Câu 1: Phân biệt toán tử định dạng chuỗi và hàm định dạng chuỗi có sẵn trong gói thư viện chuẩn python. Cho 5 ví dụ minh họa tương ứng

**Toán tử %(Định dạng kiểu cũ)**

Cú pháp: chuỗi % (giá\_trị,)

Cách thức: Giá trị được đặt vào trong chuỗi bằng cách sử dụng dấu % như trình giữ chỗ.

Ví dụ:

python

name = "Alice"

age = 30

print("Hello, my name is %s and I am %d years old." % (name, age))

Ưu điểm: Cú pháp đơn giản, dễ hiểu.

Nhược điểm: Ít linh hoạt so với format(), đặc biệt khi định dạng phức tạp.

**Phương thức format()(Định dạng kiểu mới)**

Cú pháp: chuỗi.format(giá\_trị1, giá\_trị2, ...)

Cách thức: Giá trị được đặt vào trong chuỗi bằng cách sử dụng dấu ngoặc nền {}như trình giữ chỗ.

Ví dụ:

python

name = "Bob"

price = 19.99

print("The product costs {:.2f} dollars and is bought by {}".format(price, name))

Ưu điểm: Linh hoạt hơn, hỗ trợ nhiều loại phức tạp định dạng, có thể sử dụng số hoặc tên để truyền tham số.

Nhược điểm: Cú pháp có thể kéo dài dòng hơn trong một số trường hợp đơn giản.

Ví dụ minh họa:

**Định dạng đơn giản:**

# Toán tử %

print("My name is %s" % "Alice")

# Phương thức format()

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

**Định dạng nhiều giá trị:**

# Toán tử %

print("The sum of %d and %d is %d" % (3, 4, 3+4))

# Phương thức format()

print("The sum of {} and {} is {}".format(3, 4, 3+4))

Định dạng số với độ chính xác cao:

python

# Phương thức format()

pi = 3.14159

print("Pi to two decimal places is {:.2f}".format(pi))

**Định dạng ngày tháng:**

python

import datetime

today = datetime.date.today()

# Phương thức format()

print("Today's date is: {:%B %d, %Y}".format(today))

**Format with tham số name:**

Python

# Phương thức format()

person = {"name": "Bob", "age": 30}

print("Hello, {name}. You are {age} years old.".format(\*\*person))

Câu 2:

import random

# Nhập giá trị cho đoạn [a, b]

a = float(input("Nhập giá trị bắt đầu của đoạn (a): "))

b = float(input("Nhập giá trị kết thúc của đoạn (b): "))

# Kiểm tra để đảm bảo a nhỏ hơn b

if a >= b:

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

else:

# Sinh số ngẫu nhiên trong khoảng [a, b)

random\_number = random.uniform(a, b)

# Xuất kết quả

print(f"Số ngẫu nhiên trong khoảng [{a}, {b}): {random\_number:.3f}")

Câu 3: Khác biệt cơ bản giữa list và tuple

**Tính bất biến (Immutability):**

* **List (Danh sách):** Là kiểu dữ liệu có thể thay đổi (mutable). Điều này có nghĩa là bạn có thể thay đổi các phần tử của danh sách sau khi đã tạo ra nó.
* **Tuple (Bộ dữ liệu):** Là kiểu dữ liệu không thể thay đổi (immutable). Một khi bạn tạo ra một tuple, bạn không thể thay đổi các phần tử của nó.

**Cú pháp:**

List: Sử dụng dấu ngoặc vuông [].

Tuple: Sử dụng dấu ngoặc đơn ()

**Hiệu suất:**

List: Có thể có hiệu suất thấp hơn trong một số trường hợp do tính linh hoạt của nó (thay đổi kích thước, cập nhật phần tử, v.v.).

Tuple: Thường có hiệu suất cao hơn vì chúng là bất biến và có kích thước cố định. Điều này làm cho chúng nhẹ hơn và nhanh hơn trong một số thao tác.

**Ứng dụng:**

List: Thích hợp cho các tình huống mà bạn cần một tập hợp các phần tử có thể thay đổi (ví dụ: danh sách các mục cần thực hiện).

Tuple: Thích hợp cho các tình huống mà bạn cần một tập hợp các phần tử không thay đổi (ví dụ: các giá trị cấu hình, hoặc làm khóa cho dictionary).

**Các phương thức hỗ trợ:**

List: Cung cấp nhiều phương thức như append(), extend(), remove(), pop(), v.v.

Tuple: Không có các phương thức tương tự như trên vì chúng không thể thay đổi. Tuy nhiên, bạn có thể sử dụng các phương thức count() và index().

**Tính chất hashable:**

List: Không thể được sử dụng làm khóa trong dictionary hoặc phần tử của set vì chúng không có tính hash.

Tuple: Có thể được sử dụng làm khóa trong dictionary hoặc phần tử của set nếu tất cả các phần tử của tuple cũng là hashable.

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

**Lưu trữ dữ liệu không thay đổi**

Ứng dụng:

Cấu hình và tham số: Khi bạn cần lưu trữ một tập hợp các giá trị không thay đổi (như cấu hình của một ứng dụng hoặc các tham số), tuple là lựa chọn lý tưởng vì nó không thể bị thay đổi sau khi được tạo ra.

python

Copy code

# Cấu hình ứng dụng

app\_config = ("localhost", 8080, "my\_database")

**Trả về nhiều giá trị từ một hàm**

Ứng dụng:

Hàm trả về nhiều giá trị: Tuple rất hữu ích khi một hàm cần trả về nhiều giá trị. Ví dụ, bạn có thể trả về một cặp giá trị (hoặc nhiều giá trị) từ một hàm.

python

Copy code

def get\_min\_max(values):

return (min(values), max(values))

min\_val, max\_val = get\_min\_max([3, 1, 4, 1, 5, 9])

print(f"Min: {min\_val}, Max: {max\_val}")

**Lưu trữ dữ liệu với các thuộc tính cố định**

Ứng dụng:

Điểm dữ liệu: Bạn có thể sử dụng tuple để lưu trữ các điểm dữ liệu với các thuộc tính cố định, chẳng hạn như tọa độ (x, y) trong không gian 2D.

python

Copy code

point = (10, 20)

x, y = point

print(f"X: {x}, Y: {y}")

**Sử dụng trong các cấu trúc dữ liệu như tên và tuổi**

Ứng dụng:

Danh sách các đối tượng: Tuple có thể được sử dụng để lưu trữ các đối tượng với các thuộc tính cố định, chẳng hạn như danh sách các nhân viên với tên và tuổi.

python

Copy code

employees = [

("Alice", 30),

("Bob", 25),

("Charlie", 35)

]

for name, age in employees:

print(f"Name: {name}, Age: {age}")