01_math_module

October 27, 2019

[1]:
```
import math
```

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

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

[3]:
```
help(math)
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module 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)-1 for small 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

[4]:

```
math.pi
```

[4]:
```
3.141592653589793
```

[5]:
```
math.e
```

[5]:
```
2.718281828459045
```

[6]:
```
math.inf
```

[6]:
```
inf
```

[7]:
```
math.nan
```

[7]:
```
nan
```

[8]:
```
math.tau
```

[8]:
```
6.283185307179586
```

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

[9]:
```
math.ceil(9.7), math.ceil(9.5), math.ceil(9.1), math.ceil(9.
  ↪00000000000000000001), math.ceil(9)
```

[9]:
```
(10, 10, 10, 9, 9)
```

[10]:
```
math.floor(9.7), math.floor(9.5), math.floor(9.1), math.floor(9.
  ↪00000000000000000001), math.floor(9)
```

[10]:
```
(9, 9, 9, 9, 9)
```

[11]:
```
math.trunc(9.7), math.trunc(9.5), math.trunc(9.1), math.trunc(9.
  ↪00000000000000000001), math.trunc(9)
```

[11]:
```
(9, 9, 9, 9, 9)
```

[12]:
```
int(9.7), int(9.5), int(9.1),int(9.00000000000000000001), int(9)
```

[12]:
```
(9, 9, 9, 9, 9)
```

[13]:
```
round(9.7), round(9.5), round(9.4), round(9.1), round(9.00000000000000000001)␣
  ↪# works in python 3 only
```

[13]:
```
(10, 10, 9, 9, 9)
```

[14]:
```
for each_num in range(10):
    print(each_num, '->',  round(math.pi, each_num))
```

```
0 -> 3.0
1 -> 3.1
2 -> 3.14
3 -> 3.142
4 -> 3.1416
5 -> 3.14159
6 -> 3.141593
7 -> 3.1415927
8 -> 3.14159265
9 -> 3.141592654
```

math.pow

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

```
[15]: (16.0, 16)
```

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

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

```
[17]: pow(2, 4, 8) == (2 ** 4) % 8
```

```
[17]: True
```

```
[18]: pow(2, 4, 8)
```

```
[18]: 0
```

```
[19]: math.pow(2, 4, 8)
```

```
      ␣
 ↪---------------------------------------------------------------------------

     TypeError                                 Traceback (most recent call␣
 ↪last)

     <ipython-input-19-40227826ccd1> in <module>
   ----> 1 math.pow(2, 4, 8)


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

math.remainder vs %

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

```
[20]: (0.0, 0)
```

```
[21]: math.remainder(10, 3), 10 % 3
```

```
[21]: (1.0, 1)
```

math.fmod vs %

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

```
[22]: (0.0, 0)
```

```
[23]: math.fmod(10, 3), 10 % 3
```

```
[23]: (1.0, 1)
```

math.fabs vs abs()

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

```
[24]: (9.0, 9)
```

[25]:
```
math.fabs(-9.6), abs(-9)
```

[25]:
```
(9.6, 9)
```

math.factorial

[26]:
```
math.factorial(5) ,  5 * 4 * 3 * 2 * 1
```

[26]:
```
(120, 120)
```

math.fsum() vs sum()

[27]:
```
math.fsum((1, 2, 3, 5)), sum((1, 2, 3, 5))
```

[27]:
```
(11.0, 11)
```

[28]:
```
math.fsum((1.2, 2.5, 3.5, 5.5)), sum((1.2, 2.5, 3.5, 5.5))
```

[28]:
```
(12.7, 12.7)
```

math.gcd

[29]:
```
math.gcd(12, 4) # Takes two numbers and gives GCD
```

[29]:
```
4
```

math.sqrt

[30]:
```
math.sqrt(81)
```

[30]:
```
9.0
```

[31]:
```
math.sqrt(84)
```

[31]:
```
9.16515138991168
```

math.modf

[32]:
```
math.modf(1.5) # returns fractical, integer parts of a number
```

[32]:
```
(0.5, 1.0)
```

[33]:
```
math.modf(32)
```

[33]:
```
(0.0, 32.0)
```

math.exp

[34]:
```
math.exp(10), math.e ** 10
```

[34]:
```
(22026.465794806718, 22026.465794806703)
```

[35]:
```
math.expm1(10), math.e ** 10 -1  # helps in retaining the precision of digits
```

[35]:
```
(22025.465794806718, 22025.465794806703)
```

logarithm

[36]:
```
math.log(100)
```

[36]:
```
4.605170185988092
```

[37]:
```
math.log10(100)
```

[37]:
```
2.0
```

[38]:
```
math.log1p(100)
```

[38]:
```
4.61512051684126
```

[39]:
```
math.log2(100)
```

[39]:
```
6.643856189774724
```

[40]:
```
degrees = 45
math.radians(degrees), (degrees/360.0) * 2 * math.pi
```

[40]:
```
(0.7853981633974483, 0.7853981633974483)
```

[41]:
```
radians = 90
math.degrees(radians), 180 * radians/math.pi
```

[41]:
```
(5156.620156177409, 5156.620156177409)
```

[42]:
```
math.sin(90)
```

[42]:
```
0.8939966636005579
```

[43]:
```
math.sin(math.radians(90))
```

[43]:
```
1.0
```

[44]:
```
math.cos(math.radians(90))
```

[44]:
```
6.123233995736766e-17
```

[45]:
```
math.tan(math.radians(90))
```

[45]:
```
1.633123935319537e+16
```

[46]:
```
math.asin(1)
```

[46]:
```
1.5707963267948966
```

[47]:
```
math.acos(1)
```

[47]:
```
0.0
```

[48]:
```
math.atan(1)
```

[48]:
```
0.7853981633974483
```

[49]:
```
print(math.sinh(1))
print(math.cosh(1))
print(math.tanh(1))
```

```
1.1752011936438014
1.5430806348152437
0.7615941559557649
```

[50]:
```
print(math.asinh(1))
print(math.acosh(1))
print(math.atanh(0))
```

```
0.8813735870195429
0.0
0.0
```

math.hypot - returns Euclidean distance

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

```
[51]: (3.605551275463989, 3.605551275463989)
```

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

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

```
[52]: 9.0
```

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

```
[53]: -9.0
```

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

```
[54]: -9.0
```

```
[55]: math.copysign(9, 2)
```

```
[55]: 9.0
```

isfinite, isinf, isnan

```
[56]: math.isnan(math.nan)
```

```
[56]: True
```

```
[57]: math.isfinite(78)
```

```
[57]: True
```

```
[58]: math.isfinite(math.inf)
```

```
[58]: False
```

```
[59]: math.isinf(math.inf)
```

```
[59]: True
```

math.gamma - returns gamma function of a value

```
[60]: math.gamma(1)
```

```
[60]: 1.0
```

```
[61]: math.gamma(32)
```

```
[61]: 8.222838654177925e+33
```

```
[62]: math.lgamma(32)
```

```
[62]: 78.0922235533153
```

```
[63]: math.lgamma(1)
```

```
[63]: 0.0
```

‘erf’, ‘erfc’, ‘isclose’, ‘ldexp’, ‘frexp’