### Phân biệt toán định dạng chuỗi và hàm định dạng chuỗi có sẵn trong gói thư viện chuản Python? cho 5 ví dụ minh họa tương ứng

### - **Toán tử định dạng chuỗi** % **(String Formatting Operator)**

* Đây là cách cũ hơn trong Python để định dạng chuỗi.
* Sử dụng ký tự % để chèn giá trị vào chuỗi với các loại dữ liệu được xác định bởi ký tự định dạng như %s, %d, %f, v.v.

### - **Hàm định dạng chuỗi** .format()

* Cách này được giới thiệu từ Python 3 và cung cấp cách định dạng chuỗi linh hoạt hơn.
* Sử dụng dấu {} làm vị trí đặt chèn và các giá trị được truyền vào hàm .format().

### - **F-string (Literal String Interpolation)**

* Đây là cách mới nhất và tiện lợi nhất để định dạng chuỗi, được giới thiệu từ Python 3.6. F-string cho phép chèn trực tiếp các giá trị hoặc biểu thức vào trong chuỗi bằng cách sử dụng ký tự f ở đầu chuỗi và sử dụng {} để chèn giá trị.

Ví dụ:

#### - **Toán tử định dạng chuỗi** %

Ví dụ 1: Định dạng chuỗi với kiểu dữ liệu chuỗi và số nguyên

name = "John"

age = 30

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

Ví dụ 2: Định dạng số thực với số chữ số sau dấu thập phân

pi = 3.14159

result = "Pi is approximately %.2f" % piprint(result)# Output: Pi is approximately 3.14

- **Hàm định dạng chuỗi** .format()

Ví dụ 3: Định dạng chuỗi với .format()

name = "Alice"

age = 25

result = "My name is {} and I am {} years old.".format(name, age)print(result)# Output: My name is Alice and I am 25 years old.

Ví dụ 4: Định dạng số thực với .format()

pi = 3.14159

result = "Pi is approximately {:.2f}".format(pi)print(result)# Output: Pi is approximately 3.14

#### - **F-string**

#### Ví dụ 5: Định dạng chuỗi với f-string

name = "Bob"

age = 22

result = f"My name is {name} and I am {age} years old."print(result)# Output: My name is Bob and I am 22 years old.

Ví dụ 6: Định dạng số thực với f-string

pi = 3.14159

result = f"Pi is approximately {pi:.2f}"print(result)# Output: Pi is approximately 3.14

2)Khác biệt cơ bản giữa list và tuple

| **Thuộc tính** | List | Tuple |
| --- | --- | --- |
| **Cú pháp** | Dùng dấu ngoặc vuông [] | Dùng dấu ngoặc tròn () |
| **Tính thay đổi (Mutable/Immutable)** | **Có thể thay đổi**: Bạn có thể thay đổi, thêm, xóa các phần tử của danh sách sau khi nó được tạo. | **Không thể thay đổi**: Sau khi tuple được tạo, bạn không thể thay đổi, thêm, hay xóa các phần tử. |
| **Hiệu suất** | **Chậm hơn** so với tuple, do có tính thay đổi. | **Nhanh hơn** so với list, vì nó không thay đổi sau khi được tạo. |
| **Kích thước bộ nhớ** | **Sử dụng nhiều bộ nhớ hơn** do có thể thay đổi kích thước và lưu trữ nhiều phương thức. | **Sử dụng ít bộ nhớ hơn** do tính cố định của nó. |
| **Phương thức có sẵn** | **Nhiều phương thức hơn** như append(), remove(), pop(), insert(). | **Ít phương thức hơn**, chỉ có các phương thức như count(), index(). |
| **Sử dụng** | **Thích hợp cho các tập dữ liệu có thể thay đổi** hoặc khi bạn cần thao tác với các phần tử (thêm, sửa, xóa). | **Thích hợp cho các tập dữ liệu không thay đổi** hoặc cố định như danh sách các hằng số, tọa độ, v.v. |
| **Duyệt qua** | **Duyệt qua nhanh hơn một chút** do tính năng tối ưu hóa cho việc truy cập, mặc dù hiệu suất không khác nhau nhiều đối với khối lượng nhỏ. | **Duyệt qua chậm hơn một chút** khi làm việc với các cấu trúc lớn. |

1. Ứng dụng kiểu dữ liệu tuple trong thực tế

### - **Lưu trữ dữ liệu cố định**

* Tuple là lựa chọn tốt khi cần lưu trữ các dữ liệu **không thay đổi** trong suốt vòng đời của chương trình. Ví dụ, lưu các thông tin hằng số như tọa độ địa lý, giá trị của các phép đo khoa học, hoặc các cặp giá trị liên kết.
* **Ví dụ:**

Tọa độ của một điểm cố định

coordinates = (40.7128, -74.0060) # Tọa độ của New York

### 2. **Khóa trong từ điển (Dictionary)**

* Tuple có thể được sử dụng làm **khóa** trong một **dictionary** vì chúng **bất biến**. Điều này rất hữu ích khi bạn cần sử dụng các cặp giá trị làm khóa hoặc khi các khóa không thay đổi.

**Ví dụ:**

Sử dụng tuple làm khóa

locations = {('New York', 'USA'): 8000000, ('Tokyo', 'Japan'): 14000000}print(locations[('New York', 'USA')]) # Output: 8000000

### - **Trả về nhiều giá trị từ một hàm**

* Một cách sử dụng phổ biến của tuple là **trả về nhiều giá trị** từ một hàm. Do tuple có cú pháp ngắn gọn và không cần tạo danh sách, nên nó là một lựa chọn tối ưu.

**Ví dụ:**

def get\_user\_info():

name = "Alice"

age = 30

country = "USA"

return name, age, country # Trả về tuple

user\_info = get\_user\_info()print(user\_info) # Output: ('Alice', 30, 'USA')

### 4. **Sử dụng như một cấu trúc dữ liệu đơn giản để nhóm các giá trị**

* Tuple có thể được sử dụng như một **cấu trúc dữ liệu** để nhóm các giá trị có liên quan lại với nhau, chẳng hạn như dữ liệu về ngày tháng, hoặc các cặp thông tin như tên và tuổi.

**Ví dụ:**

python

Sao chép mã

# Ngày tháng lưu trữ dưới dạng tuple

birthdate = (1990, 12, 25) # Năm, tháng, ngày

### 5. **Cải thiện hiệu suất so với list**

* Do tuple không thể thay đổi, Python có thể tối ưu hóa bộ nhớ và thời gian truy cập tốt hơn. Trong các chương trình yêu cầu hiệu suất cao hoặc các cấu trúc dữ liệu không cần thay đổi, sử dụng tuple có thể tăng cường tốc độ xử lý.

**Ví dụ:**

large\_tuple = (1, 2, 3) \* 1000000 # Tạo tuple lớn

### 6. **Bảo vệ dữ liệu khỏi sự thay đổi**

* Trong một số trường hợp, bạn cần đảm bảo rằng dữ liệu không bị thay đổi trong suốt quá trình xử lý. Bằng cách sử dụng tuple, bạn có thể đảm bảo rằng dữ liệu được bảo vệ khỏi các thay đổi vô ý.

**Ví dụ:**

Danh sách ngày nghỉ lễ không thay đổi

holidays = ("New Year", "Christmas", "Independence Day")

### 7. **Sử dụng trong lập trình đồ họa hoặc khoa học**

* **Tọa độ trong không gian 2D hoặc 3D** thường được lưu dưới dạng tuple vì chúng không thay đổi và dễ truy cập. Ví dụ, khi làm việc với các điểm ảnh, vector, hoặc các đối tượng đồ họa.

**Ví dụ:**

Tọa độ trong không gian 3D

point\_3d = (3, 4, 5)

### 8. **Làm tham số hàm**

* Tuple cũng được sử dụng để truyền nhiều tham số vào hàm một cách tiện lợi. Khi một hàm cần nhận nhiều giá trị mà không muốn thay đổi chúng, tuple là lựa chọn tốt.

**Ví dụ:**

def calculate\_distance(point1, point2):

Tính khoảng cách giữa hai điểm trong không gian 2D

return ((point2[0] - point1[0]) \*\* 2 + (point2[1] - point1[1]) \*\* 2) \*\* 0.5

point1 = (1, 2)

point2 = (4, 6)

distance = calculate\_distance(point1, point2)print(distance) # Output: 5.0