**1/Phân tích toán tử định dạng chuỗi và hàm định dạng chuỗi có sẵn trong gói thư viện chuẩn Python? Cho 5 ví dụ**

Trong Python, việc định dạng chuỗi là một công việc thường xuyên gặp phải. Nó cho phép chúng ta chèn các giá trị biến vào trong một chuỗi văn bản theo cách có cấu trúc và dễ đọc. Python cung cấp nhiều cách để định dạng chuỗi, bao gồm:

* **Toán tử %:** Cách thức định dạng chuỗi truyền thống.
* **Phương thức format():** Cách thức linh hoạt hơn và thường được ưu tiên sử dụng.
* **f-strings:** Cách thức mới nhất, xuất hiện từ Python 3.6, rất tiện lợi và hiệu quả.

**Toán tử % (Cổ điển)**

* **Cú pháp:** chuỗi % (giá\_trị1, giá\_trị2, ...)
* **Cách hoạt động:**
  + Phần bên trái toán tử % là chuỗi mẫu, chứa các dấu % để đánh dấu vị trí cần chèn giá trị.
  + Phần bên phải toán tử % là một tuple chứa các giá trị cần chèn.
  + Các dấu % trong chuỗi mẫu sẽ được thay thế tuần tự bởi các giá trị trong tuple.
* **Ví dụ:**

Python

name = "Alice"

age = 30

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

Hãy [thận trọng](/faq#coding) khi sử dụng các đoạn mã.

**Phương thức format()**

* **Cú pháp:** chuỗi.format(giá\_trị1, giá\_trị2, ...)
* **Cách hoạt động:**
  + Phần bên trong dấu ngoặc nhọn {} trong chuỗi mẫu đại diện cho vị trí cần chèn giá trị.
  + Các giá trị trong phương thức format() sẽ được chèn vào các vị trí tương ứng.
  + Có thể sử dụng chỉ số hoặc tên để xác định vị trí chèn.
* **Ví dụ:**

Python

print("Hello, my name is {} and I am {} years old.".format(name, age))

print("The value of pi is {:.2f}".format(3.14159))

Hãy [thận trọng](/faq#coding) khi sử dụng các đoạn mã.

**f-strings (Chuỗi f)**

* **Cú pháp:** f"chuỗi {biến}"
* **Cách hoạt động:**
  + Bắt đầu bằng chữ f, sau đó là chuỗi cần định dạng.
  + Các biểu thức Python có thể được đặt bên trong dấu ngoặc nhọn {}.
  + Giá trị của biểu thức sẽ được chèn vào chuỗi.
* **Ví dụ:**

Python

print(f"Hello, my name is {name} and I am {age} years old.")

Hãy [thận trọng](/faq#coding) khi sử dụng các đoạn mã.

**So sánh**

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Ưu điểm** | **Nhược điểm** |
| Toán tử % | Dễ hiểu, đơn giản | Cú pháp hơi cứng nhắc, ít linh hoạt |
| Phương thức format() | Linh hoạt, hỗ trợ nhiều kiểu định dạng | Cú pháp dài hơn, có thể phức tạp với nhiều giá trị |
| f-strings | Ngắn gọn, dễ đọc, hiệu quả | Chỉ có từ Python 3.6 trở lên |

Xuất sang Trang tính

**Khi nào nên sử dụng phương thức nào?**

* **Toán tử %:** Sử dụng cho các trường hợp đơn giản, không cần nhiều định dạng phức tạp.
* **Phương thức format():** Sử dụng khi cần định dạng phức tạp, nhiều tùy chọn.
* **f-strings:** Sử dụng cho các trường hợp cần định dạng nhanh chóng, dễ đọc.

**Nói chung, f-strings là lựa chọn tốt nhất cho hầu hết các trường hợp sử dụng hiện nay.**

**Các tính năng nâng cao**

* **Định dạng số:** Kiểm soát số chữ số thập phân, căn lề, dấu phân cách hàng ngàn.
* **Định dạng ngày tháng:** Hiển thị ngày tháng theo các định dạng khác nhau.
* **Định dạng tùy chỉnh:** Tạo các định dạng riêng bằng cách sử dụng các thư viện như string.

**Ví dụ**

Python

# Định dạng số

print(f"Pi: {3.14159:.2f}") # In ra 2 chữ số thập phân

# Định dạng ngày tháng

import datetime

today = datetime.date.today()

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

# Định dạng tùy chỉnh

print("{:<10s} {:>10s}".format("Name", "Age")) # Căn trái, căn phải

print("{:<10s} {:>10d}".format(name, age))

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

Giả sử ch múng ta có một đoạn văn bản và muốn lấy ngẫu nhiên một từ trong đó. Dưới đây là một ví dụ:

Python

import random

def random\_word(text):

"""Lấy ngẫu nhiên một từ trong một đoạn văn bản

Args:

text: Đoạn văn bản

Returns:

Một từ ngẫu nhiên từ đoạn văn bản

"""

words = text.split() # Tách đoạn văn thành các từ

random\_index = random.randint(0, len(words) - 1) # Lấy một chỉ số ngẫu nhiên

random\_word = words[random\_index]

return random\_word

# Ví dụ sử dụng

text = "Đây là một đoạn văn bản mẫu để thử nghiệm chương trình."

random\_word = random\_word(text)

print(random\_word)

**3/ Khác biệt cơ bản giữa list và tuple?**

**List** và **tuple** là hai kiểu dữ liệu được sử dụng để lưu trữ một tập hợp các giá trị trong Python. Tuy nhiên, chúng có những đặc điểm khác biệt quan trọng:

**1. 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 một list.
* **Tuple:** Không thể thay đổi sau khi được tạo. Các phần tử của một tuple được cố định.

**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:**
  + Lưu trữ các tập hợp dữ liệu có thể thay đổi kích thước.
  + Làm việc với các cấu trúc dữ liệu phức tạp như stack, queue, danh sách liên kết.
  + Đưa vào các hàm làm đối số để thay đổi giá trị bên trong hàm.
* **Tuple:**
  + Lưu trữ các dữ liệu không thay đổi, như các hằng số, các cặp khóa-giá trị trong dictionary.
  + Trả về nhiều giá trị từ một hàm.
  + Làm key cho dictionary (vì tuple là bất biến).

**Ví dụ:**

Python

# List

my\_list = [1, 2, 3, "apple", "banana"]

my\_list.append(4) # Thêm phần tử vào list

print(my\_list) # Output: [1, 2, 3, 'apple', 'banana', 4]

# Tuple

my\_tuple = (1, 2, 3)

# my\_tuple[0] = 4 # Sẽ gây ra lỗi vì tuple không thể thay đổi

print(my\_tuple) # Output: (1, 2, 3)

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

* **Sử dụng list khi:**
  + Bạn cần một cấu trúc dữ liệu linh hoạt, có thể thay đổi.
  + Bạn muốn thực hiện các thao tác thêm, xóa, sửa đổi phần tử.
* **Sử dụng tuple khi:**
  + Bạn cần một cấu trúc dữ liệu không thay đổi, đảm bảo tính toàn vẹn của dữ liệu.
  + Bạn muốn sử dụng làm key cho dictionary.
  + Bạn muốn trả về nhiều giá trị từ một hàm.

**4/Ứng dụng kiểu dữ liệu tuple trong thực tế**

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

**1. Lưu trữ dữ liệu không đổi:**

* **Tọa độ:** Ví dụ: (x, y) để biểu diễn một điểm trên mặt phẳng.
* **Kích thước:** Ví dụ: (width, height) để biểu diễn kích thước của một hình ảnh.
* **Màu sắc:** Ví dụ: (R, G, B) để biểu diễn một màu theo mô hình RGB.
* **Ngày tháng:** Ví dụ: (year, month, day) để biểu diễn một ngày.

**2. Làm khóa cho dictionary:**

Vì tuple là bất biến, nên nó có thể được sử dụng làm khóa trong một dictionary. Điều này rất hữu ích khi bạn muốn ánh xạ một tập hợp các giá trị với một khóa phức tạp.

Python

person = {

('Alice', 30): 'New York',

('Bob', 25): 'Los Angeles'

}

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

Một hàm trong Python chỉ có thể trả về một giá trị. Tuy nhiên, bằng cách sử dụng tuple, bạn có thể trả về nhiều giá trị cùng một lúc.

Python

def get\_user\_info():

name = "Alice"

age = 30

return name, age

name, age = get\_user\_info()

**4. Định nghĩa các hằng số:**

Vì tuple không thể thay đổi, nên nó thường được sử dụng để định nghĩa các hằng số trong chương trình.

Python

DAYS\_IN\_WEEK = (7,)

MONTHS\_IN\_YEAR = (12,)

**5. Unpacking tuple:**

Bạn có thể gán các phần tử của một tuple cho các biến riêng biệt.

Python

point = (3, 4)

x, y = poi

**6. Các ứng dụng khác:**

* **Đại diện cho một tập hợp các lựa chọn:** Ví dụ, trong một menu, mỗi món ăn có thể được biểu diễn bằng một tuple chứa tên món ăn và giá cả.
* **Truyền nhiều đối số cho một hàm:** Bạn có thể đóng gói các đối số thành một tuple và truyền cho hàm.

**Ưu điểm khi sử dụng tuple:**

* **Hiệu suất:** Tuple thường nhanh hơn list vì chúng không thể thay đổi, nên Python có thể tối ưu hóa việc sử dụng chúng.
* **Tính rõ ràng:** Tuple giúp code của bạn dễ đọc hơn bằng cách thể hiện rõ ý định rằng dữ liệu không nên bị thay đổi.
* **An toàn:** Việc sử dụng tuple giúp tránh được các lỗi không mong muốn do vô tình thay đổi giá trị của dữ liệu.