array

一系列值。

您可以通过将逗号分隔的值序列括在括号中来构造数组。这些值不必是同一类型。

您可以使用该方法访问和更新数组项.at()。索引是从零开始的，负索引环绕到数组的末尾。您可以使用for 循环迭代数组。数组可以使用运算符进行相加+、连接以及与整数相乘。

注意：长度为 1 的数组需要尾随逗号，如 中。这是为了消除简单的括号表达式（如 ）的歧义。空数组写为.(1,)(1 + 2) * 3()

例子

```
#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
```

构造函数

将值转换为数组。

请注意，此函数仅用于将类似集合的值转换为数组，而不是用于从单个项目创建数组。请改用数组语法(1, 2, 3)（或(1,)单元素数组）。

```
大批（字节 大批 版本）->大批
```

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

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

value 字节 或者 大批 或者 版本 *必需的* *位置性*

应转换为数组的值。

定义

len

数组中值的数量。

```
自己。伦（）->整数
```

first

返回数组中的第一项。可用于作业的左侧。如果数组为空，则会失败并出现错误。

```
自己。第一的（）->任何
```

last

返回数组中的最后一项。可用于作业的左侧。如果数组为空，则会失败并出现错误。

```
自己。最后的（）->任何
```

at

返回数组中指定索引处的项目。可用于作业的左侧。如果索引超出范围，则返回默认值；如果未指定默认值，则失败并出现错误。

```
自己。在（
  整数，
  默认：任何，
）->任何
```

index 整数 *必需的* *位置性*

检索项目的索引。如果为负数，则从后面开始索引。

default 任何

索引越界时返回的默认值。

push

将值添加到数组末尾。

```
自己。推（任何）
```

value 任何 *必需的* *位置性*

要插入到数组末尾的值。

pop

从数组中删除最后一项并将其返回。如果数组为空，则会失败并出现错误。

```
自己。流行音乐（）->任何
```

insert

将值插入到数组的指定索引处。如果索引越界，则会失败并出现错误。

```
自己。插入（
  整数，
  任何，
）
```

index 整数 *必需的* *位置性*

要插入项目的索引。如果为负数，则从后面开始索引。

value 任何 *必需的* *位置性*

要插入到数组中的值。

remove

从数组中删除指定索引处的值并返回它。

```
自己。消除（
  整数，
  默认： 任何，
）-> 任何
```

index 整数 *必需的* *位置性*

要删除项目的索引。如果为负数，则从后面开始索引。

default 任何

索引越界时返回的默认值。

slice

提取数组的子切片。如果起始或索引超出范围，则会失败并出现错误。

```
自己。切片（
  整数，
  没有任何 整数，
  数数： 整数，
）-> 大批
```

start 整数 *必需的* *位置性*

起始索引（含）。如果为负数，则从后面开始索引。

end 没有任何 或者 整数 *位置性*

结束索引（不包括）。如果省略，则提取整个切片直到数组末尾。如果为负数，则从后面开始索引。

默认：`none`

count 整数

要提取的项目数。这相当于传递`start + count`位置`end`。与 互斥`end`。

contains

数组是否包含指定值。

此方法还有专用语法：您可以编写而不是`.2 in (1, 2, 3)`。`(1, 2, 3).contains(2)`

```
自己。包含（ 任何 ）-> 布尔值
```

value 任何 *必需的* *位置性*

要搜索的值。

find

搜索给定函数返回的项目`true`并返回第一个匹配项，或者`none`如果没有匹配项则返回。

```
自己。寻找（ 功能 ）-> 任何 没有任何
```

searcher 功能 *必需的* *位置性*

应用于每个项目的函数。必须返回一个布尔值。

position

搜索给定函数返回的项目`true`，并返回第一个匹配项的索引，或者`none`如果没有匹配项则返回。

```
自己。位置（ 功能 ）->没有任何 整数
```

searcher 功能 *必需的* *位置性*

应用于每个项目的函数。必须返回一个布尔值。

range

创建一个由数字序列组成的数组。

如果仅传递一个位置参数，它将被解释为`end`范围的 。如果您通过了两个，它们将描述范围的`start`和。`end`

该函数在数组函数的作用域和全局范围内都可用。

```
大批。范围（
  整数，
  整数，
  步：整数，
）->大批
```

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

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

start 整数 *位置性*

范围的开始（含）。

默认：`0`

end 整数 *必需的* *位置性*

范围的末尾（不包括）。

step 整数

生成的数字之间的距离。

默认：1

filter

生成一个新数组，其中仅包含原始数组中给定函数返回 true 的项目。

```
自己。筛选（功能）->大批
```

test 功能 *必需的* *位置性*

应用于每个项目的函数。必须返回一个布尔值。

map

生成一个新数组，其中原始数组中的所有项目均已使用给定函数进行转换。

```
自己。地图（功能）->大批
```

mapper 功能 *必需的* *位置性*

应用于每个项目的函数。

enumerate

返回一个新数组，其中包含值及其索引。

返回的数组由`(index, value)`长度为 2 的数组形式的对组成。这些可以通过 let 绑定或 for 循环进行解构。

```
自己。枚举（开始：整数）->大批
```

start 整数

返回列表中第一对的索引。

默认：`0`

zip

将数组与其他数组一起压缩。

返回一个数组数组，其中`i`第 th 个内部数组包含`i`每个原始数组中的所有第 th 个元素。

如果要压缩的数组具有不同的长度，则会将它们压缩到最短数组的最后一个元素，并忽略所有剩余元素。

这个函数是可变的，这意味着您可以一次将多个数组压缩在一起：yields 。

```
(1, 2).zip(("A", "B"), (10, 20))((1, "A", 10), (2, "B", 20))
```

```
自己。压缩（ .. 大批 ）-> 大批
```

others `大批` *必需的* *位置性* *可变参数*

要压缩的数组。

fold

使用累加器函数将所有项目折叠为单个值。

```
自己。折叠（
  任何，
  功能，
）-> 任何
```

init `任何` *必需的* *位置性*

开始时的初始值。

folder `功能` *必需的* *位置性*

折叠功能。必须有两个参数：一个用于累计值，一个用于项目。

sum

对所有项目求和（适用于所有可以添加的类型）。

```
自己。总和（默认： 任何 ）-> 任何
```

default 任何

如果数组为空，则返回什么。如果数组可以为空，则必须设置。

product

计算所有项目的乘积（适用于所有可以相乘的类型）。

```
自己。产品（默认： 任何 ）-> 任何
```

default 任何

如果数组为空，则返回什么。如果数组可以为空，则必须设置。

any

给定函数是否返回true数组中的任何项目。

```
自己。任何（ 功能 ）-> 布尔值
```

test 功能 *必需的* *位置性*

应用于每个项目的函数。必须返回一个布尔值。

all

给定函数是否返回true数组中的所有项目。

```
自己。全部（功能）->布尔值
```

test 功能 *必需的* *位置性*

应用于每个项目的函数。必须返回一个布尔值。

flatten

将所有嵌套数组合并为一个平面数组。

```
自己。展平（）->大批
```

rev

返回一个包含相同项目但顺序相反的新数组。

```
自己。转速（）->大批
```

split

在出现指定值时拆分数组。

```
自己。分裂（任何）->大批
```

at 任何 *必需的* *位置性*

要分割的值。

join

将数组中的所有项目合并为一个。

```
自己。加入（
```

```
  任何 没有任何,
  最后的: 任何,
) -> 任何
```

separator 任何 或者 没有任何 *位置性*

要在数组的每个项目之间插入的值。

默认：none

last 任何

最后两项之间的替代分隔符。

intersperse

返回一个数组，其中相邻元素之间放置有分隔符值的副本。

```
自己。散布（ 任何 ）-> 大批
```

separator 任何 *必需的* *位置性*

将放置在每个相邻元素之间的值。

chunks

将数组拆分为不重叠的块，从开头开始，以单个剩余块结束。

除了最后一个块之外的所有块都有chunk-size元素。如果exact设置为true，则如果余数小于chunk-size元素，则将其删除。

```
自己。块（
  整数,
  精确的: 布尔值,
) -> 大批
```

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

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

chunk-size 整数 *必需的* *位置性*

每个块最多可以包含多少个元素。

exact 布尔值

如果余数小于，是否保留余数chunk-size。

默认：`false`

sorted

返回此数组的排序版本，可以选择通过给定的键函数。使用的排序算法是稳定的。

如果无法比较两个值或者关键函数（如果给定）产生错误，则返回错误。

```
自己。排序（钥匙： 功能 ）->大批
```

key 功能

如果给定，则将此函数应用于数组中的元素以确定排序的键。

dedup

对数组中的所有项目进行重复数据删除。

返回删除所有重复项的新数组。仅保留每个副本的第一个元素。

```
自己。去重（钥匙： 功能 ）->大批
```

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

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

key 功能

如果给定，则将此函数应用于数组中的元素以确定要进行重复删除的键。