**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?**

-Toán tử Định dạng Chuỗi

Toán tử % là cách định dạng chuỗi truyền thống trong Python, tương tự như định dạng chuỗi trong ngôn ngữ C. Mặc dù cách này đơn giản, nó ít linh hoạt hơn và có thể khó đọc với các định dạng phức tạp.

VD:

# Ví dụ 1: Định dạng số nguyên

name = "Alice"

age = 30

formatted\_string = "Tên: %s, Tuổi: %d" % (name, age)

print(formatted\_string) # Kết quả: Tên: Alice, Tuổi: 30

# Ví dụ 2: Định dạng số thực

pi = 3.14159

formatted\_string = "Số pi khoảng: %.2f" % pi

print(formatted\_string) # Kết quả: Số pi khoảng: 3.14

# Ví dụ 3: Định dạng với nhiều biến

name = "Bob"

score = 85.5

formatted\_string = "%s đã đạt điểm %.1f trong kỳ thi." % (name, score)

print(formatted\_string) # Kết quả: Bob đã đạt điểm 85.5 trong kỳ thi.

# Ví dụ 4: Định dạng với padding

number = 42

formatted\_string = "Số là: %05d" % number

print(formatted\_string) # Kết quả: Số là: 00042

# Ví dụ 5: Định dạng với dấu phân cách

price = 1234567.89

formatted\_string = "Giá sản phẩm: %,.2f" % price

print(formatted\_string) # Kết quả: Giá sản phẩm: 1,234,567.89

-Hàm Định dạng Chuỗi

Hàm cung cấp cách linh hoạt hơn để định dạng chuỗi. Nó cho phép chỉ định vị trí và tên tham số, làm cho mã dễ đọc và duy trì hơn.

VD:

# Ví dụ 1: Định dạng với chỉ định vị trí

name = "Alice"

age = 30

formatted\_string = "Tên: {}, Tuổi: {}".format(name, age)

print(formatted\_string) # Kết quả: Tên: Alice, Tuổi: 30

# Ví dụ 2: Định dạng số thực

pi = 3.14159

formatted\_string = "Số pi khoảng: {:.2f}".format(pi)

print(formatted\_string) # Kết quả: Số pi khoảng: 3.14

# Ví dụ 3: Định dạng với tên tham số

formatted\_string = "Tên: {name}, Tuổi: {age}".format(name="Bob", age=25)

print(formatted\_string) # Kết quả: Tên: Bob, Tuổi: 25

# Ví dụ 4: Định dạng với padding

number = 42

formatted\_string = "Số là: {:05d}".format(number)

print(formatted\_string) # Kết quả: Số là: 00042

# Ví dụ 5: Định dạng với dấu phân cách

price = 1234567.89

formatted\_string = "Giá sản phẩm: {:,.2f}".format(price)

print(formatted\_string) # Kết quả: Giá sản phẩm: 1,234,567.89

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

**trước?**

import random

a = int(input("Nhập giá trị a: "))

b = int(input("Nhập giá trị b: "))

def random\_number\_in\_range(a, b):

return random.randint(a, b)

so\_ngau\_nhien = random\_number\_in\_range(a, b)

print(f"Số ngẫu nhiên trong đoạn [{a}, {b}] là: ", so\_ngau\_nhien)

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

1. Tính Đổi Mới (Mutability)

List: Là cấu trúc dữ liệu có thể thay đổi. Bạn có thể thay đổi, thêm, hoặc xóa các phần tử trong danh sách sau khi nó đã được tạo ra.

Tuple: Là cấu trúc dữ liệu không thể thay đổi (immutable). Khi một tuple được tạo ra, các phần tử của nó không thể bị thay đổi, thêm, hoặc xóa.

2. Cú Pháp

List: Được định nghĩa bằng cách sử dụng dấu ngoặc vuông [].

Tuple: Được định nghĩa bằng cách sử dụng dấu ngoặc đơn () hoặc không dùng dấu ngoặc nếu có đủ ngữ cảnh (nhưng dấu ngoặc đơn thường được sử dụng để tránh nhầm lẫn).

3. Hiệu Suất

List: Thường có hiệu suất chậm hơn tuple vì phải hỗ trợ các thao tác thay đổi như thêm, xóa, và cập nhật phần tử.

Tuple: Có hiệu suất tốt hơn cho các thao tác đọc và truy xuất vì chúng không thay đổi và do đó có thể tối ưu hóa hơn.

4. Sử Dụng Bộ Nhớ

List: Có xu hướng tiêu tốn nhiều bộ nhớ hơn vì nó cần không gian để hỗ trợ các thao tác thay đổi.

Tuple: Tiết kiệm bộ nhớ hơn vì chúng không cần phải hỗ trợ việc thay đổi nội dung.

5. Ứng Dụng và Tính Năng

List: Thích hợp cho các tình huống mà bạn cần một cấu trúc dữ liệu có thể thay đổi, như lưu trữ và xử lý danh sách các phần tử mà bạn dự đoán sẽ cần phải thay đổi.

Tuple: Thích hợp cho các tình huống khi bạn cần một cấu trúc dữ liệu cố định và không thay đổi, chẳng hạn như các giá trị không thể thay đổi được trong các phép toán toán học, dữ liệu không thay đổi như các hằng số, hoặc dùng làm khóa trong từ điển (dict).

6. Hỗ Trợ Phương Thức

List: Có nhiều phương thức hỗ trợ như append(), remove(), extend(), pop(), và sort().

Tuple: Có ít phương thức hơn, chủ yếu là count() và index()

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

1. Hàm Trả Về Nhiều Giá Trị

Khi một hàm cần trả về nhiều giá trị, tuple là cách đơn giản và hiệu quả để gói gọn chúng lại và trả về dưới dạng một đối tượng duy nhất.

2. Chìa Khóa Trong Từ Điển

Tuple có thể được sử dụng làm chìa khóa (key) trong từ điển (dict) vì chúng là không thay đổi. Điều này hữu ích khi bạn cần sử dụng một tập hợp các giá trị không thay đổi để tra cứu.

3. Dữ Liệu Đa Dạng và Không Thay Đổi

Khi làm việc với dữ liệu cần phải không thay đổi, chẳng hạn như cấu hình, cài đặt hệ thống, hoặc cấu trúc dữ liệu bất biến, tuple là lựa chọn lý tưởng.

4. Nhóm Các Giá Trị Liên Quan

Tuple có thể được sử dụng để nhóm các giá trị liên quan lại với nhau mà không cần phải tạo một lớp riêng biệt. Điều này có thể hữu ích trong các tình huống đơn giản.

5. Tạo Danh Sách Nhanh

Tuple có thể được sử dụng để tạo các cấu trúc dữ liệu không thay đổi một cách nhanh chóng mà không cần định nghĩa một lớp hoặc cấu trúc dữ liệu riêng biệt.