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 (%)**

* **Tính năng**: Phương pháp cũ để định dạng chuỗi, sử dụng ký tự đặc biệt như ‘%s’, ‘%d’, và ‘%f’.
* **Cú pháp**: "chuỗi định dạng" % giá\_trị
* **Ưu điểm**: Dễ sử dụng cho các chuỗi đơn giản.
* **Nhược điểm**: Có thể trở nên phức tạp khi làm việc với nhiều biến và chuỗi phức tạp.

**Hàm định dạng chuỗi (str.format())**

* **Tính năng**: Được giới thiệu trong Python 3, cung cấp cách định dạng linh hoạt và dễ đọc hơn với chỉ số và tên tham số.
* **Cú pháp**: "chuỗi định dạng".format(giá\_trị)
* **Ưu điểm**: Linh hoạt hơn, dễ đọc và có thể xử lý nhiều biến.
* **Nhược điểm**: Cú pháp hơi dài hơn so với toán tử ‘%’.

**VÍ DỤ**:

VD1:

|  |  |
| --- | --- |
| **Toán tử định dạng chuỗi (%)** | **Hàm định dạng chuỗi (str.format())** |
| name = "Alice"  age = 30  formatted\_string = "Name: %s, Age: %d" % (name, age)  print(formatted\_string) | name = "Alice"  age = 30  formatted\_string = "Name: {}, Age: {}".format(name, age)  print(formatted\_string) |
| Kết quả | |
| Name: Alice, Age: 30 | Name: Alice, Age: 30 |

VD2:

|  |  |
| --- | --- |
| **Toán tử định dạng chuỗi (%)** | **Hàm định dạng chuỗi (str.format())** |
| pi = 3.14159  formatted\_string = "Value of pi: %.2f" % pi  print(formatted\_string) | pi = 3.14159  formatted\_string = "Value of pi: {:.2f}".format(pi)  print(formatted\_string) |
| Kết quả | |
| Value of pi: 3.14 | Value of pi: 3.14 |

VD3:

|  |  |
| --- | --- |
| **Toán tử định dạng chuỗi (%)** | **Hàm định dạng chuỗi (str.format())** |
| name = "Bob"  formatted\_string = "Name: %-10s" % name # Căn lề trái với độ rộng 10  print(formatted\_string) | name = "Bob"  formatted\_string = "Name: {:<10}".format(name) # Căn lề trái với độ rộng 10  print(formatted\_string) |
| Kết quả | |
| Name: Bob | Name: Bob |

VD4:

|  |  |
| --- | --- |
| **Toán tử định dạng chuỗi (%)** | **Hàm định dạng chuỗi (str.format())** |
| discount = 0.15  formatted\_string = "Discount: %.0f%%" % (discount \* 100)  print(formatted\_string) | discount = 0.15  formatted\_string = "Discount: {:.0f}%".format(discount \* 100)  print(formatted\_string) |
| Kết quả | |
| Discount: 15% | Discount: 15% |

VD5:

|  |  |
| --- | --- |
| **Toán tử định dạng chuỗi (%)** | **Hàm định dạng chuỗi (str.format())** |
| name = "Charlie"  age = 25  height = 5.9  formatted\_string = "Name: %s, Age: %d, Height: %.1f" % (name, age, height)  print(formatted\_string) | name = "Charlie"  age = 25  height = 5.9  formatted\_string = "Name: {0}, Age: {1}, Height: {2:.1f}".format(name, age, height)  print(formatted\_string) |
| Kết quả | |
| Name: Charlie, Age: 25, Height: 5.9 | Name: Charlie, Age: 25, Height: 5.9 |

1. **Viết chương trình xuất ra số ngẫu nhiên trong một đoạn bất kỳ bất cho trước?**

**import random**

def get\_random\_number\_in\_range():

start = float(input("Nhập giá trị bắt đầu : "))

stop = float(input("Nhập giá trị kết thúc : "))

if start >= stop:

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

return

random\_number = random.uniform(start, stop)

print(f"Số ngẫu nhiên trong khoảng [{start}, {stop}): {random\_number}")

get\_random\_number\_in\_range()

1. **Khác biệt cơ bản giữa list và tuple?**

|  |  |
| --- | --- |
| List | Tuple |
| Có thể thay đổi. Bạn có thể thêm, xóa hoặc sửa đổi các phần tử sau khi danh sách được tạo. | Không thể thay đổi. Sau khi tuple được tạo ra, các phần tử của nó không thể thay đổi. |
| Sử dụng dấu ngoặc vuông []. | Sử dụng dấu ngoặc đơn (). |
| Dùng khi bạn cần một tập hợp các phần tử mà có thể thay đổi. **Ví dụ**: Danh sách các mục trong giỏ hàng mà bạn có thể thêm hoặc xóa. | Dùng khi bạn cần một tập hợp các phần tử mà có thể thay đổi. **Ví dụ**: Danh sách các mục trong giỏ hàng mà bạn có thể thêm hoặc xóa. |

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

* **Lưu trữ dữ liệu không thay đổi**: Khi bạn cần lưu trữ một tập hợp các giá trị mà không thay đổi trong suốt thời gian thực thi của chương trình, tuple là sự lựa chọn hợp lý. Để bảo đảm dữ liệu không bị thay đổi.
* **Trả về nhiều giá trị từ hàm**: tuple thường được sử dụng để trả về nhiều giá trị từ một hàm, giúp mã nguồn trở nên rõ ràng và dễ đọc hơn.
* **Dữ liệu không thay đổi**: Khi bạn cần dữ liệu không thay đổi để đảm bảo tính toàn vẹn, tuple là sự lựa chọn tốt hơn list vì nó không thể bị thay đổi.
* **Khóa trong từ điển**: Vì tuple là bất biến, chúng có thể được sử dụng làm khóa trong từ điển (dict), trong khi list không thể.
* **Sắp xếp và so sánh**: tuple có thể được sử dụng trong các phép so sánh và sắp xếp, vì các phần tử của nó có thể được so sánh một cách tự nhiên.