知乎

Python标准库教程——math模块

漫星野
一学物理的。

7 人赞同了该文章

写在前面

这是一篇通过例子学习Python标准库math的教程。math库提供了许多便捷的函数，能够计算常规数学运算、三角函数、双曲函数和部分特殊函数。

> 本文翻译自 Doug Hellmann 的 PyMOTW-3项目的math模块。原文链接。
>

math — 数学函数

目的：为特殊数学运算提供函数。

`math` 模块实现了许多IEEE使用浮点数进行复杂数学运算的函数，包括对数和三角函数运算，这些函数通常可以在本地C库找到。

特殊常量

许多数学运算基于特殊常量。 `math` 提供的值有， π (pi)，e， `nan` (不是数字)，和 ∞ 。

```
# math_constants.py

import math

print('  π: {:.30f}'.format(math.pi))
print('  e: {:.30f}'.format(math.e))
print('nan: {:.30f}'.format(math.nan))
print('inf: {:.30f}'.format(math.inf))
```

```
$ python3 math_constants.py

  π: 3.14159265358979311599796346854
  e: 2.71828182845904509079559829842
nan: nan
inf: inf
```

测试异常值

浮点数计算会产生两类异常值。第一类是 `inf`（ ∞ ），当用于保存浮点数的双精度值从绝对值较大的数溢出时出现。

```
# math_isinf.py

import math

print('{:^3} {:6} {:6} {:6}'.format(
    'e', 'x', 'x**2', 'isinf'))
print('{:-^3} {:-^6} {:-^6} {:-^6}'.format(
    '', '', '', ''))

for e in range(0, 201, 20):
    x = 10.0 ** e
    y = x * x
    print('{:3d} {:<6g} {:<6g} {!s:6}'.format(
        e, x, y, math.isinf(y),
    ))
```

当此例的指数增加到足够大时，x的平方不再包含于双精度值，并将该值记录为无穷。

```
$ python3 math_isinf.py

 e  x      x**2   isinf
--- ------ ------ ------
  0 1      1      False
 20 1e+20  1e+40  False
 40 1e+40  1e+80  False
 60 1e+60  1e+120 False
```

```
160 1e+160 inf    True
180 1e+180 inf    True
200 1e+200 inf    True
```

然而，并不是所有的浮点溢出都会导致 `inf` 值。特别地，使用浮点数计算指数会导致OverflowError，而不是保留 `inf` 结果。

```
# math_overflow.py

x = 10.0 ** 200

print('x    =', x)
print('x*x  =', x * x)
print('x**2 =', end=' ')
try:
    print(x ** 2)
except OverflowError as err:
    print(err)
```

这种差别是由C Python解释器使用的库的实现差异引起的。

```
$ python3 math_overflow.py

x    = 1e+200
x*x  = inf
x**2 = (34, 'Result too large')
```

使用无穷值的除法是未定义的。一个数除以无穷的结果是 `nan` （不是数字）。

```
# math_isnan.py

import math

x = (10.0 ** 200) * (10.0 ** 200)
y = x / x

print('x =', x)
print('isnan(x) =', math.isnan(x))
```

nan 不与任何值相等，包括它本身，所以需用 isnan() 检查 nan 。

```
$ python3 math_isnan.py

x = inf
isnan(x) = False
y = x / x = nan
y == nan = False
isnan(y) = True
```

使用 isfinite（） 检查常规数与特殊值 inf 或 nan 。

```
# math_isfinite.py

import math

for f in [0.0, 1.0, math.pi, math.e, math.inf, math.nan]:
    print('{:5.2f} {!s}'.format(f, math.isfinite(f)))
```

对于特殊值， isfinite() 返回 false，否则返回 true。

```
$ python3 math_isfinite.py

 0.00 True
 1.00 True
 3.14 True
 2.72 True
  inf False
  nan False
```

比较

比较浮点数会容易出错，这是因为，计算的每个步骤都可能由于数值表示而引入误差。
isclose() 函数使用稳健的算法来最小化这些错误，并提供相对和绝对比较的方法。所用的公式等价于

些值必须相差10%以内。

```
# math_isclose.py

import math

INPUTS = [
    (1000, 900, 0.1),
    (100, 90, 0.1),
    (10, 9, 0.1),
    (1, 0.9, 0.1),
    (0.1, 0.09, 0.1),
]

print('{:^8} {:^8} {:^8} {:^8} {:^8} {:^8}'.format(
    'a', 'b', 'rel_tol', 'abs(a-b)', 'tolerance', 'close')
)
print('{:-^8} {:-^8} {:-^8} {:-^8} {:-^8} {:-^8}'.format(
    '-', '-', '-', '-', '-', '-'),
)

fmt = '{:8.2f} {:8.2f} {:8.2f} {:8.2f} {:8.2f} {!s:>8}'

for a, b, rel_tol in INPUTS:
    close = math.isclose(a, b, rel_tol=rel_tol)
    tolerance = rel_tol * max(abs(a), abs(b))
    abs_diff = abs(a - b)
    print(fmt.format(a, b, rel_tol, abs_diff, tolerance, close))
```

因为误差设置为0.1，0.1和0.09的比较返回 `Flase` 。

```
$ python3 math_isclose.py

   a        b     rel_tol  abs(a-b) tolerance  close
-------- -------- -------- -------- -------- --------
 1000.00   900.00     0.10   100.00   100.00     True
  100.00    90.00     0.10    10.00    10.00     True
   10.00     9.00     0.10     1.00     1.00     True
    1.00     0.90     0.10     0.10     0.10     True
```

```
# math_isclose_abs_tol.py

import math

INPUTS = [
    (1.0, 1.0 + 1e-07, 1e-08),
    (1.0, 1.0 + 1e-08, 1e-08),
    (1.0, 1.0 + 1e-09, 1e-08),
]

print('{:^8} {:^11} {:^8} {:^10} {:^8}'.format(
    'a', 'b', 'abs_tol', 'abs(a-b)', 'close')
)
print('{:-^8} {:-^11} {:-^8} {:-^10} {:-^8}'.format(
    '-', '-', '-', '-', '-'),
)

for a, b, abs_tol in INPUTS:
    close = math.isclose(a, b, abs_tol=abs_tol)
    abs_diff = abs(a - b)
    print('{:8.2f} {:11} {:8} {:0.9f} {!s:>8}'.format(
        a, b, abs_tol, abs_diff, close))
```

对于绝对公差，输入值之间的差必须小于给定的公差。

```
$ python3 math_isclose_abs_tol.py

   a          b      abs_tol   abs(a-b)   close
-------- ----------- -------- ---------- --------
    1.00   1.0000001    1e-08 0.000000100    False
    1.00  1.00000001    1e-08 0.000000010     True
    1.00 1.000000001    1e-08 0.000000001     True
```

nan 和 inf 是特例。

```
# math_isclose_inf.py

import math
```

`nan` 永远不接近另一个值，包括它本身。 `inf` 仅仅接近自身。

```
$ python3 math_isclose_inf.py

nan, nan: False
nan, 1.0: False
inf, inf: True
inf, 1.0: False
```

浮点数到整数的转换

`math` 模块提供了三个用来转换浮点数为整数的函数。每个函数都采用不同的方法，并将在不同的情况下很有用。

最简单的是 `trunc()` ，它会截断小数点后的数字，仅保留构成值的整数部分的有效数字。 `floor()` 将输入值转换为最大的在前整数， `ceil()` (天花板) 返回在输入值之后顺序产生最大的整数。

```
# math_integers.py

import math

HEADINGS = ('i', 'int', 'trunk', 'floor', 'ceil')
print('{:^5} {:^5} {:^5} {:^5} {:^5}'.format(*HEADINGS))
print('{:-^5} {:-^5} {:-^5} {:-^5} {:-^5}'.format(
    '', '', '', '', '',
))

fmt = '{:5.1f} {:5.1f} {:5.1f} {:5.1f} {:5.1f}'

TEST_VALUES = [
    -1.5,
    -0.8,
    -0.5,
    -0.2,
    0,
    0.2,
```

知乎

```
for i in TEST_VALUES:
    print(fmt.format(
        i,
        int(i),
        math.trunc(i),
        math.floor(i),
        math.ceil(i),
    ))
```

trunc() 等价于直接转换为 int 。

```
$ python3 math_integers.py

  i    int  trunk floor ceil
----- ----- ----- ----- -----
 -1.5  -1.0  -1.0  -2.0  -1.0
 -0.8   0.0   0.0  -1.0   0.0
 -0.5   0.0   0.0  -1.0   0.0
 -0.2   0.0   0.0  -1.0   0.0
  0.0   0.0   0.0   0.0   0.0
  0.2   0.0   0.0   0.0   1.0
  0.5   0.0   0.0   0.0   1.0
  0.8   0.0   0.0   0.0   1.0
  1.0   1.0   1.0   1.0   1.0
```

modf() 接受单个浮点数，并返回一个包含输入值的小数和整数部分的元组。

```
# math_modf.py

import math

for i in range(6):
    print('{}/2 = {}'.format(i, math.modf(i / 2.0)))
```

返回值中的两数均为浮点数。

```
$ python3 math_modf.py
```

```
3/2 = (0.5, 1.0)
4/2 = (0.0, 2.0)
5/2 = (0.5, 2.0)
```

frexp() 返回浮点数的尾数和指数，可用于创建该值更可移植的表示形式。

```
math_frexp.py
import math

print('{:^7} {:^7} {:^7}'.format('x', 'm', 'e'))
print('{:-^7} {:-^7} {:-^7}'.format('', '', ''))

for x in [0.1, 0.5, 4.0]:
    m, e = math.frexp(x)
    print('{:7.2f} {:7.2f} {:7d}'.format(x, m, e))
```

frexp() 使用公式 x = m * 2 ** e ，并返回值 m 和 e 。

```
$ python3 math_frexp.py

   x       m       e
------- ------- -------
   0.10    0.80      -3
   0.50    0.50       0
   4.00    0.50       3
```

ldexp() 是 frexp() 的反函数。

```
# math_ldexp.py

import math

print('{:^7} {:^7} {:^7}'.format('m', 'e', 'x'))
print('{:-^7} {:-^7} {:-^7}'.format('', '', ''))

INPUTS = [
    (0.8, -3),
    (0.5, 0),
```

```
    x = math.ldexp(m, e)
    print('{:7.2f} {:7d} {:7.2f}'.format(m, e, x))
```

正负号

数字的绝对值是不带符号的值。 使用 `fabs()` 计算浮点数的绝对值。

```
# math_fabs.py

import math

print(math.fabs(-1.1))
print(math.fabs(-0.0))
print(math.fabs(0.0))
print(math.fabs(1.1))
```

实际上，浮点数的绝对值表示为正值。

```
$ python3 math_fabs.py

1.1
0.0
0.0
1.1
```

为确定值的符号，可以赋予一组值相同的符号或比较两个值，使用 `copysign()` 设置已知有效值的符号。

```
# math_copysign.py

import math

HEADINGS = ('f', 's', '< 0', '> 0', '= 0')
print('{:^5} {:^5} {:^5} {:^5} {:^5}'.format(*HEADINGS))
print('{:-^5} {:-^5} {:-^5} {:-^5} {:-^5}'.format(
    '', '', '', '', '',
))
```

```
    1.0,
    float('-inf'),
    float('inf'),
    float('-nan'),
    float('nan'),
]

for f in VALUES:
    s = int(math.copysign(1, f))
    print('{:5.1f} {:5d} {!s:5} {!s:5} {!s:5}'.format(
        f, s, f < 0, f > 0, f == 0,
    ))
```

像 copysign() 这样的额外函数是必需的，因为把 nan 和 -nan 直接与其他值比较是不可行的。

```
$ python3 math_copysign.py

  f     s    < 0   > 0   = 0
----- ----- ----- ----- -----
 -1.0    -1 True  False False
  0.0     1 False False True
  1.0     1 False True  False
 -inf    -1 True  False False
  inf     1 False True  False
  nan    -1 False False False
  nan     1 False False False
```

常用计算

在二进制浮点内存中表示准确值具有挑战性。一些值不能被准确表示，并且通过重复计算操作某个值的频率越高，出现表示错误的可能性越高。 math 模块提供了计算浮点数序列之和的函数，该函数使用有效的算法最小化这些错误。

```
# math_fsum.py

import math

values = [0.1] * 10
```

知乎

```
s = 0.0
for i in values:
    s += i
print('for-loop    : {:.20f}'.format(s))

print('math.fsum() : {:.20f}'.format(math.fsum(values)))
```

给定十个值的序列，每个值都是0.1，序列和的预期值为1.0。然而，由于0.1不能被精确地表示为浮点数，误差将被引入到求和中，除非使用 `fsum()` 进行计算。

```
$ python3 math_fsum.py

Input values: [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
sum()       : 0.9999999999999988898
for-loop    : 0.9999999999999988898
math.fsum() : 1.0000000000000000000
```

`factorial()` 通常用来计算对象序列的排列数和组合数。正整数n的阶乘，表示为 n!，其定义为递归地(n - 1) !* n，以0!= = 1结束。

```
# math_factorial.py

import math

for i in [0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.1]:
    try:
        print('{:2.0f} {:6.0f}'.format(i, math.factorial(i)))
    except ValueError as err:
        print('Error computing factorial({}): {}'.format(i, err))
```

`factorial()` 仅适用于整数，但会接受浮点参数，只要它们可以转换为整数而不会丢失值。

```
$ python3 math_factorial.py

 0      1
 1      1
 2      2
```

知乎

gamma() 与 factorial() 相似，除了它处理的是实数且值向下移动了一位（ gammar() 等价于(n-1)!)。

```
# math_gamma.py

import math

for i in [0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6]:
    try:
        print('{:2.1f} {:6.2f}'.format(i, math.gamma(i)))
    except ValueError as err:
        print('Error computing gamma({}): {}'.format(i, err))
```

由于0会导致初始值为负，故不允许这样做。

```
$ python3 math_gamma.py

Error computing gamma(0): math domain error
1.1   0.95
2.2   1.10
3.3   2.68
4.4  10.14
5.5  52.34
6.6 344.70
```

lgamma() 返回输入值的gamma绝对值的自然对数。

```
# math_lgamma.py

import math

for i in [0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6]:
    try:
        print('{:2.1f} {:.20f} {:.20f}'.format(
            i,
            math.lgamma(i),
            math.log(math.gamma(i)),
```

知乎

使用 `lgamma()` 比使用 `gamma()` 的结果单独计算对数保留更高的精度。

```
$ python3 math_lgamma.py

Error computing lgamma(0): math domain error
1.1 -0.04987244125984036103 -0.04987244125983997245
2.2 0.09694746679063825923 0.09694746679063866168
3.3 0.98709857789473387513 0.98709857789473409717
4.4 2.31610349142485727469 2.31610349142485727469
5.5 3.95781396761871651080 3.95781396761871606671
6.6 5.84268005527463252236 5.84268005527463252236
```

取模运算符 (%) 用来计算除法表达式的余数 (i.e., `5 % 2 = 1`)。内置在语言中的算符可以很好的处理整数，但与许多其他浮点运算符一样，间接计算会导致数据丢失的代表性问题。 `fmod()` 为浮点数提供了更精确的实现。

```
# math_fmod.py

import math

print('{:^4} {:^4} {:^5} {:^5}'.format(
    'x', 'y', '%', 'fmod'))
print('{:-^4} {:-^4} {:-^5} {:-^5}'.format(
    '-', '-', '-', '-'))

INPUTS = [
    (5, 2),
    (5, -2),
    (-5, 2),
]

for x, y in INPUTS:
    print('{:4.1f} {:4.1f} {:5.2f} {:5.2f}'.format(
        x,
        y,
        x % y,
        math.fmod(x, y),
    ))
```

知乎

```
$ python3 math_fmod.py

 x    y     %   fmod
---- ---- ----- -----
 5.0  2.0  1.00  1.00
 5.0 -2.0 -1.00  1.00
-5.0  2.0  1.00 -1.00
```

使用 gcd() 寻找能被两数整除的最大整数，即最大公约数。

```
# math_gcd.py

import math

print(math.gcd(10, 8))
print(math.gcd(10, 0))
print(math.gcd(50, 225))
print(math.gcd(11, 9))
print(math.gcd(0, 0))
```

如果两个值都为0，则结果为0。

```
$ python3 math_gcd.py

2
10
25
1
0
```

指数和对数

指数增长曲线出现在经济学，物理学和其他科学中。 Python具有内置的幂运算符（“**”），但当需要可调用函数作为另一个函数的参数时， pow() 可能会很有用。

```
# math_pow.py
```

```
    # Typical uses
    (2, 3),
    (2.1, 3.2),

    # Always 1
    (1.0, 5),
    (2.0, 0),

    # Not-a-number
    (2, float('nan')),

    # Roots
    (9.0, 0.5),
    (27.0, 1.0 / 3),
]

for x, y in INPUTS:
    print('{:5.1f} ** {:5.3f} = {:6.3f}'.format(
        x, y, math.pow(x, y)))
```

将1增加到任意幂总是返回1.0，将任意值增加到0次方也是如此。对非数字值 `nan` 的大多数操作都返回 `nan` 。 如果指数小于1，则 `pow()` 计算开方。

```
$ python3 math_pow.py

  2.0 ** 3.000 =  8.000
  2.1 ** 3.200 = 10.742
  1.0 ** 5.000 =  1.000
  2.0 ** 0.000 =  1.000
  2.0 **   nan =    nan
  9.0 ** 0.500 =  3.000
 27.0 ** 0.333 =  3.000
```

由于平方根（1/2的指数）被频繁使用，因此有一个可以单独计算的函数。

```
# math_sqrt.py

import math
```

```
except ValueError as err:
    print('Cannot compute sqrt(-1):', err)
```

计算负数的平方根需要用到复数，这不是 `math` 能处理的。任何计算负数平方根的尝试都会导致ValueError。

```
$ python3 math_sqrt.py

3.0
1.7320508075688772
Cannot compute sqrt(-1): math domain error
```

对数函数在当 x = b ** y 时找到 y 。默认情况下， `log()` 计算自然对数(以e为底数)。如果提供第二个参数，则以该值为底数。

```
# math_log.py

import math

print(math.log(8))
print(math.log(8, 2))
print(math.log(0.5, 2))
```

当x < 1时的对数结果为负。

```
$ python3 math_log.py

2.0794415416798357
3.0
-1.0
```

给定浮点数表示法和舍入误差， `log(x，b)` 产生的计算值的准确性有限，尤其是对于某些基数而言。 `log10()` 使用比 `log()` 更精确的算法来计算 `log(x，10)` 。

```
# math_log10.py

import math
```

```
print('{:-^2} {:-^12} {:-^10} {:-^20} {:-^8}'.format(
    '', '', '', '', '',
))

for i in range(0, 10):
    x = math.pow(10, i)
    accurate = math.log10(x)
    inaccurate = math.log(x, 10)
    match = '' if int(inaccurate) == i else '*'
    print('{:2d} {:12.1f} {:10.8f} {:20.18f} {:^5}'.format(
        i, x, accurate, inaccurate, match,
    ))
```

输出结尾中带有 * 标记的显示了不精确的值。

```
$ python3 math_log10.py

i       x         accurate       inaccurate      mismatch
-- ------------ ---------- -------------------- --------
 0          1.0 0.00000000 0.00000000000000000
 1         10.0 1.00000000 1.00000000000000000
 2        100.0 2.00000000 2.00000000000000000
 3       1000.0 3.00000000 2.99999999999999556   *
 4      10000.0 4.00000000 4.00000000000000000
 5     100000.0 5.00000000 5.00000000000000000
 6    1000000.0 6.00000000 5.99999999999999112   *
 7   10000000.0 7.00000000 7.00000000000000000
 8  100000000.0 8.00000000 8.00000000000000000
 9 1000000000.0 9.00000000 8.99999999999998224   *
```

与 log10() 相似， log2() 计算等价于 math.log(x, 2) 。

```
# math_log2.py
import math

print('{:>2} {:^5} {:^5}'.format(
    'i', 'x', 'log2',
))
print('{:-^2} {:-^5} {:-^5}'.format(
```

```
    x = math.pow(2, i)
    result = math.log2(x)
    print('{:2d} {:5.1f} {:5.1f}'.format(
        i, x, result,
    ))
```

根据底层平台的不同，内置的和专用的函数通过使用以2为底的专用算法，可提供更好的性能和准确性，这是更通用的函数所没有的。

```
$ python3 math_log2.py

 i    x   log2
-- ----- -----
 0   1.0   0.0
 1   2.0   1.0
 2   4.0   2.0
 3   8.0   3.0
 4  16.0   4.0
 5  32.0   5.0
 6  64.0   6.0
 7 128.0   7.0
 8 256.0   8.0
 9 512.0   9.0
```

`log1p()` 计算牛顿-墨卡托级数(1+x的自然对数)。

```
# math_log1p.py

import math

x = 0.000000000000000000000001
print('x       :', x)
print('1 + x   :', 1 + x)
print('log(1+x):', math.log(1 + x))
print('log1p(x):', math.log1p(x))
```

`log1p()` 对于非常接近于零的 x 值更准确，因为它使用了一种补偿初始加法的四舍五入误差的算法。

```
x       : 1e-25
1 + x   : 1.0
log(1+x): 0.0
log1p(x): 1e-25
```

exp() 计算指数函数值 (e**x)。

```
# math_exp.py
import math

x = 2

fmt = '{:.20f}'
print(fmt.format(math.e ** 2))
print(fmt.format(math.pow(math.e, 2)))
print(fmt.format(math.exp(2)))
```

与其他特殊函数一样，它使用的算法所产生的结果比通用等效的 math.pow(math.e，x) 更为精确。

```
$ python3 math_exp.py

7.38905609893064951876
7.38905609893064951876
7.38905609893065040694
```

expm1() 是 log1p() 的反函数，用于计算 e ** x - 1 。

```
# math_expm1.py

import math

x = 0.000000000000000000000001

print(x)
print(math.exp(x) - 1)
print(math.expm1(x))
```

```
1e-25
0.0
1e-25
```

角度

虽然角度制在日常生活中更常用，但在科学和数学中弧度制是衡量角度的标准单位。1弧度是由两条线在圆心相交而形成的角度，两条线的末端在圆周上相距一个半径。

圆的周长记为 `2πr` ，因此弧度和之间存在一种联系，该值在三角函数计算中经常出现。这种联系使得弧度被应用在三角学和微积分中，因为使用它们可产生更紧凑的公式。

要将度转换为弧度，请使用 `radians()` 。

```
# math_radians.py

import math

print('{:^7} {:^7} {:^7}'.format(
    'Degrees', 'Radians', 'Expected'))
print('{:-^7} {:-^7} {:-^7}'.format(
    '', '', ''))

INPUTS = [
    (0, 0),
    (30, math.pi / 6),
    (45, math.pi / 4),
    (60, math.pi / 3),
    (90, math.pi / 2),
    (180, math.pi),
    (270, 3 / 2.0 * math.pi),
    (360, 2 * math.pi),
]

for deg, expected in INPUTS:
    print('{:7d} {:7.2f} {:7.2f}'.format(
        deg,
        math.radians(deg),
        expected,
```

知乎

```
$ python3 math_radians.py

Degrees Radians Expected
------- ------- -------
      0    0.00    0.00
     30    0.52    0.52
     45    0.79    0.79
     60    1.05    1.05
     90    1.57    1.57
    180    3.14    3.14
    270    4.71    4.71
    360    6.28    6.28
```

要将弧度转换为度，请使用 degree() 。

```
# math_degrees.py

import math

INPUTS = [
    (0, 0),
    (math.pi / 6, 30),
    (math.pi / 4, 45),
    (math.pi / 3, 60),
    (math.pi / 2, 90),
    (math.pi, 180),
    (3 * math.pi / 2, 270),
    (2 * math.pi, 360),
]

print('{:^8} {:^8} {:^8}'.format(
    'Radians', 'Degrees', 'Expected'))
print('{:-^8} {:-^8} {:-^8}'.format('', '', ''))
for rad, expected in INPUTS:
    print('{:8.2f} {:8.2f} {:8.2f}'.format(
        rad,
        math.degrees(rad),
        expected,
    ))
```

```
Radians  Degrees  Expected
-------- -------- --------
    0.00     0.00     0.00
    0.52    30.00    30.00
    0.79    45.00    45.00
    1.05    60.00    60.00
    1.57    90.00    90.00
    3.14   180.00   180.00
    4.71   270.00   270.00
    6.28   360.00   360.00
```

三角学

三角函数将三角形中的角度与其边长相关联。它们出现在周期性的公式中，如谐波、圆周运动，或处理角度时。标准库中的所有三角函数都采用以弧度表示的角度。给定直角三角形中的某个角度，正弦是与该角度相对的边与斜边的比值（sin A =相对/斜边）。余弦是相邻边与斜边的长度之比（cos A =相邻/斜边）。正切是对边和邻边之比(tan A =对边/邻边)。

```
# math_trig.py

import math

print('{:^7} {:^7} {:^7} {:^7} {:^7}'.format(
    'Degrees', 'Radians', 'Sine', 'Cosine', 'Tangent'))
print('{:-^7} {:-^7} {:-^7} {:-^7} {:-^7}'.format(
    '-', '-', '-', '-', '-'))

fmt = '{:7.2f} {:7.2f} {:7.2f} {:7.2f} {:7.2f}'

for deg in range(0, 361, 30):
    rad = math.radians(deg)
    if deg in (90, 270):
        t = float('inf')
    else:
        t = math.tan(rad)
    print(fmt.format(deg, rad, math.sin(rad), math.cos(rad), t))
```

tan 也可以被定义为角的正弦与其余弦之比，并且由于π/ 2和3π/ 2弧度的余弦为0，所以 tan 是

知乎

```
Degrees Radians  Sine   Cosine  Tangent
------- ------- ------- ------- -------
   0.00    0.00    0.00    1.00    0.00
  30.00    0.52    0.50    0.87    0.58
  60.00    1.05    0.87    0.50    1.73
  90.00    1.57    1.00    0.00     inf
 120.00    2.09    0.87   -0.50   -1.73
 150.00    2.62    0.50   -0.87   -0.58
 180.00    3.14    0.00   -1.00   -0.00
 210.00    3.67   -0.50   -0.87    0.58
 240.00    4.19   -0.87   -0.50    1.73
 270.00    4.71   -1.00   -0.00     inf
 300.00    5.24   -0.87    0.50   -1.73
 330.00    5.76   -0.50    0.87   -0.58
 360.00    6.28   -0.00    1.00   -0.00
```

给定点 (x，y)，点 [(0，0)，(x，0)，(x，y)] 构成的三角形的斜边长为 （x ** 2 + y ** 2 ）** 1/2，可以用 hypot() 计算。

```
# math_hypot.py

import math

print('{:^7} {:^7} {:^10}'.format('X', 'Y', 'Hypotenuse'))
print('{:-^7} {:-^7} {:-^10}'.format('', '', ''))

POINTS = [
    # simple points
    (1, 1),
    (-1, -1),
    (math.sqrt(2), math.sqrt(2)),
    (3, 4),  # 3-4-5 triangle
    # on the circle
    (math.sqrt(2) / 2, math.sqrt(2) / 2),  # pi/4 rads
    (0.5, math.sqrt(3) / 2),  # pi/3 rads
]

for x, y in POINTS:
    h = math.hypot(x, y)
    print('{:7.2f} {:7.2f} {:7.2f}'.format(x, y, h))
```

```
$ python3 math_hypot.py

   X       Y    Hypotenuse
------- ------- ----------
   1.00    1.00    1.41
  -1.00   -1.00    1.41
   1.41    1.41    2.00
   3.00    4.00    5.00
   0.71    0.71    1.00
   0.50    0.87    1.00
```

该函数还可以被用来求两点之间的距离。

```
# math_distance_2_points.py

import math

print('{:^8} {:^8} {:^8} {:^8} {:^8}'.format(
    'X1', 'Y1', 'X2', 'Y2', 'Distance',
))
print('{:-^8} {:-^8} {:-^8} {:-^8} {:-^8}'.format(
    '', '', '', '', '',
))

POINTS = [
    ((5, 5), (6, 6)),
    ((-6, -6), (-5, -5)),
    ((0, 0), (3, 4)),  # 3-4-5 triangle
    ((-1, -1), (2, 3)),  # 3-4-5 triangle
]

for (x1, y1), (x2, y2) in POINTS:
    x = x1 - x2
    y = y1 - y2
    h = math.hypot(x, y)
    print('{:8.2f} {:8.2f} {:8.2f} {:8.2f} {:8.2f}'.format(
        x1, y1, x2, y2, h,
    ))
```

```
   X1       Y1       X2       Y2     Distance
-------- -------- -------- -------- --------
    5.00     5.00     6.00     6.00     1.41
   -6.00    -6.00    -5.00    -5.00     1.41
    0.00     0.00     3.00     4.00     5.00
   -1.00    -1.00     2.00     3.00     5.00
```

math 还定义了反三角函数。

```
# math_inverse_trig.py

import math

for r in [0, 0.5, 1]:
    print('arcsine({:.1f})    = {:5.2f}'.format(r, math.asin(r)))
    print('arccosine({:.1f})  = {:5.2f}'.format(r, math.acos(r)))
    print('arctangent({:.1f}) = {:5.2f}'.format(r, math.atan(r)))
    print()
```

1.57约等于π/ 2或90度，即正弦为1而余弦为0的角度。

```
$ python3 math_inverse_trig.py

arcsine(0.0)    =  0.00
arccosine(0.0)  =  1.57
arctangent(0.0) =  0.00

arcsine(0.5)    =  0.52
arccosine(0.5)  =  1.05
arctangent(0.5) =  0.46

arcsine(1.0)    =  1.57
arccosine(1.0)  =  0.00
arctangent(1.0) =  0.79
```

双曲函数

双曲函数出现在线性微分方程中，在处理电磁场，流体力学，狭义相对论和其他高等物理和数学时

```
import math

print('{:^6} {:^6} {:^6} {:^6}'.format(
    'X', 'sinh', 'cosh', 'tanh',
))
print('{:-^6} {:-^6} {:-^6} {:-^6}'.format('', '', '', ''))

fmt = '{:6.4f} {:6.4f} {:6.4f} {:6.4f}'

for i in range(0, 11, 2):
    x = i / 10.0
    print(fmt.format(
        x,
        math.sinh(x),
        math.cosh(x),
        math.tanh(x),
    ))
```

余弦函数和正弦函数表示一个圆，而双曲余弦函数和双曲正弦函数表示双曲线的一半。

```
$ python3 math_hyperbolic.py

  X     sinh   cosh   tanh
------ ------ ------ ------
0.0000 0.0000 1.0000 0.0000
0.2000 0.2013 1.0201 0.1974
0.4000 0.4108 1.0811 0.3799
0.6000 0.6367 1.1855 0.5370
0.8000 0.8881 1.3374 0.6640
1.0000 1.1752 1.5431 0.7616
```

反双曲函数 `acosh()` ， `asinh()` 和 `atanh()` 也可用。

特殊函数

高斯误差函数常用于统计。

```
# math_erf.py
```

```
print('{:-^5} {:-^7}'.format('', ''))

for x in [-3, -2, -1, -0.5, -0.25, 0, 0.25, 0.5, 1, 2, 3]:
    print('{:5.2f} {:7.4f}'.format(x, math.erf(x)))
```

对于误差函数， `erf(-x) == -erf(x)` .

```
$ python3 math_erf.py

  x   erf(x)
----- -------
-3.00 -1.0000
-2.00 -0.9953
-1.00 -0.8427
-0.50 -0.5205
-0.25 -0.2763
 0.00  0.0000
 0.25  0.2763
 0.50  0.5205
 1.00  0.8427
 2.00  0.9953
 3.00  1.0000
```

互补误差函数为 `1-erf(x)` 。

```
# math_erfc.py

import math

print('{:^5} {:7}'.format('x', 'erfc(x)'))
print('{:-^5} {:-^7}'.format('', ''))

for x in [-3, -2, -1, -0.5, -0.25, 0, 0.25, 0.5, 1, 2, 3]:
    print('{:5.2f} {:7.4f}'.format(x, math.erfc(x)))
```

`erfc()` 的实现避免了从1减去x的小数值的精度误差。

```
$ python3 math_erfc.py
```

```
-2.00  1.9953
-1.00  1.8427
-0.50  1.5205
-0.25  1.2763
 0.00  1.0000
 0.25  0.7237
 0.50  0.4795
 1.00  0.1573
 2.00  0.0047
 3.00  0.0000
```

> **See also**
> - Standard library documentation for math
> - IEEE floating point arithmetic in Python – Blog post by John Cook about how special values arise and are dealt with when doing math in Python.
> - SciPy – Open source libraryes for scientific and mathematical calculations in Python. - **PEP 485** – "A function for testing approximate equality"

编辑于 2020-02-15

Python 标准库　编程语言　Python 入门

推荐阅读

Python初学者必须吃透这69个内置函数!

来源：公众号 AI入门学习所谓内置函数，就是Python提供的, 可以直接拿来直接用的函数，比如print，range、input等。Python内置的这些函数非常精巧 且强大的，对初学者来说，经常会忽略，但...

知乎

1 条评论 ⇌ 切换为时间排序

写下你的评论...

知乎用户 2020-02-15

资词!

赞