`math` — Funciones matemáticas

Este módulo proporciona acceso a las funciones matemáticas definidas por el estándar C.

Estas funciones no se pueden usar con números complejos; use las funciones del mismo nombre del `cmath` módulo si necesita soporte para números complejos. La distinción entre las funciones que admiten números complejos y las que no se hace porque la mayoría de los usuarios no quieren aprender tantas matemáticas como se requiere para comprender los números complejos. Recibir una excepción en lugar de un resultado complejo permite una detección más temprana del número complejo inesperado utilizado como parámetro, de modo que el programador pueda determinar cómo y por qué se generó en primer lugar.

Este módulo proporciona las siguientes funciones. Excepto cuando se indique explícitamente lo contrario, todos los valores devueltos son flotantes.

Funciones de representación y teoría de números

`math.`**`ceil`**`(`*x*`)`

Devuelve el techo de *x* , el entero más pequeño mayor o igual que *x* . Si *x* no es un flotante, delega a `x.__ceil__`, que debería devolver un `Integral` valor.

`math.`**`comb`**`(`*norte* , *k* `)`

Devuelve el número de formas de elegir *k* elementos de *n* elementos sin repetición y sin orden.

Evalúa a cuando y se evalúa a cero cuando .`n! / (k! * (n - k)!)` `k <= n` `k > n`

También llamado coeficiente binomial porque es equivalente al coeficiente del k-ésimo término en la expansión polinomial de la expresión .`(1 + x) ** n`

Se genera `TypeError` si alguno de los argumentos no son números enteros. Aumenta `ValueError` si alguno de los argumentos es negativo.

Nuevo en la versión 3.8.

`math.`**`copysign`**`(`*x* , *y* `)`

Devuelve un flotante con la magnitud (valor absoluto) de *x* pero el signo de *y* . En plataformas que admiten ceros con signo, devuelve *-1.0* `.copysign(1.0, -0.0)`

`math.`**`fabs`**`(`*x*`)`

Devuelve el valor absoluto de *x* .

`math.`**`factorial`**`(`*x*`)`

Devuelve *x* factorial como un número entero. Aumenta `ValueError` si *x* no es integral o es negativa.

En desuso desde la versión 3.9: La aceptación de flotantes con valores enteros (como `5.0`) está en desuso.

`math.`**`floor`**`(`*x*`)`

`math.`**`fmod`**`(` *x* , *y* `)`

Return , tal como lo define la biblioteca de la plataforma C. Tenga en cuenta que la expresión de Python puede no devolver el mismo resultado. La intención del estándar C es que sea exactamente (matemáticamente, con una precisión infinita) igual a para algún número entero *n* tal que el resultado tenga el mismo signo que *x* y una magnitud menor que . Python devuelve un resultado con el signo de *y* en su lugar, y puede que no sea exactamente computable para argumentos flotantes. Por ejemplo, es , pero el resultado de Python es , que no se puede representar exactamente como un flotador y se redondea al sorprendente . Por esta razón, función`fmod(x, y)` `x % y` `fmod(x, y)` `x - n*y` `abs(y)` `x % y` `fmod(-1e-100, 1e100)` `-1e-100` `-1e-100 % 1e100` `1e100-1e-100` `1e100` `fmod()` generalmente se prefiere cuando se trabaja con flotantes, mientras que Python se prefiere cuando se trabaja con números enteros.`x % y`

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

Devuelve la mantisa y el exponente de *x* como el par . *m* es un flotante *ye* es un entero tal que exactamente. Si *x* es cero, devuelve , de lo contrario . Esto se utiliza para "separar" la representación interna de un flotador de forma portátil.`(m, e)` `x == m * 2**e` `(0.0, 0)` `0.5 <= abs(m) < 1`

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

Devuelve una suma de valores de coma flotante precisa en el iterable. Evita la pérdida de precisión mediante el seguimiento de múltiples sumas parciales intermedias:

```
>>> 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
```

La precisión del algoritmo depende de las garantías aritméticas de IEEE-754 y del caso típico en el que el modo de redondeo es medio par. En algunas compilaciones que no son de Windows, la biblioteca de C subyacente usa una adición de precisión extendida y, en ocasiones, puede redondear dos veces una suma intermedia, lo que provoca que esté desactivada en su bit menos significativo.

Para una discusión más detallada y dos enfoques alternativos, consulte las recetas del libro de cocina de ASPN para una suma precisa de punto flotante .

`math.`**`gcd`**`(` `* enteros` `)`

Devuelve el máximo común divisor de los argumentos enteros especificados. Si alguno de los argumentos es distinto de cero, el valor devuelto es el entero positivo más grande que es un divisor de todos los argumentos. Si todos los argumentos son cero, el valor devuelto es `0`. `gcd()` sin argumentos devuelve `0`.

Nuevo en la versión 3.5.

Cambiado en la versión 3.9: Se agregó soporte para un número arbitrario de argumentos. Anteriormente, solo se apoyaban dos argumentos.

`math.`**`isclose`**`(` *a* , *b* , `*` , `tol_rel = 1e-09` , `tol_abs = 0.0` `)`

Retorna `True`si los valores *a* y *b* están cerca uno del otro y en `False`caso contrario.

Si dos valores se consideran cercanos o no se determina de acuerdo con las tolerancias absolutas y relativas dadas.

rel_tol es la tolerancia relativa: es la diferencia máxima permitida entre *a* y *b* , en relación con el valor absoluto mayor de *a* o *b* . Por ejemplo, para establecer una

abs_tol es la tolerancia absoluta mínima, útil para comparaciones cercanas a cero. *abs_tol* debe ser al menos cero.

Si no se producen errores, el resultado será: `.abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)`

Los valores especiales IEEE 754 de `NaN`, `infy -inf`se manejarán de acuerdo con las normas IEEE. Específicamente, `NaN`no se considera cercano a ningún otro valor, incluido `NaN`. `infy -inf`solo se consideran cercanos a ellos mismos.

Nuevo en la versión 3.5.

Ver también: **PEP 485** - Una función para probar la igualdad aproximada

`math.`**`isfinite`**(*x*)

Retorna `True`si *x* no es ni un infinito ni un NaN, y en `False`caso contrario. (Tenga en cuenta que `0.0` *se* considera finito).

Nuevo en la versión 3.2.

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

Devuelve `True`si *x* es un infinito positivo o negativo, y en `False`caso contrario.

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

Retorna `True`si *x* es un NaN (no un número), y en `False`caso contrario.

`math.`**`isqrt`**( ) ¶

Devuelve la raíz cuadrada entera del entero no negativo *n* . Este es el piso de la raíz cuadrada exacta de *n* , o equivalentemente el mayor entero *a* tal que $a^2 \le n$.

Para algunas aplicaciones, puede ser más conveniente tener el menor número entero *a* tal que $n \le a^2$, o en otras palabras, el techo de la raíz cuadrada exacta de *n* . *Para n* positivo , esto se puede calcular usando `.a = 1 + isqrt(n - 1)`

Nuevo en la versión 3.8.

`math.`**`lcm`**(*`* enteros`*)

Devuelve el mínimo común múltiplo de los argumentos enteros especificados. Si todos los argumentos son distintos de cero, el valor devuelto es el entero positivo más pequeño que es un múltiplo de todos los argumentos. Si alguno de los argumentos es cero, el valor devuelto es `0`. `lcm()`sin argumentos devuelve `1`.

Nuevo en la versión 3.9.

`math.`**`ldexp`**(*x* , *yo*)

Volver _ Esta es esencialmente la inversa de la función `.x * (2**i)` `frexp()`

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

`math.`**`nextafter`**`(` *x* , *y* `)`

Devuelve el siguiente valor de coma flotante después de *x* hacia *y* .

Si *x* es igual a *y* , devuelve *y* .

Ejemplos:

- `math.nextafter(x, math.inf)` sube: hacia el infinito positivo.
- `math.nextafter(x, -math.inf)` va hacia abajo: hacia menos infinito.
- `math.nextafter(x, 0.0)` va hacia cero.
- `math.nextafter(x, math.copysign(math.inf, x))` se aleja de cero.

Ver `math.ulp()` también

Nuevo en la versión 3.9.

`math.`**`perm`**`(` *norte* , *k = ninguno* `)`

Devuelve el número de formas de elegir *k* elementos de *n* elementos sin repetición y con orden.

Evalúa a cuando y se evalúa a cero cuando `.n! / (n - k)!k <= nk > n`

Si *k* no se especifica o es Ninguno, entonces *k* por defecto es *n* y la función devuelve `n!` .

Se genera `TypeError` si alguno de los argumentos no son números enteros. Aumenta `ValueError` si alguno de los argumentos es negativo.

Nuevo en la versión 3.8.

`math.`**`prod`**`(` *iterable* , `*` , *inicio = 1* `)`

Calcule el producto de todos los elementos en la entrada *iterable* . *El valor inicial* predeterminado para el producto es `1` .

Cuando el iterable está vacío, devuelve el valor inicial. Esta función está diseñada específicamente para usarse con valores numéricos y puede rechazar tipos no numéricos.

Nuevo en la versión 3.8.

`math.`**`remainder`**`(` *x* , *y* `)`

Devuelve el resto de estilo IEEE 754 de *x* con respecto a *y* . Para *x finito y y* finito distinto de cero , esta es la diferencia , donde es el número entero más cercano al valor exacto del cociente . Si está exactamente a la mitad de dos enteros consecutivos, se usa el entero *par* más cercano para . El resto por lo tanto siempre satisface `.x - n*ynx / yx / ynr = remainder(x, y)abs(r) <= 0.5 * abs(y)`

Los casos especiales siguen IEEE 754: en particular, es *x para cualquier x* finita y y aumenta para cualquier *x* que no sea NaN . Si el resultado de la operación de resto es cero, ese cero tendrá el mismo signo que *x*. `remainder(x, math.inf)remainder(x, 0)remainder(math.inf, x)ValueError`

En plataformas que utilizan punto flotante binario IEEE 754, el resultado de esta operación siempre se puede representar exactamente: no se introduce ningún error de redondeo.

Nuevo en la versión 3.7.

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

Devuelve *x* con la parte fraccionaria eliminada, dejando la parte entera. Esto redondea hacia 0: `trunc()`es equivalente a para *x*`floor()` positivo y equivalente a para *x* negativo . Si *x* no es un flotante, delega a , que debería devolver un valor.`ceil()``x.__trunc__``Integral`

`math.`**`ulp`**`(`*x*`)`

Devuelve el valor del bit menos significativo del float *x* :

- Si *x* es un NaN (no un número), devuelve *x* .
- Si *x* es negativo, devuelve `ulp(-x)`.
- Si *x* es un infinito positivo, devuelve *x* .
- Si *x* es igual a cero, devuelva el valor flotante representable *desnormalizado* positivo más pequeño (más pequeño que el valor flotante *normalizado* positivo mínimo , `sys.float_info.min`).
- Si *x* es igual al valor flotante positivo más grande representable, devuelve el valor del bit menos significativo de *x* , de modo que el primer valor flotante más pequeño que *x* sea `.x - ulp(x)`
- De lo contrario (*x* es un número finito positivo), devuelva el valor del bit menos significativo de *x* , de modo que el primer flotador más grande que *x* sea `.x + ulp(x)`

ULP significa “Unidad en el Último Lugar”.

Véase también `math.nextafter()`y `sys.float_info.epsilon`.

Nuevo en la versión 3.9.

Tenga en cuenta que `frexp()`y `modf()`tienen un patrón de llamada/retorno diferente al de sus equivalentes en C: toman un solo argumento y devuelven un par de valores, en lugar de devolver su segundo valor de retorno a través de un 'parámetro de salida' (no existe tal cosa en Python).

Para las funciones `ceil()`, `floor()`y , tenga en `modf()`cuenta que *todos* los números de punto flotante de magnitud suficientemente grande son enteros exactos. Los flotantes de Python normalmente no tienen más de 53 bits de precisión (lo mismo que el tipo doble de la plataforma C), en cuyo caso cualquier flotante *x* necesariamente no tiene bits fraccionarios.`abs(x) >= 2**52`

Potencia y funciones logarítmicas

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

Devuelve *e* elevado a la potencia *x* , donde *e* = 2.718281… es la base de los logaritmos naturales. Esto suele ser más preciso que o .`math.e ** x``pow(math.e, x)`

`math.`**`expm1`**`(`*x*`)`

Devuelve *e* elevado a la potencia *x* , menos 1. Aquí *e* es la base de los logaritmos naturales. Para pequeños flotadores *x* , la sustracción en puede resultar en una pérdida significativa de precisión ; la función proporciona una forma de calcular esta cantidad con total precisión:`exp(x) - 1``expm1()`

```
>>> 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
```

Nuevo en la versión 3.2.

`math.`**`log`**`(` *x* `[` , *base* `]` `)`

Con un argumento, devuelva el logaritmo natural de *x* (a la base *e*).

Con dos argumentos, devuelve el logaritmo de *x* a la *base* dada , calculado como `log(x)/log(base)`.

`math.`**`log1p`**`(` *x* `)`

Devuelve el logaritmo natural de *1+x* (base *e*). El resultado se calcula de una manera que es precisa para *x* cerca de cero.

`math.`**`log2`**`(` *x* `)`

Devuelve el logaritmo en base 2 de *x* . Esto suele ser más preciso que `.log(x, 2)`

Nuevo en la versión 3.3.

Ver también: `int.bit_length()`devuelve el número de bits necesarios para representar un número entero en binario, excluyendo el signo y los ceros iniciales.

`math.`**`log10`**`(` *x* `)`

Devuelve el logaritmo en base 10 de *x* . Esto suele ser más preciso que `.log(x, 10)`

`math.`**`pow`**`(` *x* , *y* `)`

Vuelta `x`elevada a la potencia `y`. Los casos excepcionales siguen el Anexo 'F' del estándar C99 en la medida de lo posible. En particular, y siempre devuelve , incluso cuando es un cero o un NaN. Si ambos y son finitos, es negativo y no es un número entero, entonces es indefinido y aumenta `.pow(1.0, x)``pow(x, 0.0)``1.0``x``x``y``x``y``pow(x, y)``ValueError`

A diferencia del `**`operador incorporado, `math.pow()`convierte ambos argumentos a tipo `float`. Use `**`o la `pow()`función incorporada para calcular potencias enteras exactas.

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

Devuelve la raíz cuadrada de *x* .

math.**acos**(*x*)

Devuelve el arcocoseno de *x* , en radianes. El resultado está entre `0`y `pi`.

math.**asin**(*x*)

Devuelve el arco seno de *x* , en radianes. El resultado está entre `-pi/2`y `pi/2`.

math.**atan**(*x*)

Devuelve el arco tangente de *x* , en radianes. El resultado está entre `-pi/2`y `pi/2`.

math.**atan2**(*y* , *x*)

Retorno , en radianes. El resultado está entre y . El vector en el plano desde el origen hasta el punto forma este ángulo con el eje X positivo. El punto de es que conoce los signos de ambas entradas, por lo que puede calcular el cuadrante correcto para el ángulo. Por ejemplo, y son ambos , pero es `.atan(y / x)` `-pi` `pi` `(x, y)` `atan2()` `atan(1)` `atan2(1, 1)` `pi/4` `atan2(-1, -1)` `-3*pi/4`

math.**cos**(*x*)

Devuelve el coseno de *x* radianes.

math.**dist**(*pag* , *q*)

Devuelve la distancia euclidiana entre dos puntos *p* y *q* , cada uno dado como una secuencia (o iterable) de coordenadas. Los dos puntos deben tener la misma dimensión.

Aproximadamente equivalente a:

```
sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))
```

Nuevo en la versión 3.8.

math.**hypot**(`*` *coordenadas*)

Devuelve la norma euclidiana, . Esta es la longitud del vector desde el origen hasta el punto dado por las coordenadas.`sqrt(sum(x**2 for x in coordinates))`

Para un punto bidimensional , esto es equivalente a calcular la hipotenusa de un triángulo rectángulo utilizando el teorema de Pitágoras, .`(x, y)` `sqrt(x*x + y*y)`

Cambiado en la versión 3.8: Soporte agregado para puntos n-dimensionales. Anteriormente, solo se admitía el caso bidimensional.

Modificado en la versión 3.10: Se mejoró la precisión del algoritmo para que el error máximo esté por debajo de 1 ulp (unidad en el último lugar). Más típicamente, el resultado casi siempre se redondea correctamente dentro de 1/2 ulp.

math.**sin**(*x*)

Devuelve el seno de *x* radianes.

Devuelve la tangente de *x* radianes.

Conversión angular

`math.`**`degrees`**(*x*)
Convierta el ángulo *x* de radianes a grados.

`math.`**`radians`**(*x*)
Convierta el ángulo *x* de grados a radianes.

Funciones hiperbólicas

Las funciones hiperbólicas son análogos de las funciones trigonométricas que se basan en hipérbolas en lugar de círculos.

`math.`**`acosh`**(*x*)
Devuelve el coseno hiperbólico inverso de *x* .

`math.`**`asinh`**(*x*)
Devuelve el seno hiperbólico inverso de *x* .

`math.`**`atanh`**(*x*)
Devuelve la tangente hiperbólica inversa de *x* .

`math.`**`cosh`**(*x*)
Devuelve el coseno hiperbólico de *x* .

`math.`**`sinh`**(*x*)
Devuelve el seno hiperbólico de *x* .

`math.`**`tanh`**(*x*)
Devuelve la tangente hiperbólica de *x* .

Funciones especiales

`math.`**`erf`**(*x*)
Devuelve la función de error en *x* .

La `erf()` función se puede utilizar para calcular funciones estadísticas tradicionales, como la distribución normal estándar acumulativa :

```
def phi(x):
    'Cumulative distribution function for the standard normal distribution'
    return (1.0 + erf(x / sqrt(2.0))) / 2.0
```

Nuevo en la versión 3.2.

`math.`**`erfc`**`(`*x*`)`

Devuelve la función de error complementaria en *x* . La función de error complementaria se define como . Se usa para valores grandes de *x* donde una resta de uno causaría una pérdida de significado .`1.0 - erf(x)`

Nuevo en la versión 3.2.

`math.`**`gamma`**`(`*x*`)`

Devuelve la función Gamma en *x* .

Nuevo en la versión 3.2.

`math.`**`lgamma`**`(`*x*`)`

Devuelve el logaritmo natural del valor absoluto de la función Gamma en *x* .

Nuevo en la versión 3.2.

Constantes

`math.`**`pi`**

La constante matemática *π* = 3.141592…, a la precisión disponible.

`math.`**`e`**

La constante matemática *e* = 2.718281…, a la precisión disponible.

`math.`**`tau`**

La constante matemática *τ* = 6.283185…, a la precisión disponible. Tau es una constante circular igual a 2 *π* , la relación entre la circunferencia de un círculo y su radio. Para obtener más información sobre Tau, consulte el video de Vi Hart Pi is (todavía) Wrong y comience a celebrar el día de Tau comiendo el doble de pastel.

Nuevo en la versión 3.6.

`math.`**`inf`**

Un infinito positivo de coma flotante. (Para infinito negativo, use `-math.inf`.) Equivalente a la salida de `float('inf')`.

Nuevo en la versión 3.5.

Un valor de punto flotante "no es un número" (NaN). Equivalente a la salida de `float('nan')`. Debido a los requisitos del estándar IEEE-754 , `math.nan`no `float('nan')`se consideran iguales a ningún otro valor numérico, incluidos ellos mismos. Para verificar si un número es NaN, use la `isnan()`función para probar NaN en lugar de `iso ==`. Ejemplo:

```
>>> import math
>>> math.nan == math.nan
False
>>> float('nan') == float('nan')
False
>>> math.isnan(math.nan)
True
>>> math.isnan(float('nan'))
True
```

Nuevo en la versión 3.5.

Detalle de implementación de CPython: el `math`módulo consiste principalmente en envoltorios delgados alrededor de las funciones de la biblioteca matemática de la plataforma C. El comportamiento en casos excepcionales sigue el Anexo F del estándar C99 cuando corresponda. La implementación actual generará `ValueError`operaciones no válidas como `sqrt(-1.0)`o `log(0.0)` (donde C99 Anexo F recomienda señalar operaciones no válidas o dividir por cero), y `OverflowError`para resultados que se desbordan (por ejemplo, `exp(1000.0)`). No se devolverá un NaN de ninguna de las funciones anteriores a menos que uno o más de los argumentos de entrada fueran un NaN; en ese caso, la mayoría de las funciones devolverán un NaN, pero (nuevamente siguiendo el Anexo F de C99) hay algunas excepciones a esta regla, por ejemplo o `.pow(float('nan'), 0.0)` `hypot(float('nan'), float('inf'))`
Tenga en cuenta que Python no hace ningún esfuerzo por distinguir los NaN de señalización de los NaN silenciosos, y el comportamiento de los NaN de señalización sigue sin especificarse. El comportamiento típico es tratar a todos los NaN como si estuvieran en silencio.

Ver también:

Módulo `cmath`
Versiones de números complejos de muchas de estas funciones.

"