9.2. `math` — Mathematical functions

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

These functions cannot be used with complex numbers; use the functions of the same name from the `cmath` module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1. Number-theoretic and representation functions

`math.`**`ceil`**(*x*)
Return the ceiling of *x* as a float, the smallest integer value greater than or equal to *x*.

`math.`**`copysign`**(*x*, *y*)
Return *x* with the sign of *y*. On a platform that supports signed zeros, `copysign(1.0, -0.0)` returns *-1.0*.

New in version 2.6.

`math.`**`fabs`**(*x*)
Return the absolute value of *x*.

`math.`**`factorial`**(*x*)
Return *x* factorial. Raises `ValueError` if *x* is not integral or is negative.

New in version 2.6.

`math.`**`floor`**(*x*)
Return the floor of *x* as a float, the largest integer value less than or equal to *x*.

`math.`**`fmod`**(*x*, *y*)
Return `fmod(x, y)`, as defined by the platform C library. Note that the Python expression `x % y` may not return the same result. The intent of the C standard is that `fmod(x, y)` be exactly (mathematically; to infinite precision) equal to `x - n*y` for some integer *n* such that the result has the same sign as *x* and magnitude less than `abs(y)`. Python's `x % y` returns a result with the sign of *y* instead, and may not be exactly computable for float arguments. For example, `fmod(-1e-100, 1e100)` is `-1e-100`, but the result of Python's `-1e-100 % 1e100` is `1e100-1e-100`, which cannot be represented exactly as a float, and rounds to the surprising `1e100`. For this reason, function `fmod()` is generally preferred when working with floats, while Python's `x % y` is preferred when working with integers.

math.**frexp**(*x*)

Return the mantissa and exponent of *x* as the pair `(m, e)`. *m* is a float and *e* is an integer such that `x == m * 2**e` exactly. If *x* is zero, returns `(0.0, 0)`, otherwise `0.5 <= abs(m) < 1`. This is used to “pick apart” the internal representation of a float in a portable way.

math.**fsum**(*iterable*)

Return an accurate floating point sum of values in the iterable. Avoids loss of precision by tracking multiple intermediate partial sums:

```
>>> sum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
0.9999999999999999
>>> fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
1.0
```

The algorithm’s accuracy depends on IEEE-754 arithmetic guarantees and the typical case where the rounding mode is half-even. On some non-Windows builds, the underlying C library uses extended precision addition and may occasionally double-round an intermediate sum causing it to be off in its least significant bit.

For further discussion and two alternative approaches, see the ASPN cookbook recipes for accurate floating point summation.

New in version 2.6.

math.**isinf**(*x*)

Check if the float *x* is positive or negative infinity.

New in version 2.6.

math.**isnan**(*x*)

Check if the float *x* is a NaN (not a number). For more information on NaNs, see the IEEE 754 standards.

New in version 2.6.

math.**ldexp**(*x*, *i*)

Return `x * (2**i)`. This is essentially the inverse of function `frexp()`.

math.**modf**(*x*)

Return the fractional and integer parts of *x*. Both results carry the sign of *x* and are floats.

math.**trunc**(*x*)

Return the `Real` value *x* truncated to an `Integral` (usually a long integer). Uses the `__trunc__` method.

New in version 2.6.

Note that `frexp()` and `modf()` have a different call/return pattern than their C equivalents: they take a single argument and return a pair of values, rather than returning their second return value through an ‘output parameter’ (there is no such thing in Python).

For the `ceil()`, `floor()`, and `modf()` functions, note that *all* floating-point numbers of sufficiently large magnitude are exact integers. Python floats typically carry no more than 53 bits of precision (the same as the platform C double type), in which case any float *x* with `abs(x) >= 2**52` necessarily has no fractional bits.

9.2.2. Power and logarithmic functions

`math.` **`exp`**(*x*)
Return `e**x`.

`math.` **`expm1`**(*x*)
Return `e**x - 1`. For small floats *x*, the subtraction in `exp(x) - 1` can result in a significant loss of precision; the `expm1()` function provides a way to compute this quantity to full precision:

```
>>> from math import exp, expm1
>>> exp(1e-5) - 1  # gives result accurate to 11 places
1.0000050000069649e-05
>>> expm1(1e-5)    # result accurate to full precision
1.0000050000166668e-05
```

New in version 2.7.

`math.` **`log`**(*x*[, *base*])
With one argument, return the natural logarithm of *x* (to base *e*).

With two arguments, return the logarithm of *x* to the given *base*, calculated as `log(x)/log(base)`.

Changed in version 2.3: base argument added.

`math.` **`log1p`**(*x*)
Return the natural logarithm of *1+x* (base *e*). The result is calculated in a way which is accurate for *x* near zero.

New in version 2.6.

`math.` **`log10`**(*x*)
Return the base-10 logarithm of *x*. This is usually more accurate than `log(x, 10)`.

`math.` **`pow`**(*x*, *y*)
Return `x` raised to the power `y`. Exceptional cases follow Annex 'F' of the C99 standard as far as possible. In particular, `pow(1.0, x)` and `pow(x, 0.0)` always return `1.0`, even when `x` is a zero or a NaN. If both `x` and `y` are finite, `x` is negative, and `y` is not an integer then `pow(x, y)` is undefined, and raises `ValueError`.

Unlike the built-in `**` operator, `math.pow()` converts both its arguments to type `float`. Use `**` or the built-in `pow()` function for computing exact integer powers.

Changed in version 2.6: The outcome of `1**nan` and `nan**0` was undefined.

`math.` **`sqrt`**(*x*)

Return the square root of *x*.

9.2.3. Trigonometric functions

math.**acos**(*x*)
Return the arc cosine of *x*, in radians.

math.**asin**(*x*)
Return the arc sine of *x*, in radians.

math.**atan**(*x*)
Return the arc tangent of *x*, in radians.

math.**atan2**(*y*, *x*)
Return `atan(y / x)`, in radians. The result is between `-pi` and `pi`. The vector in the plane from the origin to point `(x, y)` makes this angle with the positive X axis. The point of `atan2()` is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, `atan(1)` and `atan2(1, 1)` are both `pi/4`, but `atan2(-1, -1)` is `-3*pi/4`.

math.**cos**(*x*)
Return the cosine of *x* radians.

math.**hypot**(*x*, *y*)
Return the Euclidean norm, `sqrt(x*x + y*y)`. This is the length of the vector from the origin to point `(x, y)`.

math.**sin**(*x*)
Return the sine of *x* radians.

math.**tan**(*x*)
Return the tangent of *x* radians.

9.2.4. Angular conversion

math.**degrees**(*x*)
Convert angle *x* from radians to degrees.

math.**radians**(*x*)
Convert angle *x* from degrees to radians.

9.2.5. Hyperbolic functions

math.**acosh**(*x*)
Return the inverse hyperbolic cosine of *x*.

New in version 2.6.

math.**asinh**(*x*)

Return the inverse hyperbolic sine of *x*.

New in version 2.6.

math.**atanh**(*x*)
Return the inverse hyperbolic tangent of *x*.

New in version 2.6.

math.**cosh**(*x*)
Return the hyperbolic cosine of *x*.

math.**sinh**(*x*)
Return the hyperbolic sine of *x*.

math.**tanh**(*x*)
Return the hyperbolic tangent of *x*.

9.2.6. Special functions

math.**erf**(*x*)
Return the error function at *x*.

New in version 2.7.

math.**erfc**(*x*)
Return the complementary error function at *x*.

New in version 2.7.

math.**gamma**(*x*)
Return the Gamma function at *x*.

New in version 2.7.

math.**lgamma**(*x*)
Return the natural logarithm of the absolute value of the Gamma function at *x*.

New in version 2.7.

9.2.7. Constants

math.**pi**
The mathematical constant π = 3.141592…, to available precision.

math.**e**
The mathematical constant e = 2.718281…, to available precision.

CPython implementation detail: The `math` module consists mostly of thin wrappers around the platform C math library functions. Behavior in exceptional cases follows Annex F of the C99 standard where appropriate. The current implementation will raise `ValueError` for invalid operations like `sqrt(-1.0)` or `log(0.0)` (where C99 Annex F recommends signaling invalid operation or divide-by-zero), and `OverflowError` for results that overflow (for example, `exp(1000.0)`). A NaN will not be returned from any of the functions above unless one or more of the input arguments was a NaN; in that case, most functions will return a NaN, but (again following C99 Annex F) there are some exceptions to this rule, for example `pow(float('nan'), 0.0)` or `hypot(float('nan'), float('inf'))`.

Note that Python makes no effort to distinguish signaling NaNs from quiet NaNs, and behavior for signaling NaNs remains unspecified. Typical behavior is to treat all NaNs as though they were quiet.

Changed in version 2.6: Behavior in special cases now aims to follow C99 Annex F. In earlier versions of Python the behavior in special cases was loosely specified.

See also:

Module `cmath`
: Complex number versions of many of these functions.