目录

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. 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(30),
    )
]
```

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

3. 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.

4. 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%:

5. 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.

6. 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 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.

7. 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

8. 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…

8.1. 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.

9. 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.

9.1. 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

9.2. 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fileds的表术，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)")
    #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.
]
```

9.2.1. This is navy-blue

But this stays black

~ *Dragon* (X) ~

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

~ *Manticore* (XI) ~

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.

10. 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.

10.1. 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！！

10.2. 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**

10.3. 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.

10.4. 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 [..]

10.5. 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次迭代。

```
#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 [..]

10.6. 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

😀

10.7. 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。

10.8. 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
```

10.9. 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
<=	4

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

11. 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。

11.1. 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。

11.2. auto

自动识别类型

11.3. boolean

布尔值， true or false

11.4. 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。

11.5. 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

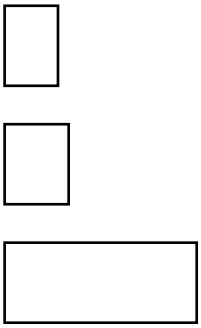
2.5

11.6. 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)
```



11.7. angle

角度，支持如下单位：

- Degrees: 180deg
- Radians: 3.14rad

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

Hello there!

11.8. ratio

整体的比例。数字+百分号。

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

Scaled apart.

11.9. 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)
```

11.10. 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

11.11. 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.

11.11.1. Methods

11.11.1.1. lighten

增亮颜色

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

11.11.1.2. darken

使颜色变暗

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

11.11.1.3. negate

反色

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

11.11.2. 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* ]
```

11.11.3. 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%)
)
```


11.11.4. D65

创建灰度图

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

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

11.12. 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
```

→

≱

$\not\geq$

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。

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

$\leftarrow$

$\rightarrow$

$\Uparrow$

11.13. 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

11.13.1. 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
```

11.14. 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

11.14.1. 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
```

11.15. 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
```

11.15.1. 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
```

11.16. 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
```

11.16.1. 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
```

11.17. 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?

11.17.1. 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
```

11.18. 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.

11.18.1. Methods

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

11.19. module

导入模块

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

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