**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?**

|  |  |  |
| --- | --- | --- |
|  | Toán tử định dạng chuỗi (‘%’) | Hàm định dạng chuỗi (str.format()) |
| Khái niệm | Toán tử % là phương pháp truyền thống để định dạng chuỗi trong Python, tương tự như trong ngôn ngữ C. Phương pháp này sử dụng ký tự % để chèn các giá trị vào chuỗi. | Hàm str.format() được giới thiệu trong Python 2.6 và 3.0, cung cấp cách linh hoạt hơn để định dạng chuỗi. Sử dụng dấu ngoặc nhọn {} để chèn giá trị vào chuỗi. |
| Đặc điểm | - **Cú pháp**: Sử dụng ký tự % và các ký tự định dạng (s, d, f, v.v.) theo sau.  - **Linh hoạt**: Ít linh hoạt hơn, đặc biệt khi định dạng chuỗi phức tạp hoặc khi làm việc với nhiều loại dữ liệu.  - **Khuyến khích**: Được coi là phương pháp cũ và ít được khuyến khích trong mã nguồn mới. | **- Cú pháp**: Sử dụng {} để chèn giá trị, với khả năng chỉ định chỉ số vị trí hoặc tên đối số.  - **Linh hoạt**: Hỗ trợ nhiều kiểu định dạng và tùy chọn hơn, phù hợp với chuỗi phức tạp và nhiều loại dữ liệu.  - **Khuyến khích**: Được khuyến khích sử dụng trong mã nguồn mới trước khi f-strings xuất hiện. |
| 5 ví dụ | **Ví dụ 1: định dạng chuỗi**  name = "Alice"  print("Hello, %s!" % name)  # Output: Hello, Alice!  **Ví dụ 2: định dạng số nguyên**  age = 30  print("You are %d years old." % age)  # Output: You are 30 years old  .  **Ví dụ 3: định dạng số thực**  pi = 3.14159  print("Pi is approximately %.2f." % pi)  # Output: Pi is approximately 3.14.  **Ví dụ 4: định dạng nhiều giá trị**  name = "Bob"  age = 25  height = 175.5  print("Name: %s, Age: %d, Height: %.1f cm" % (name, age, height))  # Output: Name: Bob, Age: 25, Height: 175.5 cm  **Ví dụ 5:định dạng nhiều từ điển**  data = {'name': 'Charlie', 'age': 35}  print("Name: %(name)s, Age: %(age)d" % data)  # Output: Name: Charlie, Age: 35 | **Ví dụ 1: Sử dụng chỉ số vị trí**  print("First: {}, Second: {}, Third: {}".format('one', 'two', 'three'))  # Output: First: one, Second: two, Third: three  **Ví dụ 2: Sử dụng nhiều biến**  name = "Bob"  age = 25  print("Name: {}, Age: {}".format(name, age))  # Output: Name: Bob, Age: 25  **Ví dụ 3: Định dạng số thực với độ chính xác**  pi = 3.14159  print("Pi is approximately {:.2f}.".format(pi))  # Output: Pi is approximately 3.14.  **Ví dụ 4: Sử dụng tên đối số**  print("Name: {name}, Age: {age}".format(name="Charlie", age=35))  # Output: Name: Charlie, Age: 35  **Ví dụ 5: Sử dụng chỉ số và tên đối số**  name = "Alice"  age = 30  height = 165.5  print("Name: {0}, Age: {1}, Height: {2:.1f} cm".format(name, age, height))  # Output: Name: Alice, Age: 30, Height: 165.5 cm |

**Câu 3 Viết chương trình xuất ra số ngẫu nhiên trong một đoạn bất kỳ bất cho trước?**

**List**:

* **Sửa đổi**: List là kiểu dữ liệu có thể thay đổi (mutable), tức là bạn có thể thay đổi các phần tử của list sau khi tạo.
* **Cú pháp**: Được khai báo bằng dấu ngoặc vuông [].
* **Ví dụ**: my\_list = [1, 2, 3, 4]
* **Phương thức**: Có nhiều phương thức hỗ trợ thay đổi như .append(), .remove(), .extend(), v.v.
* **Hiệu suất:** List:Chậm hơn và tốn nhiều bộ nhớ hơn.

**Tuple**:

* **Không sửa đổi**: Tuple là kiểu dữ liệu không thay đổi (immutable), tức là bạn không thể thay đổi các phần tử sau khi tạo.
* **Cú pháp**: Được khai báo bằng dấu ngoặc đơn ().
* **Ví dụ**: my\_tuple = (1, 2, 3, 4)
* **Phương thức**: Hạn chế hơn, không có các phương thức thay đổi, chỉ có các phương thức đọc như .count() và .index().
* **Hiệu suất:** Tuple:Nhanh hơn và chiếm ít bộ nhớ hơn.

**Câu 4** **Ứngdụng kiểu dữ liệu tuple trong thực tế?**

**Ứng dụng của kiểu dữ liệu tuple trong thực tế:**

* **Bất biến (Immutability)**: Tuple thường được sử dụng trong các trường hợp bạn cần đảm bảo rằng dữ liệu không bị thay đổi. Ví dụ, tuple có thể dùng để làm khóa trong từ điển (dictionary) vì tuple không thay đổi trong khi list thì có thể thay đổi.
* **Lưu trữ dữ liệu không thay đổi**: Tuple được sử dụng để lưu trữ dữ liệu không thay đổi, như là các điểm tọa độ (x, y) hoặc các cặp dữ liệu trong các cấu trúc dữ liệu khác.
* **Nhóm giá trị**: Tuple thường dùng để nhóm các giá trị liên quan, ví dụ, kết quả trả về từ một hàm có thể là tuple chứa nhiều giá trị, như (status, data).
* **Lưu trữ dữ liệu cấu trúc**: Tuple thường được sử dụng trong các cấu trúc dữ liệu phức tạp hơn như các bản ghi (records) vì tuple cho phép lưu trữ nhiều loại dữ liệu khác nhau cùng một lúc.
* **Sử dụng trong các thư viện và API**: Nhiều thư viện và API Python sử dụng tuple để trả về nhiều giá trị từ các hàm, ví dụ, os.path.splitext() trả về một tuple gồm tên tệp và phần mở rộng.