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

* Toán tử ‘%’ được sử dụng để dịnh dạng mọt tập hợp các biến được đặt trong một “tuple”, cùng với đối số là các ký hiệu đặc biệt như ‘%s’ , ‘%d’ , ‘%f’ , ‘%,nf’:
* %s – Chuỗi
* %d – Số nguyên
* %f – Số thực
* %.nf – Số thực với độ chính xác n

VD1: Định dạng chuỗi và số

name = "Alice"

age = 30

# Sử dụng toán tử `%`

formatted\_string = "Name: %s, Age: %d" % (name, age)

print(formatted\_string)

VD2: Định dạng số thực với số chữ số thập phân

pi = 3.14159

formatted\_string = "Value of pi: %.2f" % pi

print(formatted\_string) # Output: Value of pi: 3.14

VD3: Định dạng số nguyên với độ rộng nhất định

number = 42

formatted\_string = "Number: %5d" % number

print(formatted\_string) # Output: Number: 42

VD4: Định dạng chuỗi với độ rộng nhất định và căn giữa

text = "Hello"

formatted\_string = "Text: %10s" % text

print(formatted\_string) # Output: Text: Hello

VD5: Định dạng nhiều giá trị

name = "Bob"

age = 25

height = 175.5

formatted\_string = "Name: %s, Age: %d, Height: %.1f" % (name, age, height)

print(formatted\_string) # Output: Name: Bob, Age: 25, Height: 175.5

* ‘.format()’: được giới thiệu trong python 3.0, linh hoạt hơn so với ‘%’

VD1: Định dạng chuỗi và số

name = "Alice"

age = 30

formatted\_string = "Name: {}, Age: {}".format(name, age)

print(formatted\_string) # Output: Name: Alice, Age: 30

VD2: Định dạng số thực với số chữ số thập phân

pi = 3.14159

formatted\_string = "Value of pi: {:.2f}".format(pi)

print(formatted\_string) # Output: Value of pi: 3.14

VD3: Định dạng số nguyên với độ rộng nhất định

number = 42

formatted\_string = "Number: {:5d}".format(number)

print(formatted\_string) # Output: Number: 42

VD4: Định dạng chuỗi với độ rộng nhất định và căn giữa

text = "Hello"

formatted\_string = "Text: {:>10}".format(text)

print(formatted\_string) # Output: Text: Hello

VD5: Định dạng nhiều giá trị

name = "Bob"

age = 25

height = 175.5

formatted\_string = "Name: {}, Age: {}, Height: {:.1f}".format(name, age, height)

print(formatted\_string) # Output: Name: Bob, Age: 25, Height: 175.5

* Nội suy chuỗi, f-strings: các chuỗi bắt đầu bằng f và chúng ta có thể đưa dữ liệu vào các vị trí tương ứng trong chuỗi.

VD1: Định dạng chuỗi và số

name = "Alice"

age = 30

formatted\_string = f"Name: {name}, Age: {age}"

print(formatted\_string) # Output: Name: Alice, Age: 30

VD2: Định dạng số thực với số chữ số thập phân

pi = 3.14159

formatted\_string = f"Value of pi: {pi:.2f}"

print(formatted\_string) # Output: Value of pi: 3.14

VD3: Định dạng số nguyên với độ rộng nhất định

number = 42

formatted\_string = f"Number: {number:5d}"

print(formatted\_string) # Output: Number: 42

VD4: Định dạng chuỗi với độ rộng nhất định và căn giữa

text = "Hello"

formatted\_string = f"Text: {text:>10}"

print(formatted\_string) # Output: Text: Hello

VD5: Định dạng nhiều giá trị

name = "Bob"

age = 25

height = 175.5

formatted\_string = f"Name: {name}, Age: {age}, Height: {height:.1f}"

print(formatted\_string) # Output: Name: Bob, Age: 25, Height: 175.5

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

import random

def get\_random\_number\_in\_range():

# Nhập giới hạn dưới và trên từ người dùng

try:

lower\_bound = int(input("Nhập giới hạn dưới của đoạn số: "))

upper\_bound = int(input("Nhập giới hạn trên của đoạn số: "))

# Kiểm tra nếu giới hạn dưới lớn hơn hoặc bằng giới hạn trên

if lower\_bound >= upper\_bound:

print("Giới hạn dưới phải nhỏ hơn giới hạn trên.")

return

# Sinh số ngẫu nhiên trong đoạn [lower\_bound, upper\_bound)

random\_number = random.randint(lower\_bound, upper\_bound - 1)

# Xuất số ngẫu nhiên

print(f"Số ngẫu nhiên trong đoạn [{lower\_bound}, {upper\_bound}): {random\_number}")

except ValueError:

print("Vui lòng nhập số nguyên hợp lệ.")

# Gọi hàm

get\_random\_number\_in\_range()

Câu 3. Khác biệt cơ bản giữa list và tuple

* Khả năng thay đổi
* **List**: 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 nó đã được tạo ra.
* **Tuple**: Là kiểu dữ liệu không thể thay đổi (immutable). Khi một tuple được tạo ra, bạn không thể thay đổi các phần tử của nó.
* Cú pháp
* List sử dụng hàm tích hợp list() và []
* Tuple sử dụng ()
* Phương thức
* **List**: Cung cấp nhiều phương thức như .append(), .remove(), .extend(), .pop(), và .sort() để thay đổi danh sách.
* **Tuple**: Có ít phương thức hơn vì không thể thay đổi. Các phương thức phổ biến bao gồm .count() và .index().

Câu 4. ứng dụng tuple trong thực tế

**Sử dụng làm khóa trong từ điển**

Tuple có thể được sử dụng làm khóa trong từ điển (dictionary) vì chúng là immutable và hashable. Điều này không thể làm với danh sách (list).

**Ví dụ:**

python

Copy code

# Tạo một từ điển với tuple làm khóa

coordinates = {

(10, 20): "Location A",

(30, 40): "Location B"

}

# Truy cập giá trị bằng cách sử dụng tuple làm khóa

print(coordinates[(10, 20)]) # Output: Location A

**2. Nhóm dữ liệu không thay đổi**

Khi bạn muốn nhóm một số giá trị liên quan mà không cần thay đổi chúng sau khi tạo ra, tuple là sự lựa chọn lý tưởng. Ví dụ, khi trả về nhiều giá trị từ một hàm.

**Ví dụ:**

python

Copy code

def get\_user\_info():

return ("Alice", 30, "Engineer")

# Nhận dữ liệu từ hàm

name, age, profession = get\_user\_info()

print(name) # Output: Alice

print(age) # Output: 30

print(profession) # Output: Engineer

**3. Dùng làm giá trị mặc định cho tham số**

Tuple có thể được sử dụng để xác định giá trị mặc định cho các tham số trong hàm, đặc biệt khi có nhiều giá trị mặc định cần phải khớp.

**Ví dụ:**

python

Copy code

def display\_info(name, age, details=("N/A", "Unknown")):

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

display\_info("Bob", 25) # Output: Name: Bob, Age: 25, Details: ('N/A', 'Unknown')

**4. Đảm bảo dữ liệu không bị thay đổi**

Khi cần đảm bảo rằng dữ liệu không bị thay đổi trong suốt quá trình thực hiện của chương trình, tuple là một lựa chọn tốt.

**Ví dụ:**

python

Copy code

# Thay vì sử dụng danh sách

immutable\_data = (1, 2, 3, 4, 5)

# Đảm bảo rằng dữ liệu không bị thay đổi

**5. Sử dụng trong các cấu trúc dữ liệu phức tạp**

Tuple thường được sử dụng trong các cấu trúc dữ liệu phức tạp như cây, đồ thị, hoặc các cấu trúc dữ liệu cần tính ổn định về dữ liệu.

**Ví dụ:**

python

Copy code

# Cấu trúc dữ liệu cây

tree = ((1, 2), (3, (4, 5)))

# Duyệt qua cấu trúc dữ liệu cây

def traverse\_tree(tree):

if isinstance(tree, tuple):

for subtree in tree:

traverse\_tree(subtree)

else:

print(tree)

traverse\_tree(tree) # Output: 1 2 3 4 5

**6. Trả về nhiều giá trị từ hàm**

Khi một hàm cần trả về nhiều giá trị, tuple là một cách thuận tiện để nhóm các giá trị đó lại.

**Ví dụ:**

python

Copy code

def min\_max(numbers):

return (min(numbers), max(numbers))

result = min\_max([1, 2, 3, 4, 5])

print(result) # Output: (1, 5)