Họ và tên: Nguyễn Bùi Gia Bảo

MSSV: 23714851

BÀI LÀM

Câu 1: Phân biệt toán tử định dạng chuỗi và hàm định dạng chuỗi có sẵn trong gói thư viện chuẩn Python? Cho năm ví dụ minh họa tương ứng?

Chúng ta thường xuyên cần kết hợp các chuỗi văn bản với các giá trị của biến để tạo ra các thông báo, báo cáo hay các đầu ra tùy chỉnh. Để thực hiện việc này, Python cung cấp hai cách chính:

- **Toán tử định dạng chuỗi (f-strings):** Là cách hiện đại và phổ biến nhất, được giới thiệu từ Python 3.6.
- **Hàm format():** Là cách truyền thống hơn, vẫn hoạt động tốt nhưng có cú pháp hơi phức tạp hơn.

Toán tử Định dạng Chuỗi (f-strings)

- **Đặc điểm:**
    - Cú pháp trực quan, giống như cách chúng ta viết bình thường.
    - Hiệu năng cao hơn format().
    - Đặt chữ f trước dấu ngoặc kép hoặc đơn mở đầu của chuỗi.
    - Nhúng các biểu thức Python vào trong dấu ngoặc nhọn {}.
- **Ví dụ:**

Python

```
name = "BaoBao"
age = 19
print(f"Xin chào, tôi là {name} và tôi {age} tuổi.")
```

Hàm Định dạng Chuỗi format()

- **Đặc điểm:**
    - Linh hoạt hơn trong việc định dạng.
    - Sử dụng các placeholder (địa chỉ placeholder) để thay thế bằng giá trị.

- Gọi phương thức format() trên một chuỗi.
- Truyền các giá trị cần thay thế vào trong ngoặc tròn.

- **Ví dụ:**

Python

```
name = "Bao"
price = 19
print("Tôi là {} và giá sản phẩm là {:.2f} VND.".format(name, price))
```

So sánh và Ví dụ

Tính năng	Toán tử f-strings	Hàm format()
Cú pháp	Đơn giản, trực quan	Linh hoạt hơn
Hiệu năng	Cao hơn	Thấp hơn
Phiên bản Python	Từ 3.6	Các phiên bản trước

Ví dụ

1. Định dạng số:	print(f"Pi xấp xỉ bằng {3.14159:.2f}")	print("Pi xấp xỉ bằng {:.2f}".format(3.14159))
2. Định dạng ngày tháng:	import datetime; today = datetime.date.today(); print(f"Hôm nay là ngày {today:%d} tháng {today:%m} năm {today:%Y}")	import datetime; today = datetime.date.today(); print("Hôm nay là ngày {:%d} tháng {:%m} năm {:%Y}".format(today))
3. Định dạng nhiều giá trị:	x = 10; y = 20; print(f"{x} + {y} = {x+y}")	x = 10; y = 20; print("{} + {} = {}".format(x, y, x+y))
4. Định dạng với chỉ mục:	print("Tên tôi là {0} và tôi {1} tuổi.".format(name, age))	Tương tự
5. Định dạng với từ khóa:	print("Tên tôi là {name} và tôi {age} tuổi.".format(name=name, age=age))	Tương tự

Câu 2: Viết chương trình xuất ra số ngẫu nhiên trong một đoạn bất kỳ bất cho trước

```
import random
so_bat_dau = int(input("Nhập số bắt đầu của đoạn: "))
so_ket_thuc = int(input("Nhập số kết thúc của đoạn: "))
so_ngau_nhien = random.randint(so_bat_dau, so_ket_thuc)
print("Số ngẫu nhiên là:", so_ngau_hien)
```

Câu 3: Khác biệt cơ bản giữa list và tuple?

List và **tuple** là hai cấu trúc dữ liệu được sử dụng rất phổ biến trong Python để lưu trữ một tập hợp các giá trị. Tuy nhiên, chúng có những đặc điểm khác biệt quan trọng.

	List	**Tuple**
Định nghĩa	Một dãy các phần tử có thứ tự, có thể thay đổi được	Một dãy các phần tử có thứ tự, không thể thay đổi được
Sử dụng dấu	[]	()
Thay đổi phần tử	Có thể	Không thể
Thêm/xóa phần tử	Có thể	Không thể
Sử dụng	Lưu trữ các phần tử có thể thay đổi trong quá trình thực hiện chương trình	Lưu trữ các dữ liệu không đổi, như các hằng số, các cặp tọa độ,...

Khi nào nên sử dụng list và khi nào nên sử dụng tuple?

- **Sử dụng list khi:**
  - Bạn cần một cấu trúc dữ liệu linh hoạt.
  - Bạn cần thêm, xóa hoặc sửa đổi các phần tử.
- **Sử dụng tuple khi:**
  - Bạn cần một cấu trúc dữ liệu bất biến.
  - Bạn muốn đảm bảo rằng dữ liệu không bị thay đổi vô tình.

- Bạn muốn sử dụng làm key cho dictionary.
- Bạn muốn trả về nhiều giá trị từ một hàm.

Câu 4: Ứng dụng kiểu dữ liệu tuple trong thực tế

Tuple là một kiểu dữ liệu rất hữu ích trong Python, đặc biệt khi bạn cần một cấu trúc dữ liệu không thể thay đổi. Dưới đây là một số ứng dụng điển hình của tuple trong thực tế:

1. Lưu trữ dữ liệu không thay đổi:

- **Hằng số:** Tuple thường được sử dụng để lưu trữ các giá trị không bao giờ thay đổi, như các hằng số toán học (ví dụ: π, e), các ngày trong tuần, các tháng trong năm.
- **Cấu hình:** Các giá trị cấu hình của ứng dụng có thể được lưu trữ dưới dạng tuple để đảm bảo chúng không bị thay đổi vô tình.

2. Làm khóa cho dictionary:

- Vì tuple là bất biến nên chúng có thể được sử dụng làm khóa trong dictionary. Điều này rất hữu ích khi bạn muốn ánh xạ nhiều giá trị với nhau.

3. Trả về nhiều giá trị từ một hàm:

- Hàm trong Python chỉ có thể trả về một giá trị. Tuy nhiên, bạn có thể trả về một tuple chứa nhiều giá trị để mô phỏng việc trả về nhiều kết quả.

4. Định nghĩa các điểm tọa độ:

- Tuple thường được sử dụng để biểu diễn các điểm tọa độ trong không gian 2D hoặc 3D.

5. Xử lý dữ liệu:

- Tuple có thể được sử dụng để lưu trữ các bản ghi dữ liệu có cấu trúc cố định.

Ưu điểm của tuple:

- **Hiệu suất:** Tuple thường nhanh hơn list vì chúng là bất biến.
- **An toàn:** Việc tuple không thể thay đổi giúp tránh được các lỗi không mong muốn khi làm việc với dữ liệu.
- **Dễ đọc:** Cấu trúc của tuple rõ ràng và dễ hiểu.

Khi nào nên sử dụng tuple:

- Khi bạn cần một cấu trúc dữ liệu không thể thay đổi.
- Khi bạn cần một cấu trúc dữ liệu để lưu trữ các giá trị liên quan đến nhau.
- Khi bạn cần một khóa duy nhất cho một dictionary.
- Khi bạn muốn trả về nhiều giá trị từ một hàm.