9.2. Fonctions mathématiques — math

9.2.1. Fonctions arithmétiques et de représentation

math.ceil(*x*)

Renvoie la fonction *plafond* de *x*, le plus petit entier plus grand ou égal à *x*. Si *x* est un flottant, délègue à x.__ceil()__, qui doit renvoyer une valeur Integral.

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

Renvoie un flottant contenant la magnitude (valeur absolue) de *x* mais avec le signe de *y*. Sur les plate-formes supportant les zéros signés, copysign(1.0, -0.0) renvoie *-1.0*.

math.fabs(*x*)

Renvoie la valeur absolue de *x*.

math.factorial(*x*)

Renvoie la factorielle de *x*. Lève une ValueError si *x* n’est pas entier ou s’il est négatif.

math.floor(*x*)

Renvoie le plancher de *x*, le plus grand entier plus petit ou égal à *x*. Si *x* n’est pas un flottant, délègue à x.__floor()__, qui doit renvoyer une valeur Integral.

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

Renvoie fmod(x, y), tel que défini par la bibliothèque C de la plate-forme. Notez que l’expression Python x % y peut ne pas renvoyer le même résultat. Le sens du standard C pour fmod(x, y) est d’être exactement (mathématiquement, à une précision infinie) égal à x - n*y pour un entier *n* tel que le résultat a le signe de *x* et une magnitude inférieure à abs(y). L’expression Python x % y renvoie un résultat avec le signe de *y*, et peut ne pas être calculable exactement pour des arguments flottants. Par exemple : fmod(-1e-100, 1e100) est -1e-100, mais le résultat de l’expression Python -1e-100 % 1e100 est 1e100-1e-100, qui ne peut pas être représenté exactement par un flottant et donc qui est arrondi à 1e100. Pour cette raison, la fonction fmod() est généralement préférée quand des flottants sont manipulés, alors que l’expression Python x % y est préféré quand des entiers sont manipulés.

(Source : https://docs.python.org/fr/3.5/library/math.html)

math.frexp(*x*)

Renvoie la mantisse et l’exposant de *x* dans un couple (m, e). *m* est un flottant et *e* est un entier tels que x == m * 2**e exactement. Si *x* vaut zéro, renvoie (0, 0), sinon 0.5 <= abs(m) < 1. Ceci est utilisé pour « extraire » la représentation interne d’un flottant de manière portable.

math.fsum(*iterable*)

Renvoie une somme flottante exacte des valeurs dans l’itérable. Évite la perte de précision en gardant plusieurs sommes partielles intermédiaires :

```
>>>
- >>> 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
```

L’exactitude de cet algorithme dépend des garanties arithmétiques de IEEE-754 et des cas typiques où le mode d’arrondi est *half-even*. Sur certaines versions non Windows, la bibliothèque C sous-jacente utilise une addition par précision étendue et peut occasionnellement effectuer un double-arrondi sur une somme intermédiaire causant la prise d’une mauvaise valeur du bit de poids faible.

Pour de plus amples discussions et deux approches alternatives, voir ASPN cookbook recipes for accurate floating point summation.

math.gcd(*a*, *b*)

Renvoie le plus grand diviseur commun des entiers *a* et *b*. Si soit *a* ou *b* est différent de zéro, la valeur de gcd(a, b) est le plus grand entier positif qui divise à la fois *a* et *b*. gcd(0, 0) renvoie 0.

Nouveau dans la version 3.5.

math.isclose(*a*, *b*, *, *rel_tol=1e-09*, *abs_tol=0.0*)

Renvoie True si les valeurs *a* et *b* sont proches l’une de l’autre, et False sinon.

Déterminer si deux valeurs sont proches se fait à l’aide des tolérances absolues et relatives données en paramètres.

rel_tol est la tolérance relative – c’est la différence maximale permise entre *a* et *b*, relativement à la plus grande valeur de *a* ou de *b*. Par exemple, pour définir une tolérance de 5%,, précisez rel_tol=0.05. La tolérance par défaut est 1e-09, ce qui assure que deux valeurs sont les mêmes à partir de la 9^{e} décimale. *rel_tol* doit être supérieur à zéro.

(Source : https://docs.python.org/fr/3.5/library/math.html)

abs_tol est la tolérance absolue minimale – utile pour les comparaisons proches de zéro. *abs_tol* doit valoir au moins zéro.

Si aucune erreur n’est rencontrée, le résultat sera : abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol).

Les valeurs spécifiques suivantes : NaN, inf, et -inf définies dans la norme IEEE 754 seront manipulées selon les règles du standard IEEE. En particulier, NaN n’est considéré proche d’aucune autre valeur, NaN inclus. inf et -inf ne sont considérés proches que d’eux-mêmes.

Nouveau dans la version 3.5.

Voir aussi

PEP 485 – Une fonction pour tester des égalités approximées

math.isfinite(*x*)

Renvoie True si *n* n’est ni infini, ni NaN, et False sinon. (Notez que 0.0 *est* considéré fini.)

Nouveau dans la version 3.2.

math.isinf(*x*)

Renvoie True si *x* vaut l’infini positif ou négatif, et False sinon.

math.isnan(*x*)

Renvoie True si *x* est NaN (*Not a Number*, ou *Pas un Nombre* en français), et False sinon.

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

Renvoie x * (2**i). C’est essentiellement l’inverse de la fonction frexp().

math.modf(*x*)

Renvoie les parties entières et fractionnelle de *x*. Les deux résultats ont le signe de *x* et sont flottants.

math.trunc(*x*)

Renvoie la valeur Real *x* tronquée en un Integral (habituellement un entier). Délègue à x.__trunc()__.

Notez que les fonctions frexp() et modf() ont un système d’appel différent de leur homologue C : elles prennent un simple argument et renvoient une paire de valeurs au lieu de renvoyer leur seconde valeur de retour dans un “paramètre de sortie” (il n’y a pas de telle possibilité en Python).

(Source : https://docs.python.org/fr/3.5/library/math.html)

Pour les fonctions ceil(), floor(), et modf(), notez que *tous* les nombres flottants de magnitude suffisamment grande sont des entiers exacts. Les flottants de Python n’ont typiquement pas plus de 53 *bits* de précision (tels que le type C double de la plate-forme), en quel cas tout flottant *x* tel que abs(x) >= 2**52 n’a aucun *bit* fractionnel.

9.2.2. Fonctions logarithme et exponentielle

math.exp(*x*)

Renvoie e**x.

math.expm1(*x*)

Renvoie e**x - 1. Pour de petits flottants, la soustraction exp(x) - 1 peut résulter en une perte significative de précision; la fonction expm1() fournit un moyen de calculer cette quantité en précision complète :

```
>>>
- >>> 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
```

Nouveau dans la version 3.2.

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

Avec un argument, renvoie le logarithme naturel de *x* (en base *e*).

Avec deux arguments, renvoie le logarithme de *x* en la *base* donnée, calculé par log(x)/log(base).

math.log1p(*x*)

Renvoie le logarithme naturel de *1+x* (en base *e*). Le résultat est calculé par un moyen qui reste exact pour *x* proche de zéro.

math.log2(*x*)

Renvoie le logarithme en base 2 de *x*. C’est habituellement plus exact que log(x, 2).

Nouveau dans la version 3.3.

Voir aussi

(Source : https://docs.python.org/fr/3.5/library/math.html)

int.bit_length() renvoie le nombre de bits nécessaires pour représenter un entier en binaire, en excluant le signe et les zéros de début.

math.log10(*x*)

Renvoie le logarithme de *x* en base 10. C’est habituellement plus exact que log(x, 10).

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

Renvoie x élevé à la puissance y. Les cas exceptionnels suivent l’annexe “F” du standard C99 autant que possible. En particulier, pow(1.0, x) et pow(x, 0.0) renvoient toujours 1.0, même si x est zéro ou NaN. Si à la fois x *et* y sont finis, x est négatif et y n’est pas entier, alors pow(x, y) est non défini et lève une ValueError.

À l’inverse de l’opérateur interne **, la fonction math.pow() convertit ses deux arguments en float. Utilisez ** ou la primitive pow() pour calculer des puissances exactes d’entiers.

math.sqrt(*x*)

Renvoie la racine carrée de *x*.

9.2.3. Fonctions trigonométriques

math.acos(*x*)

Renvoie l’arc cosinus de *x*, en radians.

math.asin(*x*)

Renvoie l’arc sinus de *x*, en radians.

math.atan(*x*)

Renvoie l’arc tangente de *x*, en radians.

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

Renvoie atan(y / x), en radians. Le résultat est entre -pi et pi. Le vecteur du plan allant de l’origine vers le point (x, y) forme cet angle avec l’axe X positif. L’intérêt de atan2() est que le signe des deux entrées est connu. Donc elle peut calculer le bon quadrant pour l’angle. par exemple atan(1) et atan2(1, 1) donnent tous deux pi/4, mais atan2(-1, -1) donne -3*pi/4.

math.cos(*x*)

Renvoie le cosinus de *x* radians.

(Source : https://docs.python.org/fr/3.5/library/math.html)

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

Renvoie la norme euclidienne sqrt(x*x + y*y). C’est la longueur du vecteur allant de l’origine au point (x, y).

math.sin(*x*)

Renvoie le sinus de*x* radians.

math.tan(*x*)

Renvoie la tangente de *x* radians.

9.2.4. Conversion angulaire

math.degrees(*x*)

Convertit l’angle *x* de radians en degrés.

math.radians(*x*)

Convertit l’ange *x* de degrés en radians.

9.2.5. Fonctions hyperboliques

Hyperbolic functions sont analogues à des fonctions trigonométriques qui sont basés sur des hyperboles au lieu de cercles.

math.acosh(*x*)

Renvoie l’arc cosinus hyperbolique de *x*.

math.asinh(*x*)

Renvoie l’arc sinus hyperbolique de *x*.

math.atanh(*x*)

Renvoie l’arc tangente hyperbolique de *x*.

math.cosh(*x*)

Renvoie le cosinus hyperbolique de *x*.

math.sinh(*x*)

(Source : https://docs.python.org/fr/3.5/library/math.html)

Renvoie le sinus hyperbolique de *x*.

math.tanh(*x*)

Renvoie la tangente hyperbolique de *x*.

9.2.6. Fonctions spéciales

math.erf(*x*)

Renvoie la fonction d’erreur en *x*.

La fonction erf() peut être utilisée pour calculer des fonctions statistiques usuelles telles que la répartition de la loi normale :

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

Nouveau dans la version 3.2.

math.erfc(*x*)

Renvoie la fonction d’erreur complémentaire en *x*. La fonction d’erreur complémentaire est définie par 1.0 - erf(x). C’est utilisé pour de grandes valeurs de *x* où une soustraction d’un causerait une perte de précision.

Nouveau dans la version 3.2.

math.gamma(*x*)

Renvoie la fonction Gamma en *x*.

Nouveau dans la version 3.2.

math.lgamma(*x*)

Renvoie le logarithme naturel de la valeur absolue de la fonction gamma en *x*.

Nouveau dans la version 3.2.

9.2.7. Constantes

math.pi

La constante mathématique π = 3.141592…, à la précision disponile.

math.e

La constante mathématique e = 2.718281…, à la précision disponible.

math.inf

Un flottant positif infini. (Pour un infini négatif, utilisez -math.inf.) Équivalent au résultat de float('inf').

Nouveau dans la version 3.5.

math.nan

Un flottant valant NaN. Équivalent au résultat de float('nan').

Nouveau dans la version 3.5.

Le module math consiste majoritairement en un conteneur pour les fonctions mathématiques de la bibliothèque C de la plate-forme. Le comportement dans les cas spéciaux suit l’annexe “F” du standard C99 quand c’est approprié. L’implémentation actuelle lève une ValueError pour les opérations invalides telles que sqrt(-1.0) ou log(0.0) (où le standard C99 recommande de signaler que l’opération est invalide ou qu’il y a division par zéro), et une OverflowError pour les résultats qui débordent (par exemple exp(1000.0)). *NaN* ne sera renvoyé pour toute les fonctions ci-dessus sauf si au moins un des arguments de la fonction vaut *NaN*. Dans ce cas, la plupart des fonctions renvoient *NaN*, mais (à nouveau, selon l’annexe “F” du standard C99) il y a quelques exceptions à cette règle, par exemple pow(float('nan'), 0.0) ou hypot(float('nan'), float('inf')).

Notez que Python ne fait aucun effort pour distinguer les NaNs signalétiques des NaNs silencieux, et le comportement de signalement des NaNs reste non-spécifié. le comportement typique est de traiter tous les NaNs comme s’ils étaient silencieux.

(Source : https://docs.python.org/fr/3.5/library/math.html)