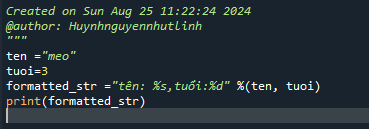
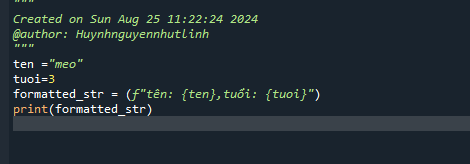
**1.**Trong Python, có nhiều cách để định dạng chuỗi, trong đó hai cách phổ biến là **toán tử định dạng chuỗi %** và các **hàm định dạng chuỗi** như str.format() và **f-strings** (định dạng chuỗi được hỗ trợ từ Python 3.6). Dưới đây là sự phân biệt giữa các phương pháp này cùng với năm ví dụ minh họa cho từng phương pháp.

Ví dụ: Cú pháp: print("Chuỗi định dạng %s" % (giá\_trị))

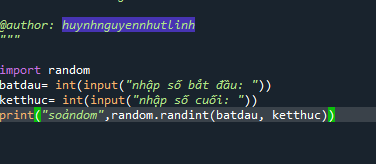


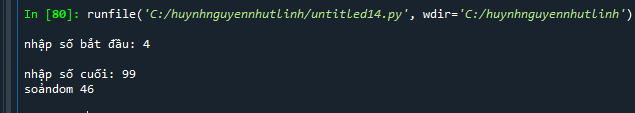


Ví dụ: print(f"Chuỗi định dạng {biểu\_thức}")

****

**2/**





**3/**

- Tính Immutability (Bất biến) và Mutability (Biến đổi)

+ List: Là một cấu trúc dữ liệu biến đổi (mutable). Điều này có nghĩa là bạn có thể thay đổi, thêm hoặc xóa các phần tử của một danh sách sau khi nó đã được tạo ra.

+ Tuple: Là một cấu trúc dữ liệu bất biến (immutable). Một khi bạn đã tạo ra một tuple, bạn không thể thay đổi, thêm hay xóa các phần tử của nó.

- Cú pháp tạo

+ List: Được tạo ra bằng cách sử dụng dấu ngoặc vuông “[]”.

+ Tuple: Được tạo ra bằng cách sử dụng dấu ngoặc đơn “()” hoặc không có dấu ngoặc cho tuple chứa một phần tử.

- Hiệu suất

+ List: Do là mutable, các thao tác với list có thể yêu cầu nhiều tài nguyên hơn về bộ nhớ và thời gian xử lý so với tuple.

+ Tuple: Do là immutable, tuples thường có hiệu suất tốt hơn khi truy cập phần tử, và chúng chiếm ít bộ nhớ hơn so với lists.

- Sử dụng

+ List: Thích hợp cho các trường hợp khi bạn cần thay đổi nội dung của tập hợp các giá trị.

+ Tuple: Thích hợp cho các trường hợp khi bạn cần một tập hợp các giá trị không thay đổi và bảo toàn tính toàn vẹn của dữ liệu.

**4/**

- Lưu trữ dữ liệu không thay đổi: Ví dụ: tọa độ địa lý như (37.7749, -122.4194) cho San Francisco.

- Trả về nhiều giá trị từ một hàm: Ví dụ: def calculate\_stats(numbers): return (min(numbers), max(numbers), sum(numbers))

- Làm khóa trong từ điển: Tuples có thể được sử dụng làm khóa trong từ điển vì chúng không thay đổi. Ví dụ: coordinate\_map = { (37.7749, -122.4194): "San Francisco" }