Họ tên: Đặng Thị Thùy Trang

Mssv: 23694231

Lớp học phần: DHTMDT19C

**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?**

Trong Python, có nhiều cách để định dạng chuỗi, tạo ra các chuỗi đầu ra đẹp mắt và dễ đọc hơn. Hai cách phổ biến nhất là sử dụng **toán tử %** và **phương thức format()**. Mỗi cách có ưu nhược điểm riêng, phù hợp với những trường hợp khác nhau.

**Toán tử % (Old-style string formatting)**

* **Cú pháp:**

Python

chuoi % (gia\_tri1, gia\_tri2, ...)

* **Cách hoạt động:**
  + %s: Định dạng chuỗi
  + %d: Định dạng số nguyên
  + %f: Định dạng số thực
  + ... và nhiều định dạng khác

**Phương thức format() (New-style string formatting)**

* **Cú pháp:**

Python

chuoi.format(gia\_tri1, gia\_tri2, ...)

* **Cách hoạt động:**
  + Sử dụng dấu ngoặc nhọn {} để đặt chỗ cho các giá trị cần thay thế.
  + Có thể sử dụng chỉ số hoặc tên để chỉ định vị trí của giá trị.

**So sánh và lựa chọn**

|  |  |  |
| --- | --- | --- |
| Tính năng | Toán tử % | Phương thức format() |
| Cú pháp | Dễ nhớ nhưng kém linh hoạt | Linh hoạt hơn, có thể sử dụng chỉ số, tên |
| Đọc hiểu | Có thể khó đọc khi có nhiều giá trị | Dễ đọc, rõ ràng hơn |
| Tính năng | Hạn chế về định dạng | Hỗ trợ nhiều định dạng phức tạp hơn |
| Python 3 | Khuyến khích không sử dụng | Được khuyến khích sử dụng |

* **Toán tử %:**
  + Các trường hợp đơn giản, cần định dạng nhanh.
  + Các dự án cũ đã sử dụng toán tử %.
* **Phương thức format():**
  + Các trường hợp phức tạp, cần định dạng linh hoạt.
  + Các dự án mới, theo chuẩn PEP 8.
  + Các trường hợp cần định dạng số với độ chính xác cao, định dạng ngày tháng, ...

**f-strings (Python 3.6 trở lên)**

**Cú pháp:**

Python

f"chuoi {gia\_tri1} {gia\_tri2}"

**Cách hoạt động:**

* + Là một cách viết tắt của phương thức format(), cho phép đặt biểu thức Python trực tiếp trong dấu ngoặc nhọn.

Phương thức format() và f-strings là những cách định dạng chuỗi hiện đại và linh hoạt hơn, được khuyến khích sử dụng trong các dự án Python hiện nay. Tuy nhiên, việc lựa chọn phương thức nào còn phụ thuộc vào từng trường hợp cụ thể và sở thích của người lập trình.

**Ví dụ**

# Toán tử %

print("Giá trị pi xấp xỉ %.2f" % 3.14159)

# Phương thức format()

name = "David"

age = 35

print("Xin chào, {}! Bạn {} tuổi.".format(name, age))

# f-strings

radius = 5

area = 3.14159 \* radius\*\*2

print(f"Diện tích hình tròn bán kính {radius} là: {area:.2f}")

Ví dụ:

Pyyhon

name = "Charlie"

age = 25

print(f"Hello, {name}. You are {age} years old.")

**Ví dụ:**

Python

name = "Charlie"

age = 25

print(f"Hello, {name}. You are {age} years old.")

**Ví dụ:**

Python

name = "Bob"

price = 19.99

print("Hello, my name is {} and the price is {:.2f}".format(name, price))

**Ví dụ:**

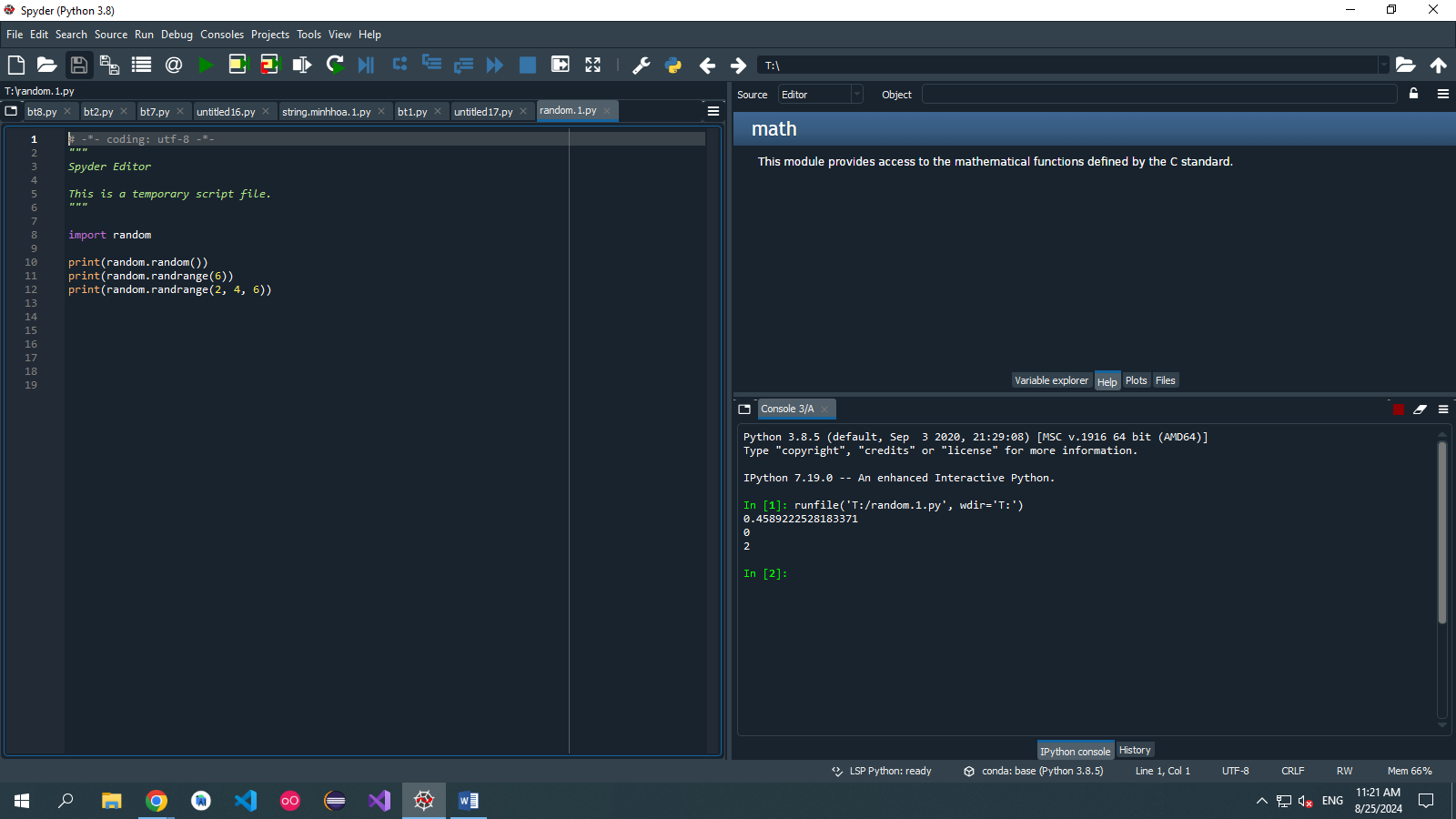
Python

name = "Alice"

age = 30

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

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



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

**List** và **tuple** là hai kiểu dữ liệu được sử dụng rất phổ biến trong Python để lưu trữ các tập hợp giá trị. Tuy nhiên, chúng có một số khác biệt quan trọng về tính chất và cách sử dụng

**-Tính biến đổi (mutability)**

**List:** Có thể thay đổi sau khi được tạo. Bạn có thể thêm, xóa hoặc sửa đổi các phần tử trong list

**Tuple:** Không thể thay đổi. Một khi tuple được tạo, các phần tử trong đó không thể thay đổi

**-Cú pháp**

**+ List:** Được bao quanh bởi dấu ngoặc vuông []

**+ Tuple:** Được bao quanh bởi dấu ngoặc tròn ()

**-Sử dụng**

* **List:**
  + Lưu trữ các phần tử có thể thay đổi theo thời gian
  + Thường được sử dụng khi cần một cấu trúc dữ liệu linh hoạt
  + Làm việc với các thuật toán liên quan đến việc thêm, xóa hoặc sắp xếp các phần tử
* **Tuple:**
  + Lưu trữ các giá trị không đổi
  + Thường được sử dụng để tạo các khóa cho dictionaries, các tham số cho hàm hoặc khi cần bảo vệ dữ liệu khỏi bị thay đổi vô tình
  + Làm việc với các dữ liệu đầu vào, các hằng số

**Tuple:** Thường nhanh hơn list một chút vì tính chất bất biến của nó giúp cho việc truy cập và so sánh các phần tử trở nên hiệu quả hơn.

**Sử dụng list khi:** Bạn cần một cấu trúc dữ liệu linh hoạt, có thể thay đổi kích thước và nội dung

**Sử dụng tuple khi:** Bạn cần một cấu trúc dữ liệu bất biến, hiệu quả và muốn bảo vệ dữ liệu khỏi bị thay đổi vô tình

**Tóm lại:** Sự khác biệt chính giữa list và tuple là về tính biến đổi. Việc lựa chọn sử dụng list hay tuple phụ thuộc vào yêu cầu cụ thể của bài toán

## Câu 4: Ứng dụng của kiểu dữ liệu Tuple trong thực tế

**Tuple** là một cấu trúc dữ liệu hữu ích trong nhiều ngôn ngữ lập trình, đặc biệt là Python. Chúng thường được sử dụng để nhóm các giá trị liên quan với nhau thành một đơn vị duy nhất, nhưng không thể thay đổi được

**Các ứng dụng thực tế:**

* **Tọa độ:** Một điểm trên một mặt phẳng có thể được biểu diễn bằng một tuple (x, y)
* **Kích thước:** Kích thước của một hình ảnh hoặc màn hình có thể được biểu diễn bằng một tuple (width, height)
* **Ngày tháng:** Một ngày tháng có thể được biểu diễn bằng một tuple (year, month, day)
* **Màu sắc:** Một màu sắc trong hệ màu RGB có thể được biểu diễn bằng một tuple (red, green, blue)
* **Kết quả của một hàm:** Một hàm tính toán diện tích và chu vi của một hình chữ nhật có thể trả về một tuple (diện\_tích, chu\_vi)
* **Dữ liệu cấu trúc:** Tuple có thể được sử dụng để tạo các cấu trúc dữ liệu đơn giản như danh sách các điểm, danh sách các ngày, v.v…

**Tóm lại,** tuple là một công cụ linh hoạt và hữu ích trong lập trình. Chúng giúp code trở nên rõ ràng hơn, dễ bảo trì hơn và đảm bảo tính toàn vẹn của dữ liệu