**ASSIGNMENT4**

**TOPIC NAME: TUPLES**

• **Introduction to tuples, immutability**

**TUPLES:**

Python Tuple is a collection of objects separated by commas. A tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

EXAMPLE:

**t = (10, 20, 30)**

**print(t)**

**print(type(t))**

**OUTPUT:**

**OUTPUT:**

(10, 20, 30)

<class 'tuple'>

**Immutability in Python**

Immutability in Python refers to an object's inability to be modified after it has been created. Immutable objects cannot have their content or state changed once they are assigned. If you attempt to modify an immutable object, a new object will be created instead.

**Examples of Immutable Objects in Python**

1. **Numbers**: int, float, complex

**Example:**

x = 10

x = x + 1 # A new integer object is created, and x now refers to it.

1. **Strings**:

**Example:**

s = "hello"

s = s + " world" # A new string object is created since strings are immutable.

1. **Tuples**:

**Example**:

t = (1, 2, 3)

t[0] = 4 # This will raise a TypeError because tuples are immutable.

**Why Use Immutable Objects?**

1. **Predictability**: Immutable objects ensure that their state cannot be changed unintentionally, making them safer to use in concurrent or multi-threaded applications.
2. **Hashability**: Immutable objects, like strings and tuples, can be used as keys in dictionaries because their hash values remain constant.
3. **Optimization**: Immutable objects can be stored and reused by Python to save memory, especially for frequently used objects (e.g., integers and strings).

**Creating and accessing elements in a tuple**

### ****Creating a Tuple****

A tuple is an immutable sequence of elements in Python. It can hold elements of different data types (e.g., integers, strings, lists).

#### ****Syntax****:

tuple\_name = (element1, element2, ...)

#### ****Examples****:

1. Empty Tuple:

Example:

my\_tuple = ()

print(my\_tuple) # Output: ()

1. Tuple with Multiple Elements:

Example:

my\_tuple = (1, 2, 3, "apple", 4.5)

print(my\_tuple) # Output: (1, 2, 3, 'apple', 4.5)

1. Single Element Tuple: You must include a trailing comma after the element to create a single-element tuple:

Example

my\_tuple = (5,)

print(my\_tuple) # Output: (5,)

1. Without Parentheses (Tuple Packing):

Example:

my\_tuple = 1, 2, 3

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

### ****Accessing Elements in a Tuple****

You can access tuple elements using **indexing** and **slicing**.

#### ****Indexing****:

* Indexing starts at 0 for the first element.
* Use negative indices to access elements from the end.

my\_tuple = (10, 20, 30, 40, 50)

# Access elements by index

print(my\_tuple[0]) # Output: 10 (first element)

print(my\_tuple[-1]) # Output: 50 (last element)

#### ****Slicing****:

* Use the syntax tuple[start:end:step] to access a subset of the tuple.
* The start index is inclusive, and the end index is exclusive.

Example:

my\_tuple = (10, 20, 30, 40, 50)

# Access a slice of the tuple

print(my\_tuple[1:4]) # Output: (20, 30, 40)

print(my\_tuple[:3]) # Output: (10, 20, 30)

print(my\_tuple[::2]) # Output: (10, 30, 50)

#### ****Iterating Over a Tuple****:

Example:

my\_tuple = ("a", "b", "c")

for item in my\_tuple:

print(item)

# Output:

# a

# b

# c

### ****Examples with Mixed Data Types****

my\_tuple = (1, "apple", [10, 20, 30], 3.14)

# Accessing individual elements

print(my\_tuple[1]) # Output: apple

# Accessing a nested element (e.g., from a list inside the tuple)

print(my\_tuple[2][1]) # Output: 20

### ****Key Characteristics****

1. **Immutable**: Once created, the elements of a tuple cannot be changed.
2. **Ordered**: Elements are stored in a specific sequence and can be accessed using indices.
3. **Allows Duplicates**: Tuples can store duplicate elements

### Basic operations with tuples: concatenation, repetition, membership.

### ****1. Concatenation****

Concatenation combines two or more tuples into one using the + operator.

#### ****Syntax****:

tuple1 + tuple2

#### ****Example****:

tuple1 = (1, 2, 3)

tuple2 = (4, 5, 6)

result = tuple1 + tuple2

print(result) # Output: (1, 2, 3, 4, 5, 6)

### ****2. Repetition****

Repetition creates a new tuple by repeating the elements of an existing tuple using the \* operator.

#### ****Syntax****:

tuple \* n

#### ****Example****:

my\_tuple = (1, 2, 3)

result = my\_tuple \* 3

print(result) # Output: (1, 2, 3, 1, 2, 3, 1, 2, 3)

### ****3. Membership****

Membership checks whether a specific element exists in a tuple using the in and not in operators.

#### ****Syntax****:

element in tuple

element not in tuple

#### ****Example****:

my\_tuple = (1, 2, 3, 4, 5)

# Check membership

print(3 in my\_tuple) # Output: True

print(10 not in my\_tuple) # Output: True

### ****4. Other Useful Operations****

#### ****Tuple Length (****len()****)****:

Gets the number of elements in a tuple.

my\_tuple = (1, 2, 3)

print(len(my\_tuple)) # Output: 3

#### ****Maximum (****max()****) and Minimum (****min()****)****:

Finds the largest and smallest elements in a tuple (only works if the tuple contains comparable elements, e.g., numbers or strings).

my\_tuple = (1, 2, 3, 4, 5)

print(max(my\_tuple)) # Output: 5

print(min(my\_tuple)) # Output: 1

#### ****Counting Elements (****tuple.count()****)****:

Counts the occurrences of a specific value.

my\_tuple = (1, 2, 2, 3, 4, 2)

print(my\_tuple.count(2)) # Output: 3

#### ****Finding Index (****tuple.index()****)****:

Returns the index of the first occurrence of a specific value.

my\_tuple = (1, 2, 3, 4, 5)

print(my\_tuple.index(3)) # Output: 2

### ****Example Using All Operations****

tuple1 = (1, 2, 3)

tuple2 = (4, 5, 6)

# Concatenation

concat\_tuple = tuple1 + tuple2

print("Concatenated Tuple:", concat\_tuple) # Output: (1, 2, 3, 4, 5, 6)

# Repetition

repeated\_tuple = tuple1 \* 2

print("Repeated Tuple:", repeated\_tuple) # Output: (1, 2, 3, 1, 2, 3)

# Membership

print(2 in tuple1) # Output: True

print(10 not in tuple2) # Output: True

# Length

print("Length of Tuple:", len(concat\_tuple)) # Output: 6

# Count and Index

print("Count of 2:", repeated\_tuple.count(2)) # Output: 2

print("Index of 4:", concat\_tuple.index(4)) # Output: 3