01_math_module

June 29, 2019

```
In [1]: import math

In [2]: print(dir(math))

['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asi


In [3]: help(math)

Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.
```

```
atanh(x, /)
    Return the inverse hyperbolic tangent of x.

ceil(x, /)
    Return the ceiling of x as an Integral.

    This is the smallest integer >= x.

copysign(x, y, /)
    Return a float with the magnitude (absolute value) of x but the sign of y.

    On platforms that support signed zeros, copysign(1.0, -0.0)
    returns -1.0.

cos(x, /)
    Return the cosine of x (measured in radians).

cosh(x, /)
    Return the hyperbolic cosine of x.

degrees(x, /)
    Convert angle x from radians to degrees.

erf(x, /)
    Error function at x.

erfc(x, /)
    Complementary error function at x.

exp(x, /)
    Return e raised to the power of x.

expm1(x, /)
    Return exp(x)-1.

    This function avoids the loss of precision involved in the direct evaluation of exp(x)-

fabs(x, /)
    Return the absolute value of the float x.

factorial(x, /)
    Find x!.

    Raise a ValueError if x is negative or non-integral.

floor(x, /)
    Return the floor of x as an Integral.
```

```
    This is the largest integer <= x.

fmod(x, y, /)
    Return fmod(x, y), according to platform C.

    x % y may differ.

frexp(x, /)
    Return the mantissa and exponent of x, as pair (m, e).

    m is a float and e is an int, such that x = m * 2.**e.
    If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

fsum(seq, /)
    Return an accurate floating point sum of values in the iterable seq.

    Assumes IEEE-754 floating point arithmetic.

gamma(x, /)
    Gamma function at x.

gcd(x, y, /)
    greatest common divisor of x and y

hypot(x, y, /)
    Return the Euclidean distance, sqrt(x*x + y*y).

isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
    Determine whether two floating point numbers are close in value.

      rel_tol
        maximum difference for being considered "close", relative to the
        magnitude of the input values
      abs_tol
        maximum difference for being considered "close", regardless of the
        magnitude of the input values

    Return True if a is close in value to b, and False otherwise.

    For the values to be considered close, the difference between them
    must be smaller than at least one of the tolerances.

    -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
    is, NaN is not close to anything, even itself.  inf and -inf are
    only close to themselves.

isfinite(x, /)
```

```
    Return True if x is neither an infinity nor a NaN, and False otherwise.

isinf(x, /)
    Return True if x is a positive or negative infinity, and False otherwise.

isnan(x, /)
    Return True if x is a NaN (not a number), and False otherwise.

ldexp(x, i, /)
    Return x * (2**i).

    This is essentially the inverse of frexp().

lgamma(x, /)
    Natural logarithm of absolute value of Gamma function at x.

log(...)
    log(x, [base=math.e])
    Return the logarithm of x to the given base.

    If the base not specified, returns the natural logarithm (base e) of x.

log10(x, /)
    Return the base 10 logarithm of x.

log1p(x, /)
    Return the natural logarithm of 1+x (base e).

    The result is computed in a way which is accurate for x near zero.

log2(x, /)
    Return the base 2 logarithm of x.

modf(x, /)
    Return the fractional and integer parts of x.

    Both results carry the sign of x and are floats.

pow(x, y, /)
    Return x**y (x to the power of y).

radians(x, /)
    Convert angle x from degrees to radians.

remainder(x, y, /)
    Difference between x and the closest integer multiple of y.

    Return x - n*y where n*y is the closest integer multiple of y.
```

```
        In the case where x is exactly halfway between two multiples of
        y, the nearest even value of n is used. The result is always exact.

    sin(x, /)
        Return the sine of x (measured in radians).

    sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.

    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)
```

Constants

```
In [4]: math.pi

Out[4]: 3.141592653589793

In [5]: math.e

Out[5]: 2.718281828459045

In [6]: math.inf

Out[6]: inf
```

```
In [44]: math.nan

Out[44]: nan

In [45]: math.tau

Out[45]: 6.283185307179586
```

Functions math.ceil, math.floor, math.trunc and int()

```
In [14]: math.ceil(9.7), math.ceil(9.5), math.ceil(9.1), math.ceil(9.000000000000000000001)

Out[14]: (10, 10, 10, 9)

In [16]: math.floor(9.7), math.floor(9.5), math.floor(9.1), math.floor(9.000000000000000000001)

Out[16]: (9, 9, 9, 9)

In [17]: math.trunc(9.7), math.trunc(9.5), math.trunc(9.1), math.trunc(9.000000000000000000001)

Out[17]: (9, 9, 9, 9)

In [15]: int(9.7), int(9.5), int(9.1),int(9.000000000000000000001)

Out[15]: (9, 9, 9, 9)

In [39]: round(9.7), round(9.5), round(9.1), round(9.000000000000000000001) # works in python

Out[39]: (10, 10, 9, 9)

In [40]: for each_num in range(10):
             print(round(math.pi, each_num))

3.0
3.1
3.14
3.142
3.1416
3.14159
3.141593
3.1415927
3.14159265
3.141592654
```

math.pow

```
In [18]: math.pow(2, 4), pow(2, 4)

Out[18]: (16.0, 16)
```

```
In [19]: math.pow(2.0, 4), pow(2.0, 4)

Out[19]: (16.0, 16.0)

In [21]: math.pow(2, 4, 8)

         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-21-40227826ccd1> in <module>
     ----> 1 math.pow(2, 4, 8)


         TypeError: pow() takes exactly 2 arguments (3 given)


In [23]: pow(2, 4, 8)

Out[23]: 0
```

math.remainder vs %

```
In [24]: math.remainder(10, 2), 10 % 2

Out[24]: (0.0, 0)

In [25]: math.remainder(10, 3), 10 % 3

Out[25]: (1.0, 1)
```

math.fabs vss abs()

```
In [27]: math.fabs(-9), abs(-9)

Out[27]: (9.0, 9)

In [28]: math.fabs(-9.6), abs(-9)

Out[28]: (9.6, 9)
```

math.factorial

```
In [30]: math.factorial(5) ,  5 * 4 * 3 * 2 * 1

Out[30]: (120, 120)
```

math.fsum() vs sum()

```
In [31]: math.fsum((1, 2, 3, 5)), sum((1, 2, 3, 5))
```

```
Out[31]: (11.0, 11)

In [32]: math.fsum((1.2, 2.5, 3.5, 5.5)), sum((1.2, 2.5, 3.5, 5.5))

Out[32]: (12.7, 12.7)
```

math.gcd

```
In [33]: math.gcd(12, 4) # Takes two numbers and gives GCD

Out[33]: 4
```

math.sqrt

```
In [36]: math.sqrt(81)

Out[36]: 9.0

In [37]: math.sqrt(84)

Out[37]: 9.16515138991168
```

math.fmod vs %

```
In [42]: math.fmod(10, 2), 10 % 2

Out[42]: (0.0, 0)

In [43]: math.fmod(10, 3), 10 % 3

Out[43]: (1.0, 1)
```

math.modf

```
In [46]: math.modf(1.5) # returns fractical, integer parts of a number

Out[46]: (0.5, 1.0)

In [47]: math.modf(32)

Out[47]: (0.0, 32.0)
```

math.exp

```
In [49]: math.exp(10), math.e ** 10

Out[49]: (22026.465794806718, 22026.465794806703)

In [50]: math.expm1(10), math.e ** 10 -1  # helps in retaining the precision of digits

Out[50]: (22025.465794806718, 22025.465794806703)
```

logarithm

```
In [51]: math.log(100)

Out[51]: 4.605170185988092

In [52]: math.log10(100)

Out[52]: 2.0

In [53]: math.log1p(100)

Out[53]: 4.61512051684126

In [54]: math.log2(100)

Out[54]: 6.643856189774724

In [55]: degrees = 45
         math.radians(degrees), (degrees/360.0) * 2 * math.pi

Out[55]: (0.7853981633974483, 0.7853981633974483)

In [56]: radians = 90
         math.degrees(radians), 180 * radians/math.pi

Out[56]: (5156.620156177409, 5156.620156177409)

In [57]: math.sin(90)

Out[57]: 0.8939966636005579

In [59]: math.sin(math.radians(90))

Out[59]: 1.0

In [60]: math.cos(math.radians(90))

Out[60]: 6.123233995736766e-17

In [61]: math.tan(math.radians(90))

Out[61]: 1.633123935319537e+16

In [64]: math.asin(1)

Out[64]: 1.5707963267948966

In [65]: math.acos(1)

Out[65]: 0.0

In [66]: math.atan(1)
```

```
Out[66]: 0.7853981633974483

In [70]: print(math.sinh(1))
         print(math.cosh(1))
         print(math.tanh(1))

1.1752011936438014
1.5430806348152437
0.7615941559557649


In [72]: print(math.asinh(1))
         print(math.acosh(1))
         print(math.atanh(0))

0.8813735870195429
0.0
0.0
```

math.hypot - returns Euclidean distance

```
In [75]: math.hypot(2, 3), math.sqrt(2 * 2 + 3 * 3)

Out[75]: (3.605551275463989, 3.605551275463989)
```

math.copysign(x, y) - return x value, with sign of y

```
In [76]: math.copysign(-9, 2)

Out[76]: 9.0

In [77]: math.copysign(-9, -2)

Out[77]: -9.0

In [78]: math.copysign(9, -2)

Out[78]: -9.0

In [79]: math.copysign(9, 2)

Out[79]: 9.0
```

'erf', 'erfc', 'gamma', 'lgamma', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'frexp'