Họ và tên: Lê Thị Ngọc Hân

MSSV: 23721801

BÀI LÀM

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?

*** Phân biệt Toán tử Định dạng Chuỗi và Hàm Định dạng Chuỗi trong Python**

1. Hiểu rõ khái niệm và sự khác biệt:

Toán tử định dạng chuỗi (f-strings):

- Được giới thiệu từ Python 3.6.
- Cú pháp trực quan, dễ đọc: f"chuỗi {biến}".
- Cho phép nhúng trực tiếp giá trị của biến vào trong chuỗi.
- Hỗ trợ các định dạng phức tạp như căn chỉnh, độ rộng, định dạng số, v.v.

Hàm định dạng chuỗi format():

- Có từ các phiên bản Python trước đó.
- Cú pháp linh hoạt hơn: "chuỗi {}".format(biến).
- Cho phép định dạng nhiều biến, sử dụng chỉ số hoặc tên để tham chiếu.
- Cũng hỗ trợ các định dạng phức tạp tương tự như f-strings.

2. So sánh ưu nhược điểm:

Tính năng: ưu điểm: Toán tử định dạng chuỗi (f-strings) - nhược điểm: Hàm định dạng chuỗi format()

Cú pháp: ưu điểm: Trực quan, dễ đọc - nhược điểm: Linh hoạt hơn

Hiệu suất: ưu điểm: Thường nhanh hơn - nhược điểm: Có thể chậm hơn một chút

Tính năng: ưu điểm: Hỗ trợ đầy đủ các định dạng- nhược điểm: Hỗ trợ đầy đủ các định dạng

Python version: ưu điểm: Từ 3.6 trở lên - nhược điểm: Từ các phiên bản trước đó

Tuyệt vời, bạn đã bắt đầu một bài làm rất tốt về định dạng chuỗi trong Python. Để hoàn thiện hơn, chúng ta hãy cùng đi sâu vào từng phần và bổ sung thêm một số chi tiết nhé.

3. Ví dụ minh họa:

a. Toán tử định dạng chuỗi:

```
name = "Hân"
age = 19
print(f"Xin chào, tôi là {name}. Tôi {age} tuổi.")
# Xuất: Xin chào, tôi là Hân. Tôi 19 tuổi.
```

b. Hàm định dạng chuỗi:

```
name = "Hân"
age = 19
print("Xin chào, tôi là {}. Tôi {} tuổi.".format(name, age))
# Xuất: Xin chào, tôi là Hân. Tôi 19 tuổi.
```

c. Định dạng số:

```
pi = 3.14159
print(f"Giá trị của pi là: {pi:.2f}")
# Xuất: Giá trị của pi là: 3.14
```

d. Căn chỉnh:

```
text = "Python"
print(f"{text:>10}")
# Xuất:       Python
```

e. Định dạng nhiều biến:

Python

```
a = 10
b = 20
```

print("Tổng của {} và {} là: {}".format(a, b, a+b))

Xuất: Tổng của 10 và 20 là: 30

4. Mở rộng và nâng cao:

Các định dạng nâng cao:

- Định dạng ngày tháng, giờ
- Định dạng số theo kiểu khoa học
- Định dạng tiền tệ

Kết hợp với các hàm khác:

- Sử dụng hàm len() để lấy độ dài chuỗi
- Sử dụng hàm str() để chuyển đổi kiểu dữ liệu sang chuỗi

Các trường hợp sử dụng:

- Tạo các chuỗi báo cáo, log
- Định dạng đầu ra của các hàm, lớp
- Tạo các chuỗi cấu hình

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

import random

def random_number(min_value, max_value):

return random.randint(min_value, max_value)

min_number = int(input("Nhập giá trị nhỏ nhất của khoảng: "))

max_number = int(input("Nhập giá trị lớn nhất của khoảng: "))

random_result = random_number(min_number, max_number)

print("Số ngẫu nhiên trong khoảng đã cho là:", random_result)

Ví dụ:

Nếu bạn nhập 1 và 100, chương trình có thể in ra một số ngẫu nhiên từ 1 đến 100, chẳng hạn như 88.

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

List và **tuple** là 2 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 biệt quan trọng.

1. Tính biến đổi (mutability):

- **List:** Là một đối tượng có thể thay đổi. Bạn có thể thêm, xóa hoặc sửa đổi các phần tử trong một list sau khi nó được tạo.
- **Tuple:** Là một đối tượng bất biến. Một khi một tuple được tạo, bạn không thể thay đổi các phần tử bên trong nó.

2. Cú pháp:

- **List:** Được bao quanh bởi các dấu ngoặc vuông [].
- **Tuple:** Được bao quanh bởi các dấu ngoặc tròn ().

3. Sử dụng:

- **List:**
    - Được sử dụng khi bạn cần một cấu trúc dữ liệu linh hoạt, có thể thay đổi kích thước và nội dung.
    - Thường được sử dụng để lưu trữ các danh sách các phần tử cùng loại

    ví dụ: một danh sách các chuỗi
- **Tuple:**
    - Được sử dụng khi bạn cần một cấu trúc dữ liệu bất biến, không thay đổi.
    - Thường được sử dụng để lưu trữ các dữ liệu không thay đổi, như các hằng số, các cặp khóa-giá trị trong một dictionary.
    - Tuple cũng được sử dụng để trả về nhiều giá trị từ một hàm.

Khi nào nên sử dụng list và khi nào nên sử dụng tuple?

- **Sử dụng list khi:**
    - Bạn cần một cấu trúc dữ liệu linh hoạt.
    - Bạn cần thêm, xóa hoặc sửa đổi các phần tử.
- **Sử dụng tuple khi:**
    - Bạn cần một cấu trúc dữ liệu bất biến.

- Bạn muốn đảm bảo rằng dữ liệu không bị thay đổi vô tình.
- Bạn muốn sử dụng làm key cho dictionary.
- Bạn muốn trả về nhiều giá trị từ một hàm.

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

Tuple là một kiểu dữ liệu rất hữu ích trong Python, đặc biệt khi bạn cần một cấu trúc dữ liệu không thể thay đổi. Dưới đây là một số ứng dụng điển hình của tuple trong thực tế:

1. Lưu trữ dữ liệu không thay đổi:

- **Hằng số:** Tuple thường được sử dụng để lưu trữ các giá trị không bao giờ thay đổi, như các hằng số toán học (ví dụ: π, e), các ngày trong tuần, các tháng trong năm.
- **Cấu hình:** Các giá trị cấu hình của ứng dụng có thể được lưu trữ dưới dạng tuple để đảm bảo chúng không bị thay đổi vô tình.

2. Làm khóa cho dictionary:

- Vì tuple là bất biến nên chúng có thể được sử dụng làm khóa trong dictionary. Điều này rất hữu ích khi bạn muốn ánh xạ nhiều giá trị với nhau.

3. Trả về nhiều giá trị từ một hàm:

- Hàm trong Python chỉ có thể trả về một giá trị. Tuy nhiên, bạn có thể trả về một tuple chứa nhiều giá trị để mô phỏng việc trả về nhiều kết quả.

4. Định nghĩa các điểm tọa độ:

- Tuple thường được sử dụng để biểu diễn các điểm tọa độ trong không gian 2D hoặc 3D.

5. Xử lý dữ liệu:

- Tuple có thể được sử dụng để lưu trữ các bản ghi dữ liệu có cấu trúc cố định. Ví dụ: một bản ghi nhân viên có thể được biểu diễn dưới dạng một tuple chứa (tên, tuổi, ngày sinh).

Ưu điểm của tuple:

- **Hiệu suất:** Tuple thường nhanh hơn list vì chúng là bất biến.

- **An toàn:** Việc tuple không thể thay đổi giúp tránh được các lỗi không mong muốn khi làm việc với dữ liệu.
- **Dễ đọc:** Cấu trúc của tuple rõ ràng và dễ hiểu.

Khi nào nên sử dụng tuple:

- Khi bạn cần một cấu trúc dữ liệu không thể thay đổi.
- Khi bạn cần một cấu trúc dữ liệu để lưu trữ các giá trị liên quan đến nhau.
- Khi bạn cần một khóa duy nhất cho một dictionary.
- Khi bạn muốn trả về nhiều giá trị từ một hàm.