

# 计算

用于计算和处理数值的模块。

这些定义是模块的一部分`calc`，默认情况下不导入。除了下面列出的函数之外，该`calc`模块还定义了常量`pi`、`tau`、`e`、`inf`和`nan`。

## 功能

### `abs`

计算数值的绝对值。

计算。腹肌（`整数` `漂浮` `长度` `角度` `比率` `分数`）-> `任何`

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

```
5
51.69pt
2fr
```

**value** `整数` `或者` `漂浮` `或者` `长度` `或者` `角度` `或者` `比率` `或者` `分数` *必需的* *位置性* 

要计算其绝对值的值。

### `pow`

将值提高到某个指数。

计算。战俘（  
    `整数` `漂浮`，  
    `整数` `漂浮`，  
）-> `整数` `漂浮`

```
#calc.pow(2, 3)
```

```
8
```

**base** 整数 或者 漂浮 必需的 位置性 ?

权力的基础。

**exponent** 整数 或者 漂浮 必需的 位置性 ?

幂的指数。

## exp

将值提高到 e 的某个指数。

计算。经验 ( 整数 漂浮 ) -> 漂浮

```
#calc.exp(1)
```

2.718281828459045

**exponent** 整数 或者 漂浮 必需的 位置性 ?

幂的指数。

## sqrt

计算数字的平方根。

计算。开方 ( 整数 漂浮 ) -> 漂浮

```
#calc.sqrt(16) \  
#calc.sqrt(2.5)
```

4  
1.5811388300841898

**value** 整数 或者 漂浮 必需的 位置性 ?

要计算其平方根的数字。必须是非负数。

## root

计算数字的实数  $n$  次方根。

如果该数为负数，则  $n$  必须为奇数。

计算。根（

  漂浮，

  整数，

） -> 漂浮

```
#calc.root(16.0, 4) \  
#calc.root(27.0, 3)
```

```
2  
3
```

**radicand**    漂浮    必需的    位置性 ?

求根的表达式

**index**    整数    必需的    位置性 ?

取被数的哪个根

## sin

计算角度的正弦值。

当使用整数或浮点数调用时，它们将被解释为弧度。

计算。罪恶（ 整数 漂浮 角度 ） -> 漂浮

```
#assert(calc.sin(90deg) == calc.sin(-270deg))  
#calc.sin(1.5) \  
#calc.sin(90deg)
```

```
0.9974949866040544
```

```
1
```

**angle**    整数 或者 漂浮 或者 角度    必需的    位置性 ②

要计算其正弦的角度。

## cos

计算角度的余弦。

当使用整数或浮点数调用时，它们将被解释为弧度。

计算。余弦( 整数 漂浮 角度 ) -> 漂浮

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

```
0.00000000000000006123233995736766
0.0707372016677029
0.00000000000000006123233995736766
```

**angle**    整数 或者 漂浮 或者 角度    必需的    位置性 ②

要计算余弦的角度。

## tan

计算角度的正切。

当使用整数或浮点数调用时，它们将被解释为弧度。

计算。晒黑( 整数 漂浮 角度 ) -> 漂浮

```
#calc.tan(1.5) \
#calc.tan(90deg)
```

```
14.101419947171719
16331239353195370
```

**angle**    整数 或者 漂浮 或者 角度    必需的    位置性 ②

要计算其正切的角度。

## asin

计算数字的反正弦。

计算。阿辛（ 整数 漂浮 ） -> 角度

```
#calc.asin(0) \  
#calc.asin(1)
```

0deg  
90deg

value 整数 或者 漂浮 必需的 位置性 ?

要计算其反正弦的数字。必须介于 -1 和 1 之间。

## acos

计算数字的反余弦。

计算。阿科斯（ 整数 漂浮 ） -> 角度

```
#calc.acos(0) \  
#calc.acos(1)
```

90deg  
0deg

value 整数 或者 漂浮 必需的 位置性 ?

要计算其反正弦的数字。必须介于 -1 和 1 之间。

## atan

计算数字的反正切。

计算。晒黑（ 整数 漂浮 ） -> 角度

```
#calc.atan(0) \
#calc.atan(1)
```

```
0deg
45deg
```

**value** 整数 或者 漂浮 必需的 位置性 ❸

要计算其反正切的数字。

## atan2

计算坐标的四象限反正切。

论据是(x, y)，不是(y, x)。

计算。阿坦2 (   
 整数 漂浮 ,   
 整数 漂浮 ,   
 ) -> 角度

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

```
45deg
-123.69deg
```

**x** 整数 或者 漂浮 必需的 位置性 ❸

X 坐标。

**y** 整数 或者 漂浮 必需的 位置性 ❸

Y 坐标。

## sinh

计算双曲角的双曲正弦。

计算。辛赫 ( 漂浮 ) -> 漂浮

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

```
0  
2.1292794550948173
```

**value** 漂浮 必需的 位置性 [?](#)

要计算其双曲正弦的双曲角。

## cosh

计算双曲角的双曲余弦。

计算。科什 ( 漂浮 ) -> 漂浮

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

```
1  
2.352409615243247
```

**value** 漂浮 必需的 位置性 [?](#)

要计算其双曲余弦的双曲角。

## tanh

计算双曲角的双曲正切。

计算。tanh ( 漂浮 ) -> 漂浮

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

```
0  
0.9051482536448664
```

**value** 漂浮 必需的 位置性 [?](#)

要计算其双曲正切的双曲角。

## log

计算数字的对数。

如果未指定底数，则以 10 为底计算对数。

计算。日志（  
 整数 漂浮，  
 根据： 漂浮，  
） -> 漂浮

```
#calc.log(100)
```

2

**value** 整数 或者 漂浮 必需的 位置性 ?

要计算其对数的数字。必须严格积极。

**base** 漂浮

对数的底。可能不为零。

默认：10.0

## ln

计算数字的自然对数。

计算。ln（ 整数 漂浮 ） -> 漂浮

```
#calc.ln(calc.e)
```

1

**value** 整数 或者 漂浮 必需的 位置性 ?

要计算其对数的数字。必须严格积极。



## fact

计算数字的阶乘。

计算。事实 ( 整数 ) -> 整数

```
#calc.fact(5)
```

120

**number** 整数 必需的 位置性 ❶

要计算其阶乘的数字。必须是非负数。

## perm

计算排列。

返回k的排列，或从一组中按顺序n选择项目的方法数。kn

计算。烫发 (   
 整数 ,   
 整数 ,   
 ) -> 整数

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

$$\text{perm}(n, k) = \frac{n!}{(n - k)!}$$
$$\text{perm}(5, 3) = 60$$

**base** 整数 必需的 位置性 ❶

基数。必须是非负数。

**numbers** 整数 必需的 位置性 ❶

排列的数量。必须是非负数。

## binom

计算二项式系数。

返回k的组合，或从一组中n选择项目的方法数，而不考虑顺序。kn

计算。比诺姆(

整数，

整数，

) -> 整数

```
#calc.binom(10, 5)
```

252

n 整数 必需的 位置性 ②

上系数。必须是非负数。

k 整数 必需的 位置性 ②

系数越低。必须是非负数。

## gcd

计算两个整数的最大公约数。

计算。最大公约数 (

整数，

整数，

) -> 整数

```
#calc.gcd(7, 42)
```

7

a 整数 必需的 位置性 ②

第一个整数。

**b** 整数 必需的 位置性 <sup>?</sup>

第二个整数。

## lcm

计算两个整数的最小公倍数。

计算。液晶厘米（

整数，

整数，

）-> 整数

```
#calc.lcm(96, 13)
```

1248

**a** 整数 必需的 位置性 <sup>?</sup>

第一个整数。

**b** 整数 必需的 位置性 <sup>?</sup>

第二个整数。

## floor

将数字向下舍入到最接近的整数。

如果数字已经是整数，则原样返回。

计算。地面（整数 漂浮）-> 整数

```
#assert(calc.floor(3.14) == 3)
#assert(calc.floor(3) == 3)
#calc.floor(500.1)
```

500

**value** 整数 或者 漂浮 必需的 位置性 

要向下舍入的数字。

## ceil


将数字向上舍入到最接近的整数。

如果数字已经是整数，则原样返回。

计算。天花板（整数 漂浮）-> 整数

```
#assert(calc.ceil(3.14) == 4)
#assert(calc.ceil(3) == 3)
#calc.ceil(500.1)
```

501

**value** 整数 或者 漂浮 必需的 位置性 

要向上舍入的数字。

## trunc

返回数字的整数部分。

如果数字已经是整数，则原样返回。

计算。截断（整数 漂浮）-> 整数

```
#assert(calc.trunc(3) == 3)
#assert(calc.trunc(-3.7) == -3)
#calc.trunc(15.9)
```

15

**value** 整数 或者 漂浮 必需的 位置性 

要截断的数字。

## fract

返回数字的小数部分。

如果数字是整数，则返回0。

计算。分形（`整数` `漂浮`）-> `整数` `漂浮`

```
#assert(calc.fract(3) == 0)
#calc.fract(-3.1)
```

-0.10000000000000009

**value** `整数` 或者 `漂浮` 必需的 位置性 ?

要截断的数字。

## round

将数字四舍五入到最近的整数。

或者，可以指定小数位数。

计算。圆形的（  
    `整数` `漂浮`，  
    数字： `整数`，  
）-> `整数` `漂浮`

```
#assert(calc.round(3.14) == 3)
#assert(calc.round(3.5) == 4)
#calc.round(3.1415, digits: 2)
```

3.14

**value** `整数` 或者 `漂浮` 必需的 位置性 ?

要舍入的数字。

**digits** `整数`

小数位数。

默认：0

## clamp

将数字限制在最小值和最大值之间。

计算。夹钳（  
整数 漂浮，  
整数 漂浮，  
整数 漂浮，  
） -> 整数 漂浮

```
#assert(calc.clamp(5, 0, 10) == 5)
#assert(calc.clamp(5, 6, 10) == 6)
#calc.clamp(5, 0, 4)
```

4

**value** 整数 或者 漂浮 必需的 位置性 ❓

要夹紧的数量。

**min** 整数 或者 漂浮 必需的 位置性 ❓

包含的最小值。

**max** 整数 或者 漂浮 必需的 位置性 ❓

包含的最大值。

## min

确定值序列的最小值。

计算。分钟（... 任何） -> 任何

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

-5  
beta

**values** 任何 必需的 位置性 ❓ 可变参数 ❓

从中提取最小值的值序列。不得为空。

## max

确定值序列的最大值。

计算。最大限度（... 任何）-> 任何

```
#calc.max(1, -3, -5, 20, 3, 6) \  
#calc.max("typst", "in", "beta")
```

```
20  
typst
```

**values**    任何    必需的    位置性 ⓘ    可变参数 ⓘ

从中提取最大值的值序列。不得为空。

## even

确定整数是否为偶数。

计算。甚至（ 整数 ）-> 布尔值

```
#calc.even(4) \  
#calc.even(5) \  
#range(10).filter(calc.even)
```

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

**value**    整数    必需的    位置性 ⓘ

检查均匀性的数量。

## odd

确定整数是否为奇数。

计算。奇怪的（ 整数 ）-> 布尔值

```
#calc.odd(4) \
#calc.odd(5) \
#range(10).filter(calc.odd)
```

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

**value** 整数 必需的 位置性 [?](#)

用于检查奇数的数字。

## rem

计算两个数的余数。

该值`calc.rem(x, y)`始终具有与 `x` 相同的符号，并且幅度小于`y`。

计算。雷姆（  
  整数 漂浮，  
  整数 漂浮，  
  ）-> 整数 漂浮

```
#calc.rem(7, 3) \
#calc.rem(7, -3) \
#calc.rem(-7, 3) \
#calc.rem(-7, -3) \
#calc.rem(1.75, 0.5)
```

```
1
1
-1
-1
0.25
```

**dividend** 整数 或者 漂浮 必需的 位置性 [?](#)

剩余部分的股息。

**divisor** 整数 或者 漂浮 必需的 位置性 [?](#)

余数的除数。

## div-euclid

执行两个数字的欧几里德除法。



此计算的结果是舍入为整数的除法结果，n使得被除数大于或等于n除数的倍。

计算。**div**-欧几里德(

整数 漂浮 ,  
整数 漂浮 ,  
) -> 整数 漂浮

```
#calc.div-euclid(7, 3) \  
#calc.div-euclid(7, -3) \  
#calc.div-euclid(-7, 3) \  
#calc.div-euclid(-7, -3) \  
#calc.div-euclid(1.75, 0.5)
```

```
2  
-2  
-3  
3  
3
```

**dividend** 整数 或者 漂浮 必需的 位置性 ?

分部的股息。

**divisor** 整数 或者 漂浮 必需的 位置性 ?

除法的除数。

## rem-euclid

这计算除法的最小非负余数。

警告：由于浮点舍入误差，如果被除数的大小远小于除数且被除数为负，则余数可能等于除数的绝对值。这仅适用于浮点输入。

计算。雷姆-欧几里德 (

整数 漂浮 ,  
整数 漂浮 ,  
) -> 整数 漂浮

```
#calc.rem-euclid(7, 3) \  
#calc.rem-euclid(7, -3) \  
#calc.rem-euclid(-7, 3) \  
#calc.rem-euclid(-7, -3) \  
#calc.rem(1.75, 0.5)
```

```
1  
1  
2  
2
```

**dividend** 整数 或者 漂浮 必需的 位置性 ?

剩余部分的股息。

**divisor** 整数 或者 漂浮 必需的 位置性 ?

余数的除数。

## quo

计算两个数字的商（取整除法）。

计算。现状（  
整数 漂浮 ,  
整数 漂浮 ,  
 ） -> 整数

```
$ "quo"(a, b) &= floor(a/b) \
"quo"(14, 5) &= #calc.quo(14, 5) \
"quo"(3.46, 0.5) &= #calc.quo(3.46, 0.5) $
```

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

$$\text{quo}(14, 5) = 2$$

$$\text{quo}(3.46, 0.5) = 6$$

**dividend** 整数 或者 漂浮 必需的 位置性 ?

商的被除数。

**divisor** 整数 或者 漂浮 必需的 位置性 ?

商的除数。

< 字节  
上一页

内容 >  
下一页