* 1. **Toán tử %:** Cú pháp đơn giản nhưng hạn chế tính linh hoạt và có thể khó đọc khi định dạng phức tạp.

**Ví dụ 1:** name = "Alice" age = 30 result = "My name is %s and I am %d years old." % (name, age) print(result)

**Ví dụ 2:** pi = 3.141592653589793 result = "The value of pi is approximately %.2f." % pi print(result)

* 1. **Phương thức định dạng chuỗi str.format():**Cung cấp nhiều khả năng hơn trong việc định dạng và có thể sử dụng chỉ số vị trí và từ khóa.

**Ví dụ 3:** name = "Bob" age = 25 result = "My name is {} and I am {} years old.".format(name, age) print(result)

**Ví dụ 4:** pi = 3.141592653589793 result = "The value of pi is approximately {:.2f}.".format(pi) print(result)

* 1. **F-strings (chuỗi định dạng kiểu f):** Cung cấp cách định dạng chuỗi rõ ràng và hiệu quả nhất, cho phép nhúng các biểu thức trực tiếp trong chuỗi.

**Ví dụ 5:** name = "Charlie" age = 28 result = f"My name is {name} and I am {age} years old." print(result)

**Ví dụ 6:** pi = 3.141592653589793 result = f"The value of pi is approximately {pi:.2f}." print(result)

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

import random

def get\_random\_number():

try:

# Nhập giá trị từ người dung

lower\_bound = int(input("Nhập giá trị bắt đầu của đoạn: "))

upper\_bound = int(input("Nhập giá trị kết thúc của đoạn: "))

# Kiểm tra tính hợp lệ của đoạn

if lower\_bound > upper\_bound:

print("Giá trị bắt đầu phải nhỏ hơn hoặc bằng giá trị kết thúc.")

return

# Sinh số ngẫu nhiên trong đoạn [lower\_bound, upper\_bound]

Random\_number = random.randint(lower\_bound, upper\_bound)

# Xuất số ngẫu nhiên print(f"Số ngẫu nhiên trong đoạn [{lower\_bound}, {upper\_bound}] là: {random\_number}")

except ValueError:

print("Vui lòng nhập giá trị số nguyên hợp lệ.")

# Gọi hàm để chạy chương trình

get\_random\_number()

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

* **List** là mutable và linh hoạt hơn trong việc thay đổi dữ liệu.

Cú pháp: Được khai báo bằng dấu ngoặc vuông []. Ví dụ: my\_list = [1, 2, 3].

* **Tuple** là immutable và thường được sử dụng khi bạn cần đảm bảo rằng dữ liệu không bị thay đổi.

Cú pháp: Được khai báo bằng dấu ngoặc đơn (). Ví dụ: my\_tuple = (1, 2, 3).

1. **Ứng dụng kiểu dữ liệu tuple trong thực tế**

- Lưu trữ dữ liệu cố định: Tuple rất hữu ích khi bạn cần lưu trữ dữ liệu mà không muốn thay đổi sau khi nó đã được tạo. Ví dụ, lưu trữ thông tin cá nhân như ngày sinh, tọa độ địa lý, hay các giá trị không thay đổi khác.

- Sử dụng làm khóa trong từ điển: Tuple có thể được sử dụng làm khóa trong từ điển vì chúng không thay đổi. Điều này rất hữu ích khi bạn cần một khóa phức tạp hơn là chỉ một giá trị đơn lẻ.

- Trả về nhiều giá trị từ một hàm: Tuple là cách phổ biến để trả về nhiều giá trị từ một hàm. Điều này giúp nhóm các giá trị liên quan lại với nhau mà không cần tạo ra một đối tượng mới.

-Dữ liệu không thay đổi trong cấu trúc dữ liệu: Trong các cấu trúc dữ liệu như namedtuple từ module collections, tuple được sử dụng để tạo ra các đối tượng có thể được truy cập qua tên thuộc tính.

- Chia nhỏ các phần tử: Tuple có thể được sử dụng để chia nhỏ các phần tử khi unpacking, điều này giúp làm mã nguồn rõ ràng hơn và dễ đọc hơn.

- Đối tượng không thay đổi trong thuật toán: Trong các thuật toán mà sự thay đổi dữ liệu có thể gây ra lỗi, việc sử dụng tuple giúp bảo vệ dữ liệu không bị thay đổi, từ đó giúp giảm lỗi và cải thiện tính chính xác.

- Hợp nhất dữ liệu trong cấu trúc hợp lý: Tuple có thể được dùng để nhóm các dữ liệu liên quan lại với nhau. Ví dụ, lưu trữ thông tin về một sản phẩm trong một hệ thống quản lý kho.