

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: *March 28, 2023*

日期: *April 4, 2023*

前言

此文档是我在学习Typst时，翻译官方manual时的一些记录。当然了，还没有更新完，同时也可以看出文档排版不太优雅。预计在我学完官方文档之后再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。我在使用过程中遇到了如下的bug或者缺陷：

- 对于中文排版不太支持，尤其是标点符号和文字之间的间距控制。
- 对于中文字体格式比如说加粗斜体等不支持
- 支持绘图
- 。 。 。 。

Casea

2023.04.04

目录

1 Align	8
2 Figure	8
3 Box	9
4 Colbreak	11
5 Columns	11
6 Enum	12
7 Grid	14
7.1 通过Figure、Grid结合绘制子图	15
8 Styling	17
8.1 set	17
8.2 show	17
9 Scripting	19
9.1 Experssions	19
9.2 Blocks	20
9.3 Let bindings	21
9.4 Conditionals	21
9.5 Loops	22
9.6 Fields	23
9.7 Methods	24
9.8 Modules	24
9.9 Operators	25
10 Types	26
10.1 none	26

10.2 auto	27
10.3 boolean	27
10.4 integer	27
10.5 float	27
10.6 length	27
10.7 angle	28
10.8 ratio	28
10.9 relative length	28
10.10 fraction	29
10.11 color	29
10.12 symbol	31
10.13 string	32
10.14 content	34
10.15 array	34
10.16 dictionary	36
10.17 function	38
10.18 arguments	40
10.19 module	40
11 Text	41
11.1 lorem	41
11.2 emph	41
11.3 strong emphasis	42
11.4 linebreak	42
11.5 lowercase	43
11.6 uppercase	44

11.7 overline	44
11.8 underline	45
11.9 raw text/code	46
11.10 small capitals	48
11.11 smart quote	49
11.12 strikethrough	50
11.13 subscript	50
11.14 superscript	51
11.15 text	52
12 Math	59
13 Layout	59
13.1 align	59
13.2 block	60
13.3 box	61
13.4 table	62
13.5 list	64
13.6 colbreak	66
13.7 clolumns	67
13.8 enum	68
13.9 grid	68
13.10 h	68
13.11 v	69
13.12 hide	69
13.13 measure	70
13.14 move	71

13.15 padding	71
13.16 page	72
13.17 pagebreak	76
13.18 par	77
13.19 parbreak	78
13.20 place	79
13.21 repeat	79
13.22 rotate	80
13.23 stack	81
13.24 scale	81
13.25 terms	82
14 Visualize	84
14.1 line	84
14.2 rect	84
14.3 square	86
14.4 circle	87
14.5 ellipse	88
14.6 image	89
15 Meta	89
15.1 bibliography	89
15.2 cite	90
15.3 ref	91
15.4 figure	93
15.5 heading	93
15.6 numbering	94

15.7 link	95
15.8 locate	96
15.9 outline	97
15.10 counter	97
15.11 query	101
15.12 document	103
16 Calculate	103
17 Construct	103
18 Data Loading	103
18.1 csv	103
18.2 json	104
18.3 plain text	105
18.4 xml	105
19 Foundations	106
19.1 Assert	106
19.2 Evaluate	107
19.3 Panic	107
19.4 Representation	107
19.5 Type	108

1. Align

```
// 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
```

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

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.

2. Figure

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
```

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 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

```

@gege shows the right pose of playing basketball

```

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

```

Figure 1 shows the right pose of playing basketball



Figure 1: Playing basketball.

3. Box

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
```

Image with baseline 50%:

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

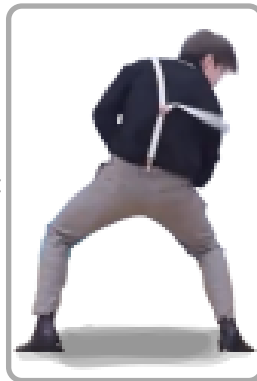


Image with baseline 50%:

4. Colbreak

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
```

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。

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.

5. Columns

`#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
```

`#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.

This research was interviews with a funded by the Na- grant of up to tional Academy of USD 40.000 for Sciences. NAoS a period of 6 provided support months.

for field tests and

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

6. Enum

`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。

```
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
```

```
#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]
]
```

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

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

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

a) Starting off ...

b) Don't forget step two

```
#block(fill:rgb("#b1f2eb"),
  width:100%,
  inset:5pt,
  radius:5pt)[
  #set enum(numbering: "1.a")
  + Different
  + Numbering
    + Nested
    + Items
  + Style

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

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

1) Different

2) Numbering

a) Nested

b) Items

3) Style

¹ Superscript

² Numbering!

c. Skipping

d. Ahead

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

c. Skipping

d. Ahead

```
#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

7. Grid

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.

```
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
```

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

Easy to learn	Great output	Intuitive
Our best Typst yet	Responsive design in print for everyone	One more thing...

7.1. 通过Figure、Grid结合绘制子图

```
#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.
  ]
)

```



(a)



(b)



(c) Test Caption

Figure 2: Test caption



Figure 3: SubFigures.

8. Styling

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

```
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]
```

This list is affected:

– Dash

This one is not:

• Bullet

• Food today?

• Work deadline

8.2. show

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：

- 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

```
#[
  #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

~ ***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.

9. Scripting

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. Experssions

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。

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

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. 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]`。

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

hello from the **world**

9.3. Let bindings

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。

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

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

This is Typst's documentation. It explains Typst.

Sum is 5.

9.4. Conditionals

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。

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

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. Loops

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。

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

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

A is a letter.

B is a letter.

C is a letter.

(3, 5, 9, 17)

- `for 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 {..}`

`for index, value in array {..}`

迭代数组中的值。还可以提供每个值的索引。

- `for value in dict {..}`

`for key, value in dict {..}`

迭代字典的键值对

- `for value in args {..}`

`for name, value in args {..}`

迭代args的键值

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。

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

  letter
}
```

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. Fields

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`

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

Hello



9.7. 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。

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

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

```

4

3

a — b — c

Methods是Typst中唯一可以修改调用值的函数。

9.8. Modules

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

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



9.9. Operators

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
!=	4
<	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

10. Types

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。

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

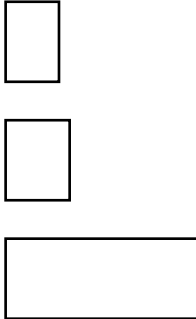
```
3.14  
10000  
2.5
```

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

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



10.7. angle

角度，支持如下单位：

- Degrees: 180deg
- Radians: 3.14rad

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

Hello there!

10.8. ratio

整体的比例。数字+百分号。

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

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。

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



10.10. fraction

定义布局中剩余空间的分布方式。

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

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

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

10.11.1.2. darken

使颜色变暗

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

10.11.1.3. negate

反色

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

10.11.2. sRGB

创建RGB色彩，颜色在sRGB空间中指定

```
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
```

```
#square(fill: rgb("#b1f2eb"))
#square(fill: rgb(87, 127, 230))
#square(fill: rgb(25%, 13%, 65%))
#text(16pt, rgb("#239dad"))[ *Typst* ]
```



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。

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

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



10.11.4. D65

创建灰度图

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

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



10.12. symbol

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。

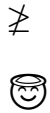
- 通用符号在sym module中定义
- Emoji在emoji module中定义

更进一步可以使用symbol函数自定义符号

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

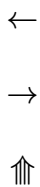
→

≧



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。

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



10.13. string

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。

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

hello world!

"hello

world"

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

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


`false``true`

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

10.13.1. Methods

```

// 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
```

10.14. content

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。

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

```
Type of Hello! is content
```

10.14.1. Methods

```
// 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
```

10.15. array

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 `()`

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

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

1

2

4

(3, 7, -3)

(3, 7, 4, 3, 2)

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

(1, 2, 3)

A, B and C

10.15.1. Methods

```
// 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
// 使用累加将所有项合并为一个值
value.fold(any,function,) -> 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
value.sorted() -> array

```

10.16. dictionary

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。

```
#let dict = (
  name: "Typst",
  born: 2019,
)

#dict.name \
#(dict.launch = 20)
#dict.len() \
#dict.keys() \
#dict.values() \
#dict.at("born") \
#dict.insert("city", "Berlin ")
#("name" in dict)
```

Typst

3

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

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

2019

```
true
```

10.16.1. Methods

```
// 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

```

10.17. function

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。

```

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

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

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

```

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

使用`#let`设置变量，通过`#`调用

```
#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?
```

Warning:
Danger is imminent!

Warning:
KEEP OFF TRACKS

once? once?

10.17.1. Methods

```
// 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
```

10.18. arguments

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)`。

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

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

ArtosFlow

Written by Jane and Joe.

10.18.1. Methods

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

10.19. module

导入模块

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

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


11. Text

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。

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

= **Blind Text**

```
#lorem(30)
```

= **More Blind Text**

```
#lorem(15)
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam 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. 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`

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

```
This is _emphasized_ \
```

```
This is #emph[too.]
```

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

```
}
```

This is *_emphasized_* differently.

This is *emphasized*.

This is *too*.

This is **emphasized** differently.

11.3. strong emphasis

字体加粗，语法糖: *

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

This is ***strong.*** \

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

#show strong: set text(red)

And this is ***evermore.***

#set strong(delta: 0)

No ***effect!***

This is **strong.**

This is **too.**

And this is **evermore.**

No **effect!**

11.4. 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: \

```
// 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
```

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

Date: 26.12.2022

Topic: Infrastructure Test

Severity: High

```
#set par(justify: true)
#let jb = linebreak(justify: true)

I have manually tuned the #jb
line breaks in this paragraph #jb
for an _interesting_ result. #jb
```

I	have	manually	tuned	the
line	breaks	in	this	paragraph
for	an	<i>interesting</i>		result.

11.5. lowercase

大小写转换

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

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

abc

my text

already low

11.6. uppercase

大小写转换

```
upper(string content) -> string content
```

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

ABC

MY TEXT

ALREADY HIGH

11.7. underline

文本上划线

```
underline(
  // 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
```

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

A line over text.

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

The Forest Theme

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

The Tale Of A Faraway Line II

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

Typography Today

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

Temple

11.8. underline

下划线用法与下划线一致

```
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

```

```

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].

```

```

Take care

  The Tale Of A Faraway Line I

                                Chapter 1

This is great. This is less great.

```

11.9. 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。

```

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
```

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

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

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

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

```
// 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
\\`\\`
```

```
// 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"
\\\`
```

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。

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



```
#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. smart quote

根据活动文本语言使用适当的引用符号。语法糖: '和"

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

```
"This is in quotes."

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

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

```
“This is in quotes.”
„Das ist in Anführungszeichen.“
« C'est entre guillemets. »
```

```
#set smartquote(enabled: false)

These are "dumb" quotes.
```

These are "dumb" quotes.

11.12. strikethrough

删除线

```
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
```

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

This is ~~not~~ relevant.

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.

This is ~~very stricken through~~.

This is ~~██████~~.

This is ~~low-ish~~.

This is ~~on-top~~.

This ~~skips~~ parts of the word.

This ~~extends~~ beyond the word.

11.13. subscript

设置下标

```
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
```

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

Revenue_{yearly}

N₁

N₁

11.14. superscript

设置上标，与下标相同

```
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
```

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

1st try!

N¹

N¹

11.15. text

以多种方式自定义文本的外观和布局。

```
text(
  // 字体系的优先顺序。
  // 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

```

```

#set text(18pt)
With a set rule.

#emph(text(blue)[
  With a function call.
])

```

With a set rule.

With a function call.

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

This is Latin. \

هذا عربي.

This is Latin.

هذا عربي.

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

هذا عربي

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

هذا عربي

هذا عربي

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

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

Italic Oblique

```
#text(weight: "light")[Light] \
#text(weight: "regular")[Regular] \
#text(weight: "medium")[Medium] \
#text(weight: 500)[Medium] \
#text(weight: "bold")[Bold]
```

Light

Regular

Medium

Medium

Bold

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

Condensed

Normal

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

very **big** text

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

Distant text.

```
#set text(spacing: 200%)  
Text with distant words.
```

A **word** 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.

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.

```
#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]
```

Typst

Typst

Typst

Typst

```
#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.

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

```
#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
```

Number 9.

Number 9.

A 12 B 34.

A 56 B 78.

A 12 B 34.

A 56 B 78.

0, 0

$\frac{1}{2}$

$\frac{1}{2}$ $\frac{1}{2}$

12. Math

13. Layout

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. align

水平或者垂直对齐内容

```
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
```

```
#set align(center)
```

Centered text, a sight to see \

In perfect balance, visually \

Not left nor right, it stands alone \

A work of art, a visual throne

Centered text, a sight to see

In perfect balance, visually

Not left nor right, it stands alone

A work of art, a visual throne

13.2. 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

```
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 `none` content,      // block内容
  ) -> content

```

```

#align(center)[
  #block(
    width:50%,
    height: 15em,
    breakable: true,
    fill: gray,
    stroke: black + 2pt,
    radius:5pt,
    inset:15pt,
    outset:1pt,
    spacing: 50%,
    lorem(20),
  )
]

```

Lorem ipsum dolor sit amet, con-
 sectetur adipiscing elit, sed do
 eiusmod tempor incididunt ut la-
 bore et dolore magnam aliquam
 quaerat.

13.3. box

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

```

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].

```

Line in _____ between.

An inline `rectangle` .

13.4. 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。



```

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
```

```
#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]
)
```

	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

```
#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 €],
)

```

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

```

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

```

Hello	Hello	Hello
A	B	C

13.5. 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。

```

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

```

```

- *Content*
  - Text
  - Math
  - Layout
  - Visualize
  - Meta
  - Symbols

- *Compute*
  #list(
    [Foundations],
    [Calculate],
    [Construct],
    [Data Loading],
  )

```

- **Content**
 - Text
 - Math
 - Layout
 - Visualize
 - Meta
 - Symbols
- **Compute**
 - Foundations
 - Calculate
 - Construct

- Data Loading

```
#set list(marker: [--])
- A more classic list
- With en-dashes

#set list(marker: ([.], [--]))
- Top-level
  - Nested
  - Items
- Items

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

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

13.6. 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。

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

```
#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. 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。

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

```
#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.

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

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. 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。

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

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

First	Second		
First		Second	
First		Second	Third

13.11. 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。

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

```
#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],
)
```

A		A		A		A	B		A		A
	B		B								
				B							
									B		
											B

13.12. 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。

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

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

```
Hello Jane

Joe
```

13.13. 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。

```
// 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
```

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

```
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。

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

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

Width of “Hey” is 19.99pt

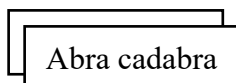
Width of “Welcome” is 45.01pt

13.14. 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。

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

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



13.15. 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。

```
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
```

```
#set align(center)
```

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



Typing speeds can be measured in words per minute.

13.16. page

设置页面格式

```
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

```

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

There you go, US friends

```
#set page(
width: 3cm,
margin: (x: 0cm),
```

```
)

#for i in range(3) {
  box(square(width: 1cm))
}
```

```
#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
```

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

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

```
#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.

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

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

#lorem(48)
```

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

#lorem(30)
```

```
#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)
```

```
#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)
```

```
#set page(background: rotate(24deg,
  text(18pt, fill: rgb("FFCBC4"))[
    *CONFIDENTIAL*
  ]
))
```

= Typst's secret plans

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

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

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

13.17. pagebreak

手动分页符

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

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

== Compound Theory

In 1984, the first ...

13.18. 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。

```
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
```

```
#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 ...

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 ...

```
#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.

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. parbreak

段落终断符

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

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

Blind text 0: Lorem ipsum dolor sit amet.

Blind text 1: Lorem ipsum dolor sit amet.

Blind text 2: Lorem ipsum dolor sit amet.

13.20. 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。

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

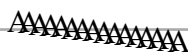
Hello, world!

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

Hello, world!



```
#for i in range(16) {
  let amount = i * 4pt
  place(center, dx: amount - 32pt, dy: amount / 8)[A]
}
```



13.21. repeat

重复内容。这在实现自定义索引、引用或大纲时很有用。

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

Sign on the dotted line:

```
#box(width: 1fr, repeat[.])

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

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

13.22. rotate

按给定角度旋转元素。

```
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
```

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

X ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂ ✂

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

Space!


```
#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()))
```

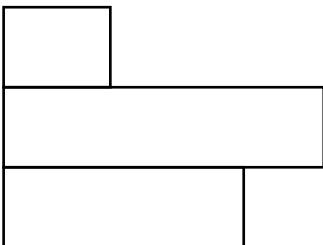


13.23. stack

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。

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

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



13.24. 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。

```
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
```

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

.bɜrɒrɪm ɪz ɪɪdɪ

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

AAA
BB B

13.25. 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。

```
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
```

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

Ligature A merged glyph.

Kerning A spacing adjustment between two adjacent letters.

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

/ **Colon**: A nice separator symbol.

Colon: A nice separator symbol.

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

/ **Colon**: A nice separator symbol.

Colon A nice separator symbol.

```
#set terms(hanging-indent: 0pt)
```

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

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

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

TeX Born in 1978.

LaTeX Born in 1984.

Typst Born in 2019.

14. Visualize

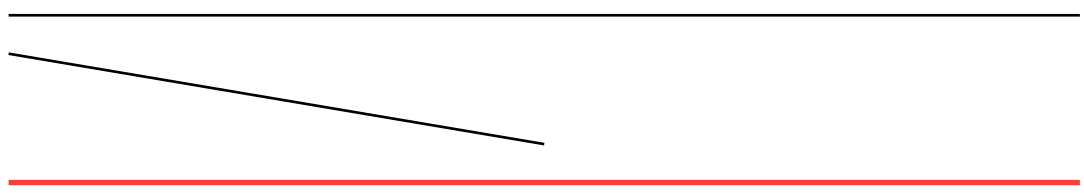
目前Typst还不支持绘制图表!

14.1. line

绘制直线

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

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



14.2. rect

绘制矩形

```
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
```

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

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



Automatically sized
to fit the content.

```
#rect(fill: blue)
#stack(
  dir: ltr,
  spacing: 1fr,
  rect(stroke: red),
  rect(stroke: 2pt),
```

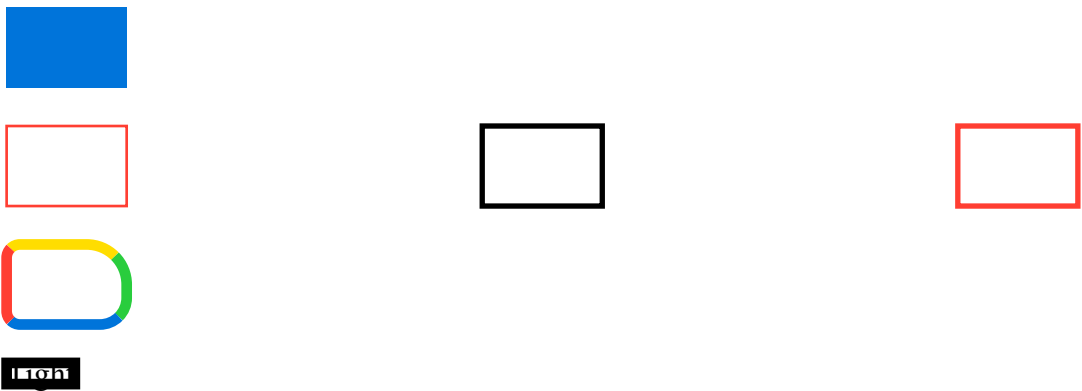
```

    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]

```



14.3. square

正方形，参数解释同上

```

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

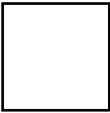
```

```

// Without content.
#square(size: 40pt)

// With content.
#square[
  Automatically \
  sized to fit.
]

```



```

Automatically
sized to fit.

```

14.4. circle

绘制带有内容的圆圈

```

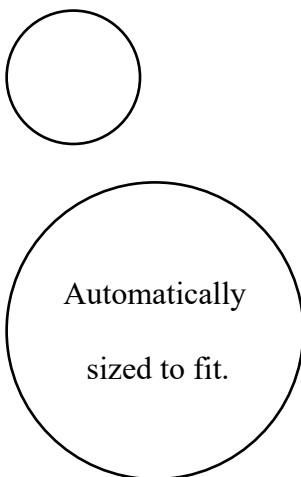
circle(
  // 圆半径，与weight和height互斥
  set radius: length,
  // 相对于父容器的宽度，互斥
  set width: auto relative length,
  // 相对于父容器的高度，互斥
  set height: auto relative length,
  // 内部填充
  set fill: none color,
  // 圆圈格式
  set stroke: none auto length color stroke,

```

```
// 内边距
set inset: relative length dictionary,
// 外边距
set outset: relative length dictionary,
set none content,
) -> content
```

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

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



14.5. ellipse

绘制椭圆，参数同上

```
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

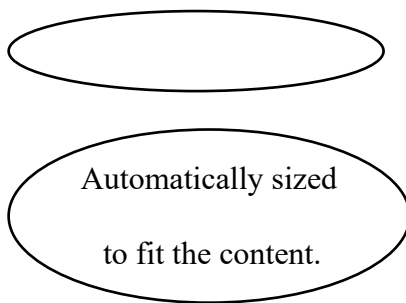
```

```

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

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

```



14.6. image

参考之前章节 Section 2 的描述

15. 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. 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。

```
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
  set style: string,
) -> content
```

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

Multiple sources say ...

`#cite("arrgh", "netwok").`

`#bibliography("works.bib")`

15.2. 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。

```
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 author-date author-title keys
set style: auto string,
) -> content
```

This was already noted by
pirates long ago. @arrgh

Multiple sources say ...

```
#cite("arrgh", "netwok").
```

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

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

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

```
#set cite(brackets: false)
```

@netwok follow these methods
in their work ...

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

```
#set cite(style: "alphanumerical")
```

Alphanumerical references.

@netwok

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

15.3. 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。

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

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

```
#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. figure

见之前章节 Section 2 描述

15.5. 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。

```
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
```

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

```
= Introduction
```

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

```
== Preliminaries
```

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

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

```
= A section
```

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

```
#outline()

#heading[Normal]
This is a normal heading.

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

15.6. 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。

```
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
```

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

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

```
1.2.3)
```

```
1.b
```

```
XII – 2
```

```
1.2.3)
```

15.7. 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。

```
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
```

```
#show link: underline
```

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

<https://example.com>

<https://example.com>

[See example.com](#)

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

hello@typst.app

[Go to top](#)

15.8. 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。

locate(function) -> content

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

My location:

(page: 96, x: 108.2pt, y: 548.95pt) !

15.8.1. methods

```
// 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
```

15.9. 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。

```
outline(
  // 设置目录标题，none表示无目录标题
  set title: none auto content,
  // 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
```

```
#outline()
```

```
= Introduction
```

```
#lorem(5)
```

```
= Prior work
```

```
#lorem(10)
```

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

```
#outline(indent: true)
```

```
= About ACME Corp.
```

```
== History
```

```
#lorem(10)
```

```
== Products
```

```
#lorem(10)
```

15.10. 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。

15.10.1. 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。

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

```
= Introduction
```

```
Some text here.
```

```
= Background
```

```
The current value is:
```

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

```
Or in roman numerals:
```

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

15.10.2. 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。

```
#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().
```

15.10.3. 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。

```
#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.
```

15.10.4. 自定义counters

自定义counter，使用字符串作为key调用counter函数。

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

1
2
6

15.10.5. time travel

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

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

= Values
#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 `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,)。

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

15.10.6. Methods

```
// 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
```

15.11. 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。

15.11.1. 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。

```
#set page(header: locate(loc => {
  let elems = query(
    heading,
    before: 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)

```
query(
  // heading figure equation reference label
```

```

    target: label function,
    location,
    before: location,
    after: location,
  ) -> content

```

15.12. 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。

```

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

```

16. Calculate

17. Construct

18. Data Loading

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。

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

```
#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。

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

```
#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

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

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

An example for a HTML file:\

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

18.4. 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。

```
xml(path: string) -> array
```

```
#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)
  }
}

```

19. 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。

```

assert(
  // 判断语句
  condition: boolean,
  // 报错信息

```

```
message: nonestring,
) ->
```

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

19.2. Evaluate

将字符串评估为 Typst 代码。

```
eval(
  source: string
) -> any
```

```
#eval("1 + 1") \
#eval("(1, 2, 3, 4).len() \
#eval("[*Strong text*]")
```

2

4

Strong text

19.3. Panic

报错

```
panic(
  ..any
) ->
```

```
#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。

```
repr(any) -> string
```

```
#none vs #repr(none) \
#"hello" vs #repr("hello") \
#(1, 2) vs #repr((1, 2)) \
#[*Hi*] vs #repr([*Hi*])
```

vs none

hello vs "hello"

(1, 2) vs (1, 2)

Hi vs strong(body: [Hi])

19.5. Type

返回值类型

```
type(any) -> string
```

```
#type(12) \
#type(14.7) \
#type("hello") \
#type(none) \
#type([Hi]) \
#type(x => x + 1)
```

integer

float

string

none

content

function