Câu 1:

### Toán tử định dạng chuỗi (%)

* **Cách hoạt động**:

Toán tử % hoạt động giống như định dạng chuỗi trong ngôn ngữ C. Nó sử dụng ký hiệu % để chèn các giá trị vào chuỗi theo một định dạng cụ thể.

* **Ưu điểm**:

Đơn giản, dễ dùng cho các chuỗi định dạng đơn giản.

* **Nhược điểm**:

Hạn chế trong các trường hợp phức tạp, khó đọc và bảo trì khi định dạng chuỗi dài hoặc có nhiều biến

* **Ví dụ**

# Chèn một số nguyên

age = 25

print("I am %d years old." % age) # Output: I am 25 years old.

# Chèn một chuỗi

name = "Alice"

print("My name is %s." % name) # Output: My name is Alice.

# Chèn một số thập phân

price = 19.99

print("The price is $%.2f." % price) # Output: The price is $19.99.

# Chèn nhiều giá trị

name = "Bob"

age = 30

print("My name is %s and I am %d years old." % (name, age)) # Output: My name is Bob and I am 30 years old.

# Chèn một ký tự

char = 'A'

print("The first letter is %c." % char) # Output: The first letter is A.

### Hàm định dạng chuỗi format()

* **Cách hoạt động**:

Hàm format() là cách hiện đại hơn để định dạng chuỗi. Nó sử dụng dấu ngoặc nhọn {} làm chỗ trống để chèn các giá trị. Các giá trị có thể được định dạng theo vị trí hoặc tên biến.

* **Ưu điểm**:

Rất linh hoạt, hỗ trợ cả định dạng theo thứ tự và theo tên biến.

Dễ đọc, bảo trì và mở rộng, đặc biệt khi định dạng chuỗi dài hoặc phức tạp.

* **Nhược điểm**:

Có thể phức tạp hơn so với cách sử dụng toán tử % đối với các chuỗi đơn giản.

* **Ví dụ**

# Chèn một số nguyên

age = 25

print("I am {} years old.".format(age)) # Output: I am 25 years old.

# Chèn một chuỗi

name = "Alice"

print("My name is {}.".format(name)) # Output: My name is Alice.

# Chèn một số thập phân với định dạng

price = 19.99

print("The price is ${:.2f}.".format(price)) # Output: The price is $19.99.

# Chèn nhiều giá trị theo thứ tự

name = "Bob"

age = 30

print("My name is {} and I am {} years old.".format(name, age)) # Output: My name is Bob and I am 30 years old.

# Chèn nhiều giá trị theo tên

print("My name is {name} and I am {age} years old.".format(name="Charlie", age=35)) # Output: My name is Charlie and I am 35 years old.

Tóm lại:

· **Toán tử** %: Thích hợp cho các chuỗi định dạng đơn giản, cú pháp gần gũi với lập trình C, nhưng kém linh hoạt.

· **Hàm** format(): Linh hoạt, mạnh mẽ và dễ đọc hơn, phù hợp với các chuỗi phức tạp hoặc khi cần định dạng theo tên biến.

Câu 2:

import random

# Đoạn bất kỳ

start = 1

end = 100

# Tạo số ngẫu nhiên trong đoạn [start, end]

random\_number = random.randint(start, end)

# Xuất ra số ngẫu nhiên

print("Số ngẫu nhiên trong đoạn [", start, ",", end, "] là:", random\_number)

Câu 3:

### 1. ****Khả năng thay đổi (Mutability)****

* **List**: Mutable (Có thể thay đổi), nghĩa là bạn có thể thay đổi các phần tử của list sau khi nó được tạo. Bạn có thể thêm, xóa, hoặc cập nhật các phần tử.
* **Tuple**: Immutable (Không thể thay đổi), nghĩa là sau khi tuple được tạo, bạn không thể thay đổi, thêm, xóa, hoặc cập nhật các phần tử bên trong nó.

### 2. ****Cú pháp****

* **List**: Được tạo bằng dấu ngoặc vuông []:

my\_list = [1, 2, 3]

* **Tuple**: Được tạo bằng dấu ngoặc tròn ():

my\_tuple = (1, 2, 3)

### 3. ****Hiệu suất****

* **List**: Do có thể thay đổi, list tiêu tốn nhiều bộ nhớ hơn và các thao tác có thể chậm hơn so với tuple.
* **Tuple**: Do không thể thay đổi, tuple thường chiếm ít bộ nhớ hơn và các thao tác truy cập phần tử nhanh hơn.

### 4. ****Các thao tác hỗ trợ****

* **List**: Hỗ trợ nhiều thao tác như thêm (append()), xóa (remove(), pop()), thay đổi giá trị phần tử, và có thể sử dụng các phương thức như extend(), insert(), reverse(), v.v.
* **Tuple**: Không hỗ trợ các thao tác thay đổi như thêm, xóa hoặc cập nhật phần tử. Tuy nhiên, bạn có thể truy cập các phần tử và sử dụng các phương thức như count() và index().

### 5. ****Ứng dụng****

* **List**: Thường được sử dụng khi bạn cần một tập hợp các phần tử có thể thay đổi trong suốt quá trình chạy của chương trình.
* **Tuple**: Thường được sử dụng khi bạn cần một tập hợp các phần tử không thay đổi, đảm bảo tính toàn vẹn của dữ liệu, ví dụ như lưu trữ tọa độ hoặc các hằng số.

### 6. ****Khả năng sử dụng làm khóa trong từ điển (Dictionary Keys)****

* **List**: Do có thể thay đổi, list không thể được sử dụng làm khóa trong từ điển (dictionary).
* **Tuple**: Do không thể thay đổi, tuple có thể được sử dụng làm khóa trong từ điển.

Câu 4:

Kiểu dữ liệu `tuple` trong Python được ứng dụng rộng rãi trong nhiều tình huống thực tế, đặc biệt khi bạn cần một tập hợp dữ liệu cố định, không thay đổi. Dưới đây là một số ví dụ về cách sử dụng `tuple` trong thực tế:

**1. Lưu trữ Tọa độ (Coordinates)**

- `Tuple` thường được sử dụng để lưu trữ các tọa độ địa lý (latitude, longitude), hoặc tọa độ trong không gian 2D, 3D.

**2. Lưu trữ Các Giá Trị Không Thay Đổi (Constant Values)**

- Khi bạn muốn đảm bảo rằng một tập hợp các giá trị sẽ không thay đổi, bạn có thể sử dụng `tuple`.

**3. Sử dụng làm Khóa trong Từ Điển (Dictionary Keys)**

- Do `tuple` là immutable (không thể thay đổi), chúng có thể được sử dụng làm khóa trong từ điển. Điều này hữu ích khi bạn cần sử dụng một cặp hoặc nhiều giá trị làm khóa.

```

**4. Trả Về Nhiều Giá Trị từ Một Hàm**

- Khi một hàm cần trả về nhiều giá trị, `tuple` là một cách tiện lợi để gói các giá trị này lại.

**5. Làm Tập Dữ Liệu Bất Biến trong Cấu Trúc Dữ Liệu Phức Tạp**

- Trong các cấu trúc dữ liệu phức tạp như danh sách của danh sách, tuple thường được sử dụng để đảm bảo rằng các phần tử bên trong không bị thay đổi.

**6. Sử dụng trong Unpacking (Giải nén Giá Trị)**

- `Tuple` có thể được sử dụng để giải nén nhiều giá trị vào các biến một cách dễ dàng.

**7. Tạo Dữ Liệu Đa Chiều Cố Định**

- Trong các tính toán số học và khoa học, `tuple` có thể được sử dụng để đại diện cho các vector hoặc ma trận cố định.

Tóm lại, `tuple` được sử dụng khi bạn muốn bảo vệ dữ liệu khỏi bị thay đổi hoặc khi bạn cần một cấu trúc dữ liệu đơn giản, hiệu quả và cố định. Điều này đặc biệt quan trọng trong các tình huống yêu cầu hiệu suất cao hoặc tính toàn vẹn của dữ liệu.