean,
  // 报错信息
  message: nonestring,
) ->
```

#Code

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

#Demo

19.2 Evaluate

将字符串评估为 Typst 代码。

#Func

```
eval(
  source: string
) -> any
```

#Code

```
#eval("1 + 1") \
#eval("(1, 2, 3, 4)".len() \
#eval("[*Strong text*]")
```

#Demo

2

4

Strong text

19.3 Panic

报错

#Func

```
panic(
  ..any
) ->
```

#Code

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

19.4 Representation

以文本形式输出想要的字符串。当插入到内容中时，大多数值都显示为具有语法突出显示的等宽表示形式。例外情况是无、整数、浮点数、字符串、内容和函数。

#Func

```
repr(any) -> string
```

#Code

```
#none vs #repr(none) \
#"hello" vs #repr("hello") \
#(1, 2) vs #repr((1, 2)) \
#[*Hi*] vs #repr([*Hi*])
```

#Demo

vs none

hello vs "hello"

(1, 2) vs (1, 2)

Hi vs strong(body: [Hi])

19.5 Type

返回值类型

#Func

```
type(any) -> string
```

#Code

```
#type(12) \
#type(14.7) \
#type("hello") \
#type(none) \
#type([Hi]) \
#type(x => x + 1)
```

#Demo

integer

float

string



none

content

function

					
wj	zwj	zwnj	zws	space.nobreak	space.en
					
space.quad	space.third	space.quarter	space.sixth	space.med	space.fig
					
space.punct	space.thin	space.hair	paren.l	paren.r	paren.t
					
paren.b	brace.l	brace.r	brace.t	brace.b	bracket.l
					
bracket.l.double	bracket.r	bracket.r.double	bracket.t	bracket.b	turtle.l
					
turtle.r	turtle.t	turtle.b	bar.v	bar.v.double	bar.v.triple
					
bar.v.broken	bar.v.circle	bar.h	fence.l	fence.l.double	fence.r
					
fence.r.double	fence.dotted	angle	angle.l	angle.l.double	angle.r
					
angle.r.double	angle.acute	angle.arc	angle.arc.rev	angle.rev	angle.right
					
angle.right.rev	angle.right.arc	angle.right.dot	angle.right.sq	angle.spatial	angle.spheric
					
angle.spheric.rev	angle.spheric.top	amp	amp.inv	ast	ast.low
					
ast.basic	ast.double	ast.triple	ast.small	ast.op	ast.circle












































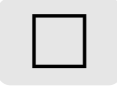




























					
ast.sq	at	backslash	backslash.circle	backslash.not	co
					
colon	colon.eq	colon.double.eq	comma	dagger	dagger.double
					
dash.en	dash.em	dash.fig	dash.wave	dash.wave.double	dash.wave.colon
					
dash.wave.circle	dot	dot.op	dot.c	dot.basic	dot.circle
					
dot.circle.big	dot.square	dot.double	dot.triple	dot.quad	excl
					
excl.double	excl.inv	excl.quest	quest	quest.double	quest.excl
					
quest.inv	interrobang	hash	hyph	hyph.minus	hyph.nobreak
					
hyph.point	hyph.soft	percent	copyright	copyright.sound	permille
					
pilcrow	pilcrow.rev	section	semi	semi.rev	slash
					
slash.double	slash.triple	dots.h	dots.h.c	dots.v	dots.down
					
dots.up	tilde	tilde.op	tilde.basic	tilde.eq	tilde.eq.not
					
tilde.eq.rev	tilde.eqq	tilde.eqq.not	tilde.neqq	tilde.not	tilde.rev
















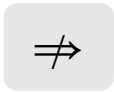





























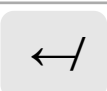


























					
tilde.rev.eqq	tilde.triple	acute	acute.double	breve	caret
					
caron	hat	diaer	grave	macron	quote.double
					
quote.single	quote.l.double	quote.l.single	quote.r.double	quote.r.single	quote.angle.l.double
					
quote.angle.l.single	quote.angle.r.double	quote.angle.r.single	quote.high.double	quote.high.single	quote.low.double
					
quote.low.single	prime	prime.rev	prime.double	prime.double.rev	prime.triple
					
prime.triple.rev	prime.quad	plus	plus.circle	plus.circle.arrow	plus.circle.big
					
plus.dot	plus.minus	plus.small	plus.square	plus.triangle	minus
					
minus.circle	minus.dot	minus.plus	minus.square	minus.tilde	minus.triangle
					
div	div.circle	times	times.big	times.circle	times.circle.big
					
times.div	times.l	times.r	times.square	times.triangle	ratio
					
eq	eq.star	eq.circle	eq.colon	eq.def	eq.delta
					
eq.equi	eq.est	eq.gt	eq.lt	eq.m	eq.not


















































					
eq.prec	eq.quest	eq.small	eq.succ	gt	gt.circle
					
gt.dot	gt.double	gt.eq	gt.eq.lt	gt.eq.not	gt.eqq
					
gt.lt	gt.lt.not	gt.neqq	gt.not	gt.ntilde	gt.small
					
gt.tilde	gt.tilde.not	gt.triple	gt.triple.nested	lt	lt.circle
					
lt.dot	lt.double	lt.eq	lt.eq.gt	lt.eq.not	lt.eqq
					
lt.gt	lt.gt.not	lt.neqq	lt.not	lt.ntilde	lt.small
					
lt.tilde	lt.tilde.not	lt.triple	lt.triple.nested	approx	approx.eq
					
approx.not	prec	prec.approx	prec.double	prec.eq	prec.eqq
					
prec.eqq	prec.napprox	prec.neqq	prec.not	prec.ntilde	prec.tilde
					
succ	succ.approx	succ.double	succ.eq	succ.eqq	succ.eqq
					
succ.napprox	succ.neqq	succ.not	succ.ntilde	succ.tilde	ident
					
ident.not	ident.strict	prop	nothing	nothing.rev	without


\complement	\in	\notin	\ni	\nexists	\ni
complement	in	in.not	in.rev	in.rev.not	in.rev.small
ϵ	\subset	$\subset\cdot$	$\subset\!\!\!\subset$	\subseteq	\nsubseteq
in.small	subset	subset.dot	subset.double	subset.eq	subset.eq.not
\sqsubseteq	\nsubseteq	\subsetneq	$\not\subset$	\sqsubset	\subsetneqq
subset.eq.sq	subset.eq.sq.not	subset.neq	subset.not	subset.sq	subset.sq.neq
\supset	$\supset\cdot$	$\supset\!\!\!\supset$	\supseteq	$\not\supseteq$	\sqsupseteq
supset	supset.dot	supset.double	supset.eq	supset.eq.not	supset.eq.sq
\nsupseteq	\supsetneq	$\not\supset$	\sqsupset	$\supseteq\!\!\!\neq$	\cup
supset.eq.sq.not	supset.neq	supset.not	supset.sq	supset.sq.neq	union
$\cup\!\!\!\rightarrow$	\bigcup	$\dot{\cup}$	$\dot{\bigcup}$	\biguplus	$\dot{\cup}$
union.arrow	union.big	union.dot	union.dot.big	union.double	union.minus
$\dot{\cup}$	$\dot{\cup}$	$\dot{\cup}$	\sqcup	\sqcup	\sqcup
union.or	union.plus	union.plus.big	union.sq	union.sq.big	union.sq.double
\cap	$\cap\!\!\!\wedge$	\bigcap	$\cap\!\!\!\cdot$	$\cap\!\!\!\cap$	\sqcap
sect	sect.and	sect.big	sect.dot	sect.double	sect.sq
\sqcap	$\cap\!\!\!\cap$	∞	∞	∂	∇
sect.sq.big	sect.sq.double	infinity	oo	diff	nabla
\sum	$\sum\!\!\!\int$	\prod	\prod	\int	$\int\!\!\!\hookrightarrow$
sum	sum.integral	product	product.co	integral	integral.arrow.hook
$\int\!\!\!\curvearrowleft$	\oint	\oint	\oint	$\int\!\!\!\rightarrow$	\iint
integral.ccw	integral.cont	integral.cont.ccw	integral.cont.cw	integral.cw	integral.double
\iiint	$\int\!\!\!\cap$	$\int\!\!\!\sqsubset$	$\int\!\!\!\S$	$\int\!\!\!\times$	\iiint
integral.quad	integral.sect	integral.sq	integral.surf	integral.times	integral.triple

					
integral.union	integral.vol	laplace	forall	exists	exists.not
					
top	bot	not	and	and.big	and.curly
					
and.dot	and.double	or	or.big	or.curly	or.dot
					
or.double	models	therefore	because	qed	compose
					
convolve	multimap	divides	divides.not	perp	perp.circle
					
parallel	parallel.circle	parallel.not	diameter	join	join.r
					
join.l	join.l.r	degree	degree.c	degree.f	smash
					
wreath	bitcoin	dollar	euro	franc	lira
					
peso	pound	ruble	rupee	won	yen
					
ballot	ballot.x	checkmark	checkmark.light	floral	floral.l
					
floral.r	notes.up	notes.down	refmark	servicemark	maltese
					
suit.club	suit.diamond	suit.heart	suit.spade	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	ellipse.stroked.h	ellipse.stroked.v	ellipse.filled.h	ellipse.filled.v
					
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	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	rect.stroked.h	rect.stroked.v	rect.filled.h
					
rect.filled.v	penta.stroked	penta.filled	hexa.stroked	hexa.filled	diamond.stroked
					
diamond.stroked.small	diamond.stroked.medium	diamond.stroked.dot	diamond.filled	diamond.filled.medium	diamond.filled.small
					
lozenge.stroked	lozenge.stroked.small	lozenge.stroked.medium	lozenge.filled	lozenge.filled.small	lozenge.filled.medium

					
star.op	star.stroked	star.filled	arrow.r	arrow.r.long	arrow.r.long.squiggly
					
arrow.r.long.bar	arrow.r.bar	arrow.r.curve	arrow.r.dashed	arrow.r.dotted	arrow.r.double
					
arrow.r.double.bar	arrow.r.double.long	arrow.r.double.long.bar	arrow.r.double.not	arrow.r.filled	arrow.r.hook
					
arrow.r.loop	arrow.r.not	arrow.r.quad	arrow.r.squiggly	arrow.r.stop	arrow.r.stroked
					
arrow.r.tail	arrow.r.triple	arrow.r.twohead.bar	arrow.r.twohead	arrow.r.wave	arrow.l
					
arrow.l.bar	arrow.l.curve	arrow.l.dashed	arrow.l.dotted	arrow.l.double	arrow.l.double.bar
					
arrow.l.double.long	arrow.l.double.long.bar	arrow.l.double.not	arrow.l.filled	arrow.l.hook	arrow.l.long
					
arrow.l.long.bar	arrow.l.long.squiggly	arrow.l.loop	arrow.l.not	arrow.l.quad	arrow.l.squiggly
					
arrow.l.stop	arrow.l.stroked	arrow.l.tail	arrow.l.triple	arrow.l.twohead.bar	arrow.l.twohead
					
arrow.l.wave	arrow.l.r	arrow.l.r.double	arrow.l.r.double.long	arrow.l.r.double.not	arrow.l.r.filled
					
arrow.l.r.long	arrow.l.r.not	arrow.l.r.stroked	arrow.l.r.wave	arrow.t	arrow.t.bar
					
arrow.t.curve	arrow.t.dashed	arrow.t.double	arrow.t.filled	arrow.t.quad	arrow.t.stop

					
arrow.t.stroked	arrow.t.triple	arrow.t.twohead	arrow.t.b	arrow.t.b.double	arrow.t.b.filled
					
arrow.t.b.stroked	arrow.b	arrow.b.bar	arrow.b.curve	arrow.b.dashed	arrow.b.double
					
arrow.b.filled	arrow.b.quad	arrow.b.stop	arrow.b.stroked	arrow.b.triple	arrow.b.twohead
					
arrow.tr	arrow.tr.double	arrow.tr.filled	arrow.tr.hook	arrow.tr.stroked	arrow.tr.bl
					
arrow.br	arrow.br.double	arrow.br.filled	arrow.br.hook	arrow.br.stroked	arrow.tl
					
arrow.tl.double	arrow.tl.filled	arrow.tl.hook	arrow.tl.stroked	arrow.tl.br	arrow.bl
					
arrow.bl.double	arrow.bl.filled	arrow.bl.hook	arrow.bl.stroked	arrow.ccw	arrow.ccw.half
					
arrow.cw	arrow.cw.half	arrow.zigzag	arrows.rr	arrows.ll	arrows.tt
					
arrows.bb	arrows.lr	arrows.lr.stop	arrows.rl	arrows.tb	arrows.bt
					
arrows.rrr	arrows.lll	arrowhead.t	arrowhead.b	harpoon.rt	harpoon.rt.bar
					
harpoon.rt.stop	harpoon.rb	harpoon.rb.bar	harpoon.rb.stop	harpoon.lt	harpoon.lt.bar
					
harpoon.lt.stop	harpoon.lt.rt	harpoon.lt.rb	harpoon.lb	harpoon.lb.bar	harpoon.lb.stop

					
harpoon.lb.rb	harpoon.lb.rt	harpoon.tl	harpoon.tl.bar	harpoon.tl.stop	harpoon.tl.bl
					
harpoon.tl.br	harpoon.tr	harpoon.tr.bar	harpoon.tr.stop	harpoon.tr.br	harpoon.tr.bl
					
harpoon.bl	harpoon.bl.bar	harpoon.bl.stop	harpoon.br	harpoon.br.bar	harpoon.br.stop
					
harpoons.rtrb	harpoons.blbr	harpoons.bltr	harpoons.lbrb	harpoons.ltlb	harpoons.ltrb
					
harpoons.ltrt	harpoons.rblb	harpoons.rtlb	harpoons.rltl	harpoons.tlbr	harpoons.tltr
					
tack.r	tack.r.long	tack.l	tack.l.long	tack.l.short	tack.l.r
					
tack.t	tack.t.big	tack.t.double	tack.t.short	tack.b	tack.b.big
					
tack.b.double	tack.b.short	alpha	beta	beta.alt	chi
					
delta	epsilon	epsilon.alt	eta	gamma	iota
					
kai	kappa	kappa.alt	lambda	mu	nu
					
ohm	ohm.inv	omega	omicron	phi	phi.alt
					
pi	pi.alt	psi	rho	rho.alt	sigma

τ	θ	ϑ	υ	ξ	ζ
tau	theta	theta.alt	upsilon	xi	zeta
A	B	X	Δ	E	H
Alpha	Beta	Chi	Delta	Epsilon	Eta
Γ	I	K	K	Λ	M
Gamma	Iota	Kai	Kappa	Lambda	Mu
N	Ω	O	Φ	Π	Ψ
Nu	Omega	Omicron	Phi	Pi	Psi
P	Σ	T	Θ	Y	Ξ
Rho	Sigma	Tau	Theta	Upsilon	Xi
Z	\aleph	\beth	\gimel	shin	\mathbb{A}
Zeta	alef	bet	gimel	shin	AA
\mathbb{B}	\mathbb{C}	\mathbb{D}	\mathbb{E}	\mathbb{F}	\mathbb{G}
BB	CC	DD	EE	FF	GG
\mathbb{H}	\mathbb{I}	\mathbb{J}	\mathbb{K}	\mathbb{L}	\mathbb{M}
HH	II	JJ	KK	LL	MM
\mathbb{N}	\mathbb{O}	\mathbb{P}	\mathbb{Q}	\mathbb{R}	\mathbb{S}
NN	OO	PP	QQ	RR	SS
\mathbb{T}	\mathbb{U}	\mathbb{V}	\mathbb{W}	\mathbb{X}	\mathbb{Y}
TT	UU	VV	WW	XX	YY
\mathbb{Z}	ℓ	h	\hbar	\AA	K
ZZ	ell	planck	planck.reduce	angstrom	kelvin
\Re	\Im				
Re	Im				