TÁC GIẢ: 8 SYNC

CỘNG ĐỒNG

- Kevin Nguyễn
- Nhóm Chia Sẻ Công Nghệ
- Nhóm BlockChain
- Tiktok: 8 Sync
- Youtube: 8 Sync Dev
- Zalo

KHÓA HỌC:

- Fullstack Python
- Fullstack Nextjs
- Fullstack Android-IOS

Tài liệu sẽ được cập nhật định kì và thông báo trong group nên các bạn chú ý nhen .

PYTHON - SỐ

Python có sẵn hỗ trợ để lưu trữ và xử lý dữ liệu số (Python Numbers). Hầu hết thời gian bạn làm việc với số trong hầu hết các ứng dụng Python. Rõ ràng, bất kỳ ứng dụng máy tính nào cũng đều xử lý số. Hướng dẫn này sẽ thảo luận về các loại số Python khác nhau và các thuộc tính của chúng.

PYTHON - CÁC LOẠI SỐ

Có ba loại số được tích hợp sẵn trong Python:

1. Số nguyên (int)
2. Số dấu chấm động (float)
3. Số phức

Python cũng có một loại dữ liệu Boolean được tích hợp sẵn gọi là bool. Nó có thể được coi như một loại con của kiểu int, vì hai giá trị có thể có của nó là True và False biểu diễn cho các số nguyên 1 và 0 tương ứng.

PYTHON – SỐ NGUYÊN

Trong Python, bất kỳ số nào mà không có khả năng lưu trữ phần thập phân là số nguyên. (Lưu ý rằng nếu phần thập phân trong một số là 0, điều đó không có nghĩa là nó là một số nguyên. Ví dụ, một số 10.0 không phải là một số nguyên, nó là một số dấu chấm động với phần thập phân là 0 có giá trị số là 10.) Một số nguyên có thể là số 0, số nguyên dương hoặc số nguyên âm. Ví dụ, 1234, 0, -55 đều đại diện cho các số nguyên trong Python.

Có ba cách để tạo một đối tượng số nguyên. Bằng cách (a) biểu diễn literal, (b) bất kỳ biểu thức nào đánh giá thành một số nguyên, và (c) sử dụng hàm int().

Literal là một ký hiệu được sử dụng để biểu diễn một hằng số trực tiếp trong mã nguồn. Ví dụ –

< 8 Sync Dev />

```
a = 10
```

Tuy nhiên, xem xét việc gán biến số nguyên c như sau.

< 8 Sync Dev />

```
a = 10
b = 20
```

```
c = a + b

print ("a:", a, "type:", type(a))
print ("c:", c, "type:", type(c))
```

Nó sẽ tạo ra đầu ra sau:

< 8 Sync Dev />

```
a: 10 type: <class 'int'>
c: 30 type: <class 'int'>
```

Ở đây, c thực sự là một biến số nguyên, nhưng biểu thức a + b được đánh giá trước và giá trị của nó được gán gián tiếp cho c.

Phương pháp thứ ba để tạo một đối tượng số nguyên là với giá trị trả về của hàm int(). Nó chuyển đổi một số dấu chấm động hoặc một chuỗi thành một số nguyên.

< 8 Sync Dev />

```
a = int(10.5)
b = int("100")
```

Bạn có thể biểu diễn một số nguyên dưới dạng số nhị phân, bát phân hoặc số thập lục phân. Tuy nhiên, bên trong, đối tượng được lưu trữ là một số nguyên.

SỐ NHỊ PHÂN TRONG PYTHON

Một số gồm chỉ các chữ số nhị phân (1 và 0) và tiền tố bằng "0b" là một số nhị phân. Nếu bạn gán một số nhị phân cho một biến, nó vẫn là một biến số nguyên.

Để biểu diễn một số nguyên dưới dạng nhị phân, lưu trữ trực tiếp dưới dạng literal, hoặc sử dụng hàm int(), trong đó cơ sở được đặt thành 2.

< 8 Sync Dev />

```
a = 0b101
print ("a:", a, "type:", type(a))

b = int("0b101011", 2)
print ("b:", b, "type:", type(b))
```

Nó sẽ tạo ra đầu ra sau –

< 8 Sync Dev />

```
a: 5 type: <class 'int'>
b: 43 type: <class 'int'>
```

Cũng có một hàm bin() trong Python. Nó trả về một chuỗi nhị phân tương đương của một số nguyên.

< 8 Sync Dev />

```
a = 43
b = bin(a)
print ("Số Nguyên:", a, "Tương đương nhị phân:", b)
```

Nó sẽ tạo ra đầu ra sau –

< 8 Sync Dev />

```
Số Nguyên: 43 Tương đương nhị phân: 0b101011
```

Một số bát phân được tạo thành từ các chữ số từ 0 đến 7. Để chỉ ra rằng số nguyên sử dụng chú thích bát phân, cần được tiền tố bằng "0o" (viết thường o) hoặc "0O" (viết hoa o). Biểu diễn literal của số bát phân như sau −

```
< 8 Sync Dev />
```

```
a = 0O107
print (a, type(a))
```

Nó sẽ tạo ra đầu ra sau −

```
< 8 Sync Dev />
```

```
71 <class 'int'>
```

Lưu ý rằng đối tượng được lưu trữ bên trong là số nguyên

Số phức trong Python:

Trong Python, số phức được biểu diễn bằng một phần thực và một phần ảo, trong đó phần ảo được ký hiệu bằng chữ "j". Một số phức có thể được tạo bằng cách sử dụng hàm complex() hoặc bằng cách cung cấp một số thực và một số ảo cho hàm này.

Ví dụ:

```
< 8 Sync Dev />
```

```
# Sử dụng hàm complex()
z1 = complex(2, 3)
print(z1)  # Kết quả: (2+3j)

# Sử dụng phần thực và phần ảo
z2 = 4 + 5j
print(z2)  # Kết quả: (4+5j)
```

Khi làm việc với số phức, bạn có thể truy cập phần thực và phần ảo của số phức bằng cách sử dụng các thuộc tính `real` và `imag`, tương ứng.

< 8 Sync Dev />

```
z = 2 + 3j
print("Phần thực:", z.real)  # Kết quả: 2.0
print("Phần ảo:", z.imag)    # Kết quả: 3.0
```

Các phép toán cơ bản như cộng, trừ, nhân và chia cũng được hỗ trợ cho số phức trong Python. Ví dụ:

< 8 Sync Dev />

```
z1 = 2 + 3j
z2 = 4 + 5j

# Cộng hai số phức
sum_z = z1 + z2
print("Tổng:", sum_z)  # Kết quả: (6+8j)

# Nhân hai số phức
product_z = z1 * z2
print("Tích:", product_z)  # Kết quả: (-7+22j)
```

Để tính số phức liên conjuate (nghịch đảo của số phức), bạn có thể sử dụng phương thức `conjugate()`.

< 8 Sync Dev />

```
z = 2 + 3j
conjugate_z = z.conjugate()
print("Liên conjuguate của z:", conjugate_z)  # Kết quả: (2-3j)
```

Ngoài ra, Python cũng cung cấp một số hàm toán học cho số phức như `abs()`, `phase()`, và `polar()` để tính giá trị tuyệt đối, góc và biểu diễn dạng cực của số phức.


```
z = 3 + 4j

# Tính giá trị tuyệt đối
abs_z = abs(z)
print("Giá trị tuyệt đối:", abs_z)  # Kết quả: 5.0

# Tính góc (phase) của số phức
phase_z = phase(z)
print("Góc của số phức:", phase_z)  # Kết quả:
0.9272952180016122 (đơn vị radian)

# Biểu diễn số phức dưới dạng cực (r, phi)
polar_z = polar(z)
print("Biểu diễn cực:", polar_z)  # Kết quả: (5.0,
0.9272952180016122)
```

Đó là cách bạn có thể làm việc với số phức trong Python, bao gồm cách tạo, thực hiện các phép toán và truy cập các thuộc tính và phương thức của chúng.