# عدد صحیح (int)

صفحه اصلی / دانش‌نامه / پایتون / انواع داده

آخرین به‌روزرسانی: ۲۶ اردیبهشت ۱۴۰۵

نوع داده عدد صحیح (int) در پایتون یکی از بنیادی‌ترین و پرکاربردترین انواع داده این زبان برنامه‌نویسی است. اعداد صحیح (integer) مقادیری هستند که بخش اعشاری ندارند و می‌توانند مثبت، منفی یا صفر باشند. یکی از مهم‌ترین ویژگی‌های int در پایتون، دقت نامحدود آن است؛ به این معنا که برخلاف بسیاری از زبان‌های دیگر، هیچ حد بالایی برای بزرگی اعداد صحیح وجود ندارد و پایتون می‌تواند اعداد به هر اندازه‌ای را نگه دارد. int در پایتون یک کلاس کامل است که علاوه بر عملیات عددی، متدهای اختصاصی مفیدی نیز دارد.

## فهرست مطالب:





## ایجاد عدد صحیح در پایتون

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

**۱) نوشتار مستقیم:**

```
>>> x = 42
>>> y = -7
>>> z = 0
>>> type(x)
<class 'int'>
```

**۲) نوشتار در مبناهای مختلف:**

```
>>> binary = 0b1010       # binary
>>> binary
10

>>> octal = 0o17          # octal
>>> octal
15

>>> hexadecimal = 0xFF    # hexadecimal
>>> hexadecimal
255
```

**۳) تابع int:** سازنده نوع داده عدد صحیح در پایتون، تابع `int` است:

```
int(x)
int(x, base=10)
```

مقدار بازگشتی تابع `int` همیشه یک عدد صحیح از نوع `int` است.

### راهنمای جامع تابع int

در مورد تابع `int` که سازنده یا Constructor برای اعداد صحیح در پایتون است بیشتر بدانید: **آشنایی با تابع int در پایتون**

## عملیات پایه روی عدد صحیح

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

**عملیات حسابی:** پایتون تمام عملیات حسابی استاندارد را برای نوع داده `int` پشتیبانی می‌کند:

```
>>> a, b = 17, 5

>>> a + b      # جمع
22
>>> a - b      # تفریق
12
>>> a * b      # ضرب
85
>>> a / b      # نتیجه تقسیم عادی، همیشه عدد اعشاری است
3.4
>>> a // b     # نتیجه تقسیم صحیح، همیشه عدد صحیح است
3
>>> a % b      # باقیمانده
2
>>> a ** b     # توان
1419857
>>> abs(-a)    # قدر مطلق
17
```

**نکته جالب:** نتیجه تقسیم صحیح همیشه به سمت منفی بی‌نهایت گرد می‌شود، نه به سمت صفر:

```
>>> 7 // 2
3
>>> -7 // 2
-4
```

**عملیات بیتی:** عملیات بیتی فقط برای اعداد صحیح تعریف شده‌اند و در برنامه‌نویسی سطح پایین، رمزنگاری و پرچم‌های باینری (`binary flags`) کاربرد دارند:

```
>>> x = 0b1010  # 10
>>> y = 0b1100  # 12

>>> x & y       # Bitwise AND
8               # 0b1000

>>> x | y       # Bitwise OR
14              # 0b1110

>>> x ^ y       # Bitwise XOR
6               # 0b0110

>>> ~x          # Bitwise NOT
-11             # -(x+1)
```

**تبدیل اعداد صحیح به مبناهای دیگر:** پایتون توابع پیش‌ساخته‌ای برای نمایش اعداد صحیح در مبناهای رایج دارد:

```
>>> n = 255

>>> bin(n)      # binary
'0b11111111'

>>> oct(n)      # octal
'0o377'

>>> hex(n)      # hexadecimal
'0xff'
```

## متدهای پرکاربرد عدد صحیح

نوع `int` در پایتون متدهای اختصاصی مفیدی دارد که در کار با داده‌های باینری و محاسبات سطح پایین به کار می‌آیند.

**متد `bit_length`:** تعداد بیت‌های لازم برای نمایش عدد در مبنای ۲ را برمی‌گرداند (بدون علامت و صفرهای پیشرو):

```
>>> n = 37
>>> bin(n)
'0b100101'
>>> n.bit_length()
6
>>> (-37).bit_length()
6
>>> (0).bit_length()
0
>>> (1).bit_length()
1
>>> (255).bit_length()
8
```

کاربرد رایج این متد در تعیین حداقل تعداد بایت‌های لازم برای ذخیره یک عدد است:

```
>>> n = 1000
>>> bytes_needed = (n.bit_length() + 7) // 8
>>> bytes_needed
2
```

**متد bit_count:** تعداد بیت‌های ۱ در نمایش باینری قدر مطلق عدد را برمی‌گرداند (به نام «population count» یا وزن هَمینگ نیز شناخته می‌شود). این متد از پایتون 3.10 اضافه شده است:

```
>>> n = 19
>>> bin(n)
'0b10011'
>>> n.bit_count()
3
```

در نمایش باینری عدد ۱۹، تعداد ۳ بیت دارای مقدار 1 وجود دارد. پس خروجی متد bit_count روی عدد ۱۹، عدد ۳ را بر می‌گرداند.

**متد to_bytes:** این متد، عدد صحیح را به آرایه‌ای از بایت‌ها تبدیل می‌کند:

```
>>> (1024).to_bytes(2, byteorder='big')
b'\x04\x00'

>>> (1024).to_bytes(4, byteorder='little')
b'\x00\x04\x00\x00'

>>> (-1024).to_bytes(4, byteorder='big', signed=True)
b'\xff\xff\xfc\x00'
```

**متد from_bytes:** برعکس متد to_bytes عمل می‌کند و یک آرایه بایت را به عدد صحیح تبدیل می‌کند:

```
>>> int.from_bytes(b'\x04\x00', byteorder='big')
1024

>>> int.from_bytes(b'\x00\x04', byteorder='little')
1024

>>> int.from_bytes(b'\xff\xff\xfc\x00', byteorder='big', signed=True)
-1024

>>> int.from_bytes([255, 0, 0], byteorder='big')
16711680
```

این متد در تجزیه پروتکل‌های شبکه، خواندن فرمت‌های باینری فایل و ارتباط با سخت‌افزار کاربرد گسترده‌ای دارد.

**متد as_integer_ratio:** این متد، نسبت معادل عدد صحیح را به صورت یک جفت (صورت و مخرج) برمی‌گرداند. برای اعداد صحیح، مخرج همیشه 1 است:

```
>>> (42).as_integer_ratio()
(42, 1)

>>> (-7).as_integer_ratio()
(-7, 1)

>>> (6.5).as_integer_ratio()
(13, 2)
```

این متد بیشتر برای یکپارچگی با نوع داده `float` و `Fraction` مفید است، چون هر دو همین این متد را دارند.

## مثال واقعی از کاربرد عدد صحیح

در یک سناریوی فرضی، شما می‌خواهید یک سیستم ساده برای مدیریت موجودی انبار بنویسید که شامل عملیات عددی، بررسی شرایط مرزی و تبدیل واحدها باشد:

```python
inventory = {
    "Laptop": {"stock": 47, "price": 45_000_000, "low_stock": 10},
    "Mouse":  {"stock": 3,  "price": 850_000,    "low_stock": 20},
    "Keyboard":{"stock": 28, "price": 2_200_000,  "low_stock": 15},
    "Headset":{"stock": 0,  "price": 3_200_000,  "low_stock": 5},
}

discount_rate = 15

def apply_discount(price, rate):
    return price * (100 - rate) // 100

def stock_status(item, data):
    if data["stock"] == 0:
        return "In-Stock"
    elif data["stock"] < data["low_stock"]:
        return "Low-Stock"
    else:
        return "Out-Stock"

total_value = 0
print("=== Stock Report ===\n")

for item, data in inventory.items():
    discounted = apply_discount(data["price"], discount_rate)
    value = data["stock"] * discounted
    total_value += value
    status = stock_status(item, data)
    print(f"{item}: {data['stock']} | Discount Price: {discounted:,} | Status:
{status}")

print(f"\nTotal Value: {total_value:,} Rials")

print(f"\nRequired Bits for Total Value: {total_value.bit_length()} Bits")
```

```
print(f"Required Bytes for Total Value: {(total_value.bit_length() + 7) // 8}
Bytes")
```

نمونه خروجی این کد به صورت زیر خواهد بود:

```
=== Stock Report ===

Laptop: 47 | Discount Price: 38,250,000 | Status: Out-Stock
Mouse: 3 | Discount Price: 722,500 | Status: Low-Stock
Keyboard: 28 | Discount Price: 1,870,000 | Status: Out-Stock
Headset: 0 | Discount Price: 2,720,000 | Status: In-Stock

Total Value: 1,852,277,500 Rials

Required Bits for Total Value: 31 Bits
Required Bytes for Total Value: 4 Bytes
```

در این مثال، از تقسیم صحیح // برای محاسبه قیمت تخفیف‌دار بدون ورود به float استفاده شد که نتیجه‌ای دقیق‌تر در محاسبات مالی می‌دهد. همچنین متد bit_length برای تحلیل فضای ذخیره‌سازی مورد نیاز به کار رفت.

## مقایسه int با float و Decimal

نوع int دقت کامل دارد، اما نوع داده float دقت محدودی دارد که می‌تواند به خطاهای ظریف منجر شود:

```
>>> 0.1 + 0.2 == 0.3
False

>>> 1 + 2 == 3
True
```

وقتی دقت اعشاری حیاتی است (مثل محاسبات مالی)، از کلاس Decimal ماژول decimal استفاده می‌شود:

```
from decimal import Decimal

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

## تفاوت int پایتون با C و Java

تفاوت اصلی int در پایتون با زبان‌هایی مثل C و Java در اندازه، مدیریت حافظه و رفتار هنگام سرریز (overflow) است. در پایتون، int دارای اندازه نامحدود است؛ یعنی بسته به مقدار عدد، به‌صورت خودکار بزرگ‌تر می‌شود و محدود به ۳۲ یا ۶۴ بیت نیست. بنابراین سرریز عددی (overflow) به شکل کلاسیک وجود ندارد و فقط مصرف حافظه بیشتر می‌شود.

اما در زبان‌هایی مثل C و Java اگر مقدار عدد از محدوده خارج شود، overflow رخ می‌دهد و مقدار به صورت دورانی یا خطای منطقی ذخیره می‌شود. در Java اگر به عدد بزرگ‌تر نیاز باشد باید از کلاس BigInteger استفاده شود، در حالی که در پایتون این قابلیت به‌صورت پیش‌فرض در int وجود دارد.

## سوالات متداول

### 1. تفاوت تابع `int` و `round` چیست؟

تابع `int` بخش اعشاری را بدون گرد کردن حذف (`truncate`) می‌کند و همیشه به سمت صفر می‌رود. اما تابع `round` عدد را به نزدیک‌ترین مقدار گرد می‌کند. برای گرد کردن به پایین، از متد `floor` و برای گرد کردن به بالا از متد `ceil` ماژول `math` استفاده کنید.

---

### 2. آیا می‌توانم `int` را با `float` مقایسه کنم؟

بله. پایتون به طور خودکار مقدار دقیق هر دو عدد را مقایسه می‌کند.

---

### 3. عملگر تقسیم صحیح // چه تفاوتی با تقسیم عادی / دارد؟

عملگر / همیشه یک عدد اعشاری برمی‌گرداند، حتی اگر نتیجه عددی صحیح باشد. عملگر // تقسیم صحیح انجام می‌دهد و نتیجه را به سمت منفی بی‌نهایت گرد می‌کند.

---

### 4. آیا `int` در پایتون همان `bool` است؟

نه، ولی `bool` یک زیرنوع از `int` است. یعنی هر مقدار `bool` یک `int` هم هست، اما هر `int` لزوماً `bool` نیست. `True` معادل 1 و `False` معادل 0 است و می‌توان آن‌ها را در محاسبات عددی استفاده کرد.

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

---

دسته: انواع داده - پایتون - دانش‌نامه برنامه‌نویسی