**CÂU HỎI TỔNG HỢP**

**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.**

**TL:**

**1. Toán tử % (C-style String Formatting)**

Toán tử % là cách định dạng chuỗi cũ hơn, tương tự như trong ngôn ngữ lập trình C. Nó cho phép bạn chèn giá trị vào chuỗi bằng cách sử dụng các ký tự đặc biệt.

**Cú pháp cơ bản:** "chuỗi" % giá trị

**Các ký tự định dạng phổ biến:**

* %s - Chuỗi
* %d - Số nguyên
* %f - Số thực

**2. Toán tử định dạng chuỗi (f-string)**

* **Giới thiệu:** Được giới thiệu từ Python 3.6, f-string cung cấp một cách trực quan và linh hoạt để nhúng các biểu thức Python vào chuỗi.
* **Cú pháp:** Bắt đầu bằng chữ f, sau đó là nội dung chuỗi. Bạn có thể đặt các biểu thức Python trong cặp dấu ngoặc nhọn {} bên trong chuỗi.
* **Ưu điểm:**
  + Đọc dễ hơn, trực quan hơn so với các phương thức định dạng khác.
  + Hỗ trợ các biểu thức Python phức tạp bên trong dấu ngoặc nhọn.
  + Hiệu suất cao hơn

**3. Hàm str.format() (Format String Method)**

Hàm str.format() là phương pháp định dạng chuỗi mới hơn, cung cấp nhiều tính năng linh hoạt và dễ sử dụng hơn.

**Cú pháp cơ bản:** "chuỗi".format(giá trị)

### Tổng kết

* **Toán tử %**: Phương pháp cũ, thường gặp trong các mã nguồn cũ và các ngôn ngữ lập trình tương tự C. Thích hợp cho các định dạng đơn giản nhưng ít linh hoạt hơn.
* **Hàm str.format()**: Phương pháp mới hơn và linh hoạt hơn, hỗ trợ nhiều tùy chọn định dạng và có thể dễ đọc hơn trong nhiều tình huống.
* **f-string:** là phương thức được khuyến khích sử dụng nhất vì cú pháp đơn giản, trực quan và hiệu suất cao.

Cả hai phương pháp đều có thể sử dụng để định dạng chuỗi, nhưng str.format() thường được ưa chuộng hơn trong các ứng dụng hiện đại do tính linh hoạt và sự rõ ràng của nó.

**Ví dụ:**

**1.In thông tin người dùng:**

* **f-string:** print(f"Tên: {ten}, Tuổi: {tuoi}, Thành phố: {thanh\_pho}")
* **format():** print("Tên: {}, Tuổi: {}, Thành phố: {}".format(ten, tuoi, thanh\_pho))
* **Toán tử %:** print("Tên: %s, Tuổi: %d, Thành phố: %s" % (ten, tuoi, thanh\_pho))

**2.Định dạng số:**

* **f-string:** print(f"Pi xấp xỉ bằng {3.14159:.2f}")
* **format():** print("Pi xấp xỉ bằng {:.2f}".format(3.14159))
* **Toán tử %:** print("Pi xấp xỉ bằng %.2f" % 3.14159)

**3.Định dạng ngày tháng:**

* **f-string:** from datetime import datetime; print(f"Hôm nay là ngày {datetime.now():%d/%m/%Y}")
* **format():** from datetime import datetime; print("Hôm nay là ngày {:%d/%m/%Y}".format(datetime.now()))

**4.Định dạng chuỗi nhiều dòng:**

* **f-string:**

Python

text = (f"Dòng 1\n"

f"Dòng 2\n"

f"Dòng 3")

print(text)

* **format():**

Python

text = ("Dòng 1\n"

"Dòng 2\n"

"Dòng 3")

print(text)

**5.Định dạng phức tạp:**

* **f-string:**

Python

ten = "Huy"

diem = [10, 8.5, 9]

print(f"{name} có điểm trung bình là {(sum(diem)/len(diem)):.2f}")

* **format():**

Python

ten = "Trang"

diem = [7.5, 9, 9.5]

print("{} có điểm trung bình là {:.2f}".format(ten, sum(diem)/l

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

**TL:**

1. **Xuất ra một số nguyên ngẫu nhiên trong khoảng từ a đến b**

import random

so\_ngau\_nhien = random.randint(a, b)

print("Số nguyên ngẫu nhiên:", so\_ngau\_nhien)

1. **Xuất ra một số thực ngẫu nhiên trong khoảng từ a đến b**

import random

so\_thuc\_ngau\_nhien = random.uniform(a, b)

print("Số thực ngẫu nhiên:", so\_thuc\_ngau\_nhien)

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

**TL:**

**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 nhau rất quan trọng.

**1. Độ biến đổi (mutability):**

* **List:** Có thể thay đổi sau khi được tạo. Bạn có thể thêm, xóa hoặc sửa đổi các phần tử trong một list.
* **Tuple:** Không thể thay đổi sau khi được tạo. Các phần tử của tuple được cố định.

**2. Cách khai báo:**

* **List:** Được khai báo bằng dấu ngoặc vuông []. Ví dụ: my\_list = [1, 2, 3, "hello"]
* **Tuple:** Được khai báo bằng dấu ngoặc tròn (). Ví dụ: my\_tuple = (1, 2, 3,"hello")

**3. Sử dụng:**

* **List:**
  + Lưu trữ các phần tử có thể thay đổi theo thời gian.
  + Làm việc với các cấu trúc dữ liệu động.
  + Thường được sử dụng trong các thuật toán và cấu trúc dữ liệu khác.
* **Tuple:**
  + Lưu trữ các dữ liệu không thay đổi.
  + Làm khóa cho dictionary.
  + Trả về nhiều giá trị từ một hàm.
  + Tăng tốc độ thực thi trong một số trường hợp vì tính bất biến.

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

**TL:**

Dưới đây là một số ứng dụng điển hình của tuple trong thực tế:

**1. Lưu trữ các giá trị không đổi:**

* **Tọa độ:** Một điểm trên mặt phẳng có thể được biểu diễn bằng một tuple (x, y). Vì tọa độ thường không thay đổi, nên tuple là một lựa chọn phù hợp.
* **Màu sắc:** Một màu sắc trong hệ thống RGB có thể được biểu diễn bằng một tuple (R, G, B), với R, G, B lần lượt là các giá trị thành phần màu đỏ, xanh lá và xanh dương.
* **Ngày tháng:** Một ngày tháng có thể được biểu diễn bằng một tuple (ngày, tháng, năm).

**2. Làm khóa cho dictionary:**

Vì tuple là bất biến, nên chúng có thể được sử dụng làm khóa cho dictionary. Điều này rất hữu ích khi bạn muốn ánh xạ một cặp giá trị (hoặc nhiều giá trị hơn) đến một giá trị khác.

**3. Trả về nhiều giá trị từ một hàm:**

Một hàm trong Python có thể trả về nhiều giá trị bằng cách đóng gói chúng trong một tuple.

**4. Định nghĩa các cấu trúc dữ liệu đơn giản:**

Tuple có thể được sử dụng để định nghĩa các cấu trúc dữ liệu đơn giản như điểm, khoảng thời gian, phân số, v.v.

**5. Các ứng dụng khác:**

* **Lưu trữ các hằng số:** Các hằng số như các ngày trong tuần, tháng trong năm thường được định nghĩa bằng tuple.
* **Truyền nhiều đối số cho một hàm:** Tuple có thể được sử dụng để truyền một số lượng không xác định các đối số cho một hàm.
* **Xử lý dữ liệu:** Tuple có thể được sử dụng để lưu trữ các bản ghi dữ liệu, chẳng hạn như các hàng trong một bảng dữ liệu.