ماژول math

آخرین به‌روزرسانی: ۰۱ خرداد ۱۴۰۵

ماژول `math` یکی از پرکاربردترین ماژول‌های **کتابخانه استاندارد پایتون** است که دسترسی به توابع ریاضی رایج و ثابت‌های مهم را فراهم می‌کند. این ماژول در واقع یک پوشش نازک روی توابع ریاضی کتابخانه C استاندارد است و بنابراین بسیار سریع است. توابع این ماژول برای کار با اعداد حقیقی (`int` و `float`) طراحی شده‌اند و از اعداد مختلط پشتیبانی نمی‌کنند؛ برای اعداد مختلط باید از ماژول `cmath` استفاده شود.

ماژول `math` توابع خود را در ۸ دسته اصلی سازمان‌دهی کرده است: نظریه اعداد، حساب اعشاری، دستکاری اعداد اعشاری، توان و لگاریتم، جمع و ضرب، تبدیل زاویه، مثلثاتی و هایپربولیک. علاوه بر توابع، ثابت‌های ریاضی مهمی مثل عدد پی و ثابت اویلر هم در این ماژول تعریف شده‌اند.

فهرست مطالب:

برای استفاده از این ماژول، ابتدا باید آن را ایمپورت کنید:

وبسایت شخصی رضا قلعه‌خانی

```
import math
```

ثابت‌های ریاضی

ماژول `math` پنج ثابت ریاضی از پیش تعریف‌شده دارد:

```
>>> import math

>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> math.tau
6.283185307179586
>>> math.inf
inf
>>> -math.inf
-inf
>>> math.nan
nan
```

نکته: همانطور که قبلاً در نوشته معرفی **اعداد اعشاری** صحبت شد، `math.nan` با هیچ مقداری برابر نیست، حتی با خودش! برای بررسی `NaN` همیشه از `math.isnan` استفاده کنید.

```
>>> math.nan == math.nan     # روش غلط
False
>>> math.isnan(math.nan)     # روش درست
True
```

توابع نظریه اعداد

- تابع `factorial` – فاکتوریل عدد صحیح غیرمنفی را برمی‌گرداند. ورودی باید **عدد صحیح** باشد
- تابع `gcd` – بزرگ‌ترین مقسوم‌علیه مشترک اعداد صحیح ورودی را برمی‌گرداند
- تابع `lcm` – کوچک‌ترین مضرب مشترک اعداد صحیح ورودی را برمی‌گرداند

وبسایت شخصی رضا واعظ‌خانی

- تابع `isqrt` – جذر صحیح عدد غیرمنفی را برمی‌گرداند (نتیجه همواره `int` است)
- تابع `comb` و `perm` – ترکیبیات و جایگشت را محاسبه می‌کنند

```
>>> math.factorial(10)
3628800
>>> math.gcd(100, 75, 50)
25
>>> math.lcm(3, 5, 7)
105
>>> math.isqrt(17)
4
>>> math.comb(10, 3)    # C(10,3) = 10!/(3!×7!)
120
>>> math.perm(10, 3)    # P(10,3) = 10!/7!
720
```

توابع حساب اعشاری

- تابع `ceil` – کوچک‌ترین عدد صحیح بزرگ‌تر یا مساوی `x` را برمی‌گرداند (گرد به بالا)
- تابع `floor` – بزرگ‌ترین عدد صحیح کوچک‌تر یا مساوی `x` را برمی‌گرداند (گرد به پایین)
- تابع `trunc` – بخش اعشاری را حذف می‌کند و فقط بخش صحیح را برمی‌گرداند (همیشه به سمت صفر گرد می‌کند)
- تابع `fabs` – قدر مطلق `x` را همیشه به صورت `float` برمی‌گرداند (برخلاف تابع `abs`)
- تابع `modf` – بخش اعشاری و بخش صحیح را به صورت یک جفت برمی‌گرداند
- تابع `fmod` – باقیمانده تقسیم `x` بر `y` را طبق تعریف کتابخانه C محاسبه می‌کند (تفاوت آن با عملگر `%` پایتون در علامت نتیجه است)
- تابع `fma` – عملیات ضرب و جمع ترکیبی را با دقت بیشتر از `(x * y) + z` معمولی محاسبه می‌کند (چون فقط یک بار گرد می‌شود)

وبسایت شخصی رضا شعبانی

```
>>> math.ceil(3.9)
4
>>> math.ceil(-3.2)
-3
>>> math.floor(3.2)
3
>>> math.floor(-3.2)
-4
>>> math.trunc(3.9)
3
>>> math.fabs(-3.14)
3.14
>>> math.fabs(-7)
7.0
>>> math.modf(3.14)
(0.14000000000000012, 3.0)
>>> math.fmod(-10.5, 3.0)
-1.5
>>> -10.5 % 3.0
1.5
>>> math.fma(2.0, 3.0, 4.0)      # (2.0×3.0)+4.0
10.0
```

وبسایت شخصی رضا وثوقانی

توابع دستکاری اعداد اعشاری

- تابع `isclose` – بررسی می‌کند که آیا دو عدد به اندازه کافی به هم نزدیک هستند یا خیر.
- تابع `isfinite` – بررسی می‌کند که آیا مقدار ورودی متناهی است یا خیر
- تابع `isinf` – – بررسی می‌کند که آیا مقدار ورودی نامتناهی است یا خیر
- تابع `isnan` – بررسی می‌کند که آیا مقدار ورودی `nan` است یا خیر
- تابع `copysign` – قدر مطلق آرگومان اول را با علامت آرگومان دوم ترکیب می‌کند:
- تابع `frexp` و `ldexp` – مکمل یکدیگرند و برای کار با نمایش باینری اعداد اعشاری به کار می‌روند

```
>>> math.isclose(0.1 + 0.2, 0.3)   # correct way
True
>>> 0.1 + 0.2 == 0.3               # wrong way
False
>>> math.isclose(1.0, 1.001, rel_tol=0.01)    # relative tolerance 1%
True
>>> math.isfinite(3.14)
True
>>> math.isfinite(math.inf)
False
>>> math.isinf(1e308 * 10)      # overflow → inf
True
>>> math.copysign(3.0, -1.0)
-3.0
>>> math.frexp(8.0)             # 8.0=0.5×2^4
(0.5, 4)
>>> math.ldexp(0.5, 4)          # 0.5×2^4=8.0
8.0
```

تابع `isclose` جایگزینی امن برای عملگر `==` در مقایسه مستقیم اعداد اعشاری است.

نکته مهم: وقتی از تابع `isclose` برای مقایسه با 0.0 استفاده می‌کنید، حتماً `abs_tol` را تعریف کنید. چون `rel_tol` نسبی است و نسبت به صفر معنی ندارد.

```
>>> math.isclose(0.0, 1e-10, abs_tol=1e-9)
True
>>> math.isclose(0.0, 1e-10)
False
```

توابع توان، نمایی و لگاریتمی

- تابع `sqrt` – جذر (ریشه دوم) اعداد مثبت را محاسبه می‌کند
- تابع `cbrt` – ریشه سوم اعداد مثبت را محاسبه می‌کند
- تابع `pow` – آرگومان اول را به توان آرگومان دوم می‌رساند
- تابع `exp` – عدد e را به توان می‌رساند

- تابع exp2 – عدد ۲ را به توان می‌رساند
- تابع expm1 – مقدار e^x - 1 را حساب می‌کند (برای xهای خیلی نزدیک به صفر به کار می‌رود)
- تابع log – لگاریتم عدد را در مبنای دلخواه حساب می‌کند
- تابع log2 – لگاریتم عدد را در مبنای ۲ حساب می‌کند
- تابع log10 – لگاریتم عدد را در مبنای ۱۰ حساب می‌کند
- تابع log1p – مقدار ln(1+x) را حساب می‌کند (برای xهای خیلی نزدیک به صفر به کار می‌رود)

```
>>> math.sqrt(16.0)
4.0
>>> math.cbrt(-8.0)
-2.0
>>> math.pow(2.0, 10)
1024.0
>>> math.exp(1)             # e^1
2.718281828459045
>>> math.exp2(10)           # 2^10
1024.0
>>> math.expm1(1e-10)       # e^x - 1
1.00000000005e-10
>>> math.log(math.e)
1.0
>>> math.log2(1024)
10.0
>>> math.log10(1000)
2.9999999999999996
>>> math.log1p(1e-10)       # ln(1+x)
1e-10
```

نکته: دقت کنید که نتیجه محاسبه log2(x) از نتیجه log(x, 2) دقیق‌تر است. توابع log10 و expm1 و log1p هم نسبت به محاسبات دستی مقادیر آنها، دقت بهتری دارند.

توابع جمع و ضرب

- تابع fsum – جمع دقیق اعداد اعشاری یک ایتریبل را برمی‌گرداند
- تابع prod – ضرب تمام عناصر یک ایتریبل را برمی‌گرداند

- تابع `hypot` – نُرم اقلیدسی (فاصله ریشه مجموع مربعات) را محاسبه می‌کند
- تابع `dist` – فاصله اقلیدسی بین دو نقطه را محاسبه می‌کند
- تابع `sumprod` – مجموع حاصل‌ضرب‌های متناظر دو ایتریبل را با دقت بالا برمی‌گرداند

```
>>> sum([0.1] * 10)                # not precise
0.9999999999999999
>>> math.fsum([0.1] * 10)          # precise
1.0
>>> math.prod([1, 2, 3, 4, 5])
120
>>> math.prod([2, 3, 4], start=10)  # start from 10
240
>>> math.hypot(3, 4)               # √(3²+4²)=√25
5.0
>>> math.dist([0, 0], [3, 4])
5.0
>>> math.sumprod([1, 2, 3], [4, 5, 6])      # (1×4)+(2×5)+(3×6)=32
32
```

توابع تبدیل زاویه

تمام توابع مثلثاتی ماژول `math` با رادیان کار می‌کنند.

- تابع `degrees` – تبدیل رادیان به درجه را انجام می‌دهد
- تابع `radians` – تبدیل درجه به رادیان را انجام می‌دهد

```
>>> math.degrees(math.pi)
180.0
>>> math.degrees(math.pi / 2)
90.0
>>> math.radians(180)
3.141592653589793
```

وبسایت شخصی رضا قلعه‌خانی

توابع مثلثاتی

دقت کنید که همه توابع مثلثاتی در ماژول math، ورودی را به رادیان می‌گیرند:

```
>>> math.sin(math.pi/2)        # sin(90°) = 1
1.0
>>> math.cos(0)                  # cos(0°) = 1
1.0
>>> math.tan(math.pi/4)        # tan(45°) = 1
0.9999999999999999
>>> math.asin(1.0)               # arcsin(1) = π/2
1.5707963267948966
>>> math.acos(0.0)               # arccos(0) = π/2
1.5707963267948966
>>> math.atan(1.0)               # arctan(1) = π/4
0.7853981633974483
```

توابع هایپربولیک

ماژول math در پایتون از توابع مثلثاتی هایپربولیک به صورت کامل پشتیبانی می‌کند:

```
>>> math.sinh(0)
0.0
>>> math.cosh(1)
1.5430806348152437
>>> math.tanh(1)
0.7615941559557649
>>> math.asinh(1.1752011936438014)
1.0
>>> math.acosh(1.5430806348152437)
1.0
>>> math.atanh(0.7615941559557649)
0.9999999999999999
```

توابع ویژه

توابع دیگری مانند `erf` و `erfc` و ... در ماژول `math` وجود دارند که در محاسبات آمار و احتمال می‌توانند به کار گرفته شوند.

مثال واقعی از ماژول `math`

در یک سناریوی فرضی، شما سیستمی برای تحلیل هندسی و آماری می‌نویسید که شامل محاسبات مثلثاتی، آمار توصیفی و بهینه‌سازی ترکیبیاتی است.

کدهای این مثال به همراه خروجی نهایی را می‌توانید از **اینجا** دانلود کنید.

وبسایت شخصی رضا قلعه‌خانی

سوالات متداول

1. تفاوت `math.sqrt` با `x ** 0.5` چیست؟

هر دو جذر عدد را محاسبه می‌کنند، اما `math.sqrt` سریع‌تر است چون مستقیماً تابع C را فراخوانی می‌کند و همیشه عدد اعشاری برمی‌گرداند. `x ** 0.5` کمی کندتر است ولی روی اعداد مختلط هم کار می‌کند.

---

2. چرا از `math.isclose` به جای `==` برای اعداد اعشاری استفاده کنیم؟

اعداد اعشاری به دلیل نمایش باینری دارای خطاهای گرد کردن هستند. `math.isclose` بر اساس تلرانس نسبی و مطلق مقایسه می‌کند و برای مقایسه امن اعداد اعشاری طراحی شده است. برای مقایسه با صفر، حتماً پارامتر `abs_tol` را تعیین کنید.

---

3. تفاوت `math.fsum` با تابع `sum` پیش‌ساخته چیست؟

تابع `sum` به ترتیب جمع می‌کند و خطاهای گرد کردن تجمع می‌یابند. `math.fsum` از الگوریتمی استفاده می‌کند که چند جمع جزئی واسطه را نگه می‌دارد و در پایان آن‌ها را ترکیب می‌کند تا خطا به حداقل برسد. برای لیست‌های کوچک تفاوت عملی کمی دارند، اما برای لیست‌های بزرگ یا وقتی دقت اهمیت دارد، `fsum` انتخاب درست‌تری است.

---

4. آیا ماژول `math` پایتون برای اعداد مختلط کار می‌کند؟

خیر. توابع ماژول `math` فقط برای اعداد حقیقی (`int` و `float`) طراحی شده‌اند. اگر آرگومانی از محدوده خارج شود، خطای `ValueError` می‌دهد. برای کار با اعداد مختلط از ماژول `cmath` استفاده کنید که نسخه‌های مختلط همین توابع را دارد.

---

5. تفاوت `math.pow(x, y)` با عملگر `**` چیست؟

تابع `math.pow(x, y)` هر دو آرگومان را به `float` تبدیل می‌کند و همیشه `float` برمی‌گرداند. عملگر `**` نوع داده را حفظ می‌کند. برای توان‌های صحیح دقیق، از عملگر `**` یا تابع پیش‌ساخته `pow` استفاده کنید. برای توان‌های اعشاری و زمانی که `float` مطلوب است، `math.pow` مناسب‌تر است.

جهت کسب اطلاعات بیشتر می‌توانید به **مستندات رسمی پایتون برای ماژول `math`** مراجعه کنید.

دسته: کتابخانه استاندارد - پایتون - دانش‌نامه برنامه‌نویسی

وبسایت شخصی رضا قهرمانی

وبسایت شخصی رضا قلعه‌خانی