عدد اعشاری (float)

آخرین به‌روزرسانی: ۰۱ خرداد ۱۴۰۵

نوع داده عدد اعشاری (float) در پایتون برای نمایش اعداد اعشاری استفاده می‌شود. این نوع داده بر پایه استاندارد IEEE 754 پیاده‌سازی شده که معادل نوع double در زبان C است و از ۶۴ بیت حافظه استفاده می‌کند: ۱ بیت علامت، ۱۱ بیت توان و ۵۲ بیت مانتیس. نوع float در پایتون علاوه بر تمام عملیات حسابی معمول، متدهای اختصاصی مفیدی دارد. یکی از مهم‌ترین ویژگی‌هایی که هر برنامه‌نویس پایتون باید بداند این است که اعداد اعشاری دقت محدود دارند و برخی محاسبات به نتایجی که از نظر ریاضی دقیق نیستند منجر می‌شوند؛ درک این ویژگی برای نوشتن کد صحیح حیاتی است.

فهرست مطالب:

ایجاد عدد اعشاری در پایتون

داده‌هایی از نوع عدد اعشاری در پایتون را می‌توان به روش‌های مختلفی ساخت.

۱) نوشتار مستقیم:

```
>>> x = 3.14
>>> y = -0.5
```

۲) نوشتار علمی:

```
>>> speed_of_light = 2.998e8       # 2.998 × 10^8
>>> electron_mass  = 9.109e-31     # 9.109 × 10^-31
>>> avogadro       = 6.022E23      # 6.022 × 10^23

>>> speed_of_light
299800000.0
>>> electron_mass
9.109e-31
```

۳) تابع `float`: سازنده نوع داده عدد اعشاری در پایتون، تابع `float` است:

```
float(x=0.0)
```

مقدار بازگشتی تابع `float` همیشه یک عدد اعشاری از نوع `float` است.

راهنمای جامع تابع `float`

در مورد تابع `float` که سازنده یا Constructor برای عدد اعشاری در پایتون است بیشتر بدانید: **آشنایی با تابع `float` در پایتون**

نکته: از نسخه 3.6 به بعد پایتون، می‌توان برای خوانایی بهتر از کاراکتر زیرخط (_) به عنوان جداکننده استفاده کرد.

```
>>> pi_approx   = 3.141_592_653
>>> pi_approx
3.141592653
```

عملیات پایه روی اعداد اعشاری

در ادامه، تعدادی مثال از عملیات روی اعداد اعشاری را با هم مرور خواهیم کرد.

عملیات حسابی: پایتون تمام عملیات حسابی استاندارد را برای نوع داده `float` پشتیبانی می‌کند. وقتی یک `int` و یک `float` با هم در یک عملیات شرکت کنند، پایتون به صورت خودکار `int` را به `float` تبدیل می‌کند:

```
>>> a, b = 7.5, 2.0

>>> a + b          # جمع
9.5
>>> a - b          # تفریق
5.5
>>> a * b          # ضرب
15.0
>>> a / b          # تقسیم عادی
3.75
>>> a // b         # تقسیم صحیح
3.0
>>> a % b          # باقیمانده
1.5
>>> a ** b         # توان
56.25
>>> abs(-a)        # قدر مطلق
7.5
>>> 5 + 2.0        # int + float → float (widening)
7.0
```

توابع گرد کردن و برش: در ماژول `math` از **کتابخانه استاندارد پایتون** نیز، توابعی برای عملیات روی اعداد اعشاری وجود دارد.

```
>>> import math

>>> round(3.14159, 2)
3.14
>>> round(3.5)
4
>>> round(2.5)
2
>>> math.floor(3.7)
3
>>> math.ceil(3.2)
4
>>> math.trunc(3.9)
3
```

دقت اعداد اعشاری در پایتون

نوع داده `float` در پایتون بر اساس `double` در زبان C پیاده‌سازی شده است. دقت آن حدود ۱۵–۱۷ رقم معنی‌دار اعشاری است. برای اطلاع از مشخصات دقیق روی سیستم خود می‌توانید از ماژول `sys` استفاده کنید. ثوابت مختلف این ماژول که در مثال زیر آورده شده، به ترتیب بزرگ‌ترین عدد اعشاری قابل نمایش، کوچک‌ترین عدد اعشاری قابل نمایش، کوچک‌ترین تفاوت قابل تشخیص و تعداد ارقام معنی‌دار اعشاری را نشان می‌دهند:

```
>>> import sys
>>> sys.float_info.max
1.7976931348623157e+308

>>> sys.float_info.min
2.2250738585072014e-308

>>> sys.float_info.epsilon
2.220446049250313e-16

>>> sys.float_info.dig
15
```

مهم‌ترین ویژگی که باید در کار با `float` در پایتون بدانید، خطاهای نمایش باینری است. این رفتار ذاتی استاندارد IEEE 754 است، نه باگ پایتون! چون کسرهایی مثل 0.1 نمایش دقیق باینری ندارند.

```
>>> 0.1 + 0.2
0.30000000000000004

>>> 0.1 + 0.2 == 0.3
False

>>> 1.1 + 2.2
3.3000000000000003
```

راه‌حل‌هایی برای غلبه بر این نوع خطاها در پایتون تعبیه شده است:

۱) مقایسه با تِلُرانس: می‌توانید از تابع `isclose` ماژول `math` برای مقایسه با تلرانس قابل قبول استفاده کنید. این تابع برای مقایسه میزان نزدیکی دو مقدار با یکدیگر است.

```
>>> import math

>>> math.isclose(0.1 + 0.2, 0.3)
True

>>> math.isclose(0.1 + 0.2, 0.3, rel_tol=1e-9)
True
```

۲) استفاده از کلاس `Decimal`: در محاسبات مالی که دقت اعشاری حائز اهمیت است، کلاس `Decimal` از ماژول `decimal` می‌تواند راه‌گشا باشد.

```
from decimal import Decimal, ROUND_HALF_UP

>>> Decimal("0.1") + Decimal("0.2")
Decimal('0.3')

>>> Decimal("0.1") + Decimal("0.2") == Decimal("0.3")
True

>>> price = Decimal("19.995")
>>> price.quantize(Decimal("0.01"), rounding=ROUND_HALF_UP)
Decimal('20.00')
```

۳) استفاده از کلاس Fraction: برای دقت کامل در محاسبات اعشاری، می‌توانید از کلاس `Fraction` ماژول `fractions` نیز استفاده کنید.

```
from fractions import Fraction

>>> Fraction(1, 10) + Fraction(2, 10)
Fraction(3, 10)

>>> float(Fraction(1, 10) + Fraction(2, 10))
0.3
```

متدهای پرکاربرد اعداد اعشاری

نوع داده `float` در پایتون متدهای اختصاصی مفیدی دارد که در کار با اعداد اعشاری در سطح پیشرفته‌تر کاربرد دارند.

متد is_integer: بررسی می‌کند که آیا عدد اعشاری یک مقدار صحیح را نمایش می‌دهد یا خیر (یعنی بخش اعشاری آن صفر است) و مقدار `True` یا `False` برمی‌گرداند. به عنوان مثال، می‌توان از این متد برای اعتبارسنجی اینکه آیا نتیجه یک محاسبه اعشاری بدون باقیمانده است یا خیر استفاده کرد.

```
>>> (3.0).is_integer()
True

>>> def is_divisible(a, b):
...     return (a / b).is_integer()

>>> is_divisible(10, 2)
True

>>> is_divisible(10, 3)
False
```

متد as_integer_ratio: نسبت دقیق معادل عدد float را به صورت یک جفت اعداد صحیح (صورت و مخرج) برمی‌گرداند. این نسبت به صورت کوچک‌ترین مقدار ممکن با مخرج مثبت است و دقیقاً برابر مقدار اصلی float است. تبدیل دقیق عدد اعشاری به Fraction بدون خطای نمایش می‌تواند یکی از کاربردهای این متد باشد.

```
>>> (0.25).as_integer_ratio()
(1, 4)

>>> (-1.5).as_integer_ratio()
(-3, 2)
```

متد hex: نمایش هگزادسیمال دقیق عدد float را به صورت استرینگ برمی‌گرداند. این نمایش کاملاً دقیق است و هیچ خطای گردکردنی ندارد:

```
>>> (3.14).hex()
'0x1.91eb851eb851fp+1'

>>> (0.1).hex()
'0x1.999999999999ap-4'
```

متد fromhex: برعکس متد hex عمل می‌کند: یک استرینگ هگزادسیمال را به عدد اعشاری تبدیل می‌کند. فاصله‌های ابتدا و انتها نادیده گرفته می‌شوند و داده ورودی به بزرگی و کوچکی حروف، حساس است:

```
>>> float.fromhex('0x1.91eb851eb851fp+1')
3.14

>>> float.fromhex('0x1.999999999999ap-4')
0.1
```

کاربرد اصلی متدهای hex و fromhex در موقعیت‌هایی است که نیاز دارید مقدار float را بدون هرگونه خطای گردکردنی، ذخیره کنید یا انتقال دهید.

```
>>> original = 3.141592653589793
>>> encoded  = original.hex()
>>> decoded  = float.fromhex(encoded)
>>> original == decoded
True
```

متد from_number: این متد که از نسخه 3.14 پایتون معرفی شده، یک عدد دلخواه را به float تبدیل می‌کند. برخلاف تابع float، این متد استرینگ‌ها را نمی‌پذیرد و صرفاً برای تبدیل انواع عددی طراحی شده است:

```
>>> float.from_number(42)
42.0

>>> float.from_number(3.14)
3.14

>>> float.from_number(True)
1.0
```

این متد در ساخت زیرکلاس‌هایی از float که باید رفتار تبدیل را کنترل کنند بیشترین کاربرد را دارد.

مثال واقعی از کاربرد اعداد اعشاری

در یک سناریوی فرضی، شما سیستمی برای محاسبه فاکتور فروش با اعمال تخفیف، مالیات و گزارش‌گیری می‌نویسید. استفاده صحیح از دقت اعشاری در محاسبات مالی حیاتی است. در این مثال از کلاس Decimal برای گردکردن صحیح مبالغ مالی، تابع math.fsum برای جمع دقیق لیست اعداد، متد is_integer برای بررسی وضعیت مقادیر و متد as_integer_ratio برای نمایش ماهیت باینری float استفاده شده است.

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

مقادیر ویژه نوع float در پایتون

نوع داده float در پایتون، سه مقدار ویژه دارد که در محاسبات علمی و مدیریت خطا اهمیت دارند: مثبت بی‌نهایت، منفی بی‌نهایت و نامعین (مبهم).

```
>>> pos_inf = float("inf")
>>> neg_inf = float("-inf")
>>> nan     = float("nan")

>>> pos_inf + 1000
inf
>>> pos_inf * 2
inf
>>> pos_inf * -1
-inf
>>> pos_inf - pos_inf
nan
```

نکته طلایی: هرگز از `==` برای بررسی nan استفاده نکنید! چون `nan == nan` همیشه False است.

مقایسه float با int و Decimal

همانطور که قبلاً در نوشته نوع داده عدد صحیح (int) هم توضیح داده شد، نوع int دقت کامل دارد اما نوع داده float دقت محدودی دارد (حدود۱۵-۱۷ رقم معنی‌دار) که می‌تواند به خطاهای ظریف منجر شود.

ویژگی	float	int	Decimal
دقت	۱۵–۱۷ رقم معنی‌دار	نامحدود	تعریف‌شده توسط کاربر
سرعت	بسیار سریع	کند برای اعداد بزرگ	کندتر از float
مصرف حافظه	ثابت ۶۴ بیت	پویا	بیشتر از float
دقت اعشاری	خطاهای نمایش باینری	بدون خطا	دقیق
کاربرد	علمی و مهندسی	شمارش و اندیس	مالی و حسابداری

سوالات متداول

1. تفاوت `round` پایتون با گرد کردن معمولی چیست؟

پایتون از Banker’s Rounding یا گرد کردن به زوج استفاده می‌کند: وقتی عدد دقیقاً در میان دو مقدار باشد (مثل 2.5 یا 3.5)، به نزدیک‌ترین عدد زوج گرد می‌شود. بنابراین `round(0.5)` مقدار 0 و `round(1.5)` مقدار 2 را برمی‌گرداند. این روش خطاهای تجمعی را در محاسبات آماری به حداقل می‌رساند.

---

2. چه زمانی متد `hex` مفید است؟

متد `float.hex` زمانی مفید است که بخواهید مقدار دقیق یک `float` را بدون هیچ خطای گردکردنی ذخیره، منتقل یا دیباگ کنید. نمایش هگزادسیمال دقیقاً همان بیت‌هایی را نشان می‌دهد که در حافظه ذخیره شده‌اند. `float.fromhex` هم تضمین می‌کند که مقدار بازیابی‌شده با اصلی کاملاً برابر است.

---

3. تفاوت `math.floor` با تابع `int` چیست؟

تابع `int` بخش اعشاری را به سمت صفر حذف می‌کند. تابع `math.floor` عدد را به سمت بی‌نهایت منفی گرد می‌کند. برای اعداد مثبت، نتیجه یکسان است اما برای اعداد منفی تفاوت دارند.

---

4. چه زمانی به جای `float` از `Decimal` استفاده کنیم؟

همیشه در محاسبات مالی، حسابداری و بانکی از `Decimal` استفاده کنید. هر جا که دقت اعشاری دقیق اهمیت دارد و خطاهای کوچک مثل 0.000000000000001 قابل قبول نیستند، `Decimal` انتخاب درست است. برای محاسبات علمی و مهندسی که دقت ۱۵ رقم معنی‌دار کافی است و سرعت اهمیت دارد، `float` مناسب‌تر است.

جهت کسب اطلاعات بیشتر می‌توانید به **مستندات رسمی پایتون برای نوع داده عدد اعشاری (`float`)** مراجعه کنید.

دسته: انواع داده - پایتون - دانش‌نامه برنامه‌نویسی