**1. Phân biệt toán tử định dạng chuỗi và hàm định dạng chuỗi trong Python**

**Toán tử định dạng chuỗi (`%`):**

- Toán tử `%` là cách cũ để định dạng chuỗi trong Python. Nó hoạt động giống như hàm `sprintf` trong C, cho phép bạn nhúng các giá trị vào một chuỗi bằng cách sử dụng các ký tự đặc biệt (ví dụ: `%s`, `%d`, `%f`).

- Toán tử `%`: Cú pháp đơn giản nhưng ít linh hoạt. Thích hợp với các định dạng chuỗi đơn giản.

Ví dụ minh họa:

Ví dụ 1: Định dạng chuỗi:

name = "Alice"

formatted\_string = "Hello, %s!" % name

print(formatted\_string)

Kết quả:

Hello, Alice!

Ví dụ 2: Định dạng số nguyên:

age = 30

formatted\_string = "You are %d years old." % age

print(formatted\_string)

Kết quả:

You are 30 years old.

Ví dụ 3: Định dạng số thực:

pi = 3.14159

formatted\_string = "Pi is approximately %.2f." % pi

print(formatted\_string)

Kết quả:

Pi is approximately 3.14.

Ví dụ 4: Định dạng nhiều biến:

name = "Alice"

age = 30

formatted\_string = "Name: %s, Age: %d" % (name, age)

print(formatted\_string)

Kết quả:

Name: Alice, Age: 30

Ví dụ 5: Định dạng với padding (thêm khoảng trống):

total = 50

formatted\_string = "Total: %5d" % total

print(formatted\_string)

Kết quả:

Total: 50

**Hàm định dạng chuỗi (`str.format()`):**

- Hàm `str.format()` là cách mới hơn và linh hoạt hơn để định dạng chuỗi, được giới thiệu từ Python 2.7 và trở thành tiêu chuẩn trong Python 3. Nó cho phép nhúng giá trị vào chuỗi bằng cách sử dụng dấu ngoặc nhọn `{}`.

- `str.format()`: Linh hoạt hơn, cho phép định dạng phức tạp, dễ đọc và bảo trì.

Ví dụ minh họa:

Ví dụ 1: Định dạng chuỗi:

name = "Alice"

formatted\_string = "Hello, {}!".format(name)

print(formatted\_string)

Kết quả:

Hello, Alice!

Ví dụ 2: Định dạng số nguyên:

age = 30

formatted\_string = "You are {} years old.".format(age)

print(formatted\_string)

Kết quả:

You are 30 years old.

Ví dụ 3: Định dạng số thực:

pi = 3.14159

formatted\_string = "Pi is approximately {:.2f}.".format(pi)

print(formatted\_string)

Kết quả:

Pi is approximately 3.14.

Ví dụ 4: Định dạng nhiều biến:

name = "Alice"

age = 30

formatted\_string = "Name: {}, Age: {}".format(name, age)

print(formatted\_string)

Kết quả:

Name: Alice, Age: 30

Ví dụ 5: Định dạng với padding (thêm khoảng trống):

total = 50

formatted\_string = "Total: {:5d}".format(total)

print(formatted\_string)

Kết quả:

Total: 50

**3. Khác biệt cơ bản giữa list và tuple?**

**List:**

- Có thể thay đổi (mutable): Bạn có thể thay đổi, thêm, xóa các phần tử sau khi list được tạo.

- Khai báo: Sử dụng dấu ngoặc vuông `[]`.

- Hiệu suất: Chậm hơn tuple vì cần cấp phát bộ nhớ động khi thay đổi kích thước.

- Sử dụng: Thích hợp khi bạn cần một tập hợp các phần tử có thể thay đổi.

**Tuple:**

- Không thể thay đổi (immutable): Sau khi được tạo, bạn không thể thay đổi các phần tử bên trong tuple.

- Khai báo: Sử dụng dấu ngoặc tròn `()`.

- Hiệu suất: Nhanh hơn list, vì không cần cấp phát bộ nhớ động sau khi tạo.

- Sử dụng: Thích hợp khi bạn cần một tập hợp các phần tử cố định, không thay đổi.

**4. Ứng dụng kiểu dữ liệu tuple trong thực tế?**

- Lưu trữ dữ liệu không thay đổi: Tuple được sử dụng khi bạn muốn bảo vệ dữ liệu không bị thay đổi. Ví dụ, lưu trữ tọa độ địa lý (latitude, longitude) hoặc dữ liệu cấu hình.

- Khóa của dictionary: Vì tuple là kiểu dữ liệu không thay đổi, nó có thể được sử dụng làm khóa trong dictionary, trong khi list thì không thể.

- Trả về nhiều giá trị từ một hàm: Các hàm có thể trả về nhiều giá trị dưới dạng một tuple.

- Hiệu suất cao hơn: Trong những tình huống cần xử lý dữ liệu với tốc độ cao, tuple được ưu tiên hơn list do hiệu suất tốt hơn khi làm việc với các tập hợp dữ liệu không thay đổi.