**Module 7) Python – Collections, functions and Modules**

* Accessing List

1. Understanding how to create and access elements in a list.

* Lists can be created using square brackets []. Elements are accessed using their index, starting from 0.

1. Indexing in lists (positive and negative indexing).

* Positive indexing starts from 0, and negative indexing starts from -1 (last element).

1. Slicing a list: accessing a range of elements.

* Slicing is used to access a range of elements: list[start:stop:step].
* List Operations

1. Common list operations: concatenation, repetition, membership.

* **Concatenation:**
* Combines two or more lists into one.
* Syntax: list3 = list1 + list2
* **Ex** :

list1 = [1, 2, 3]

list2 = [4, 5, 6]

combined = list1 + list2

print(combined) # Output: [1, 2, 3, 4, 5, 6]

* **Repetition:**
* Repeats the elements of a list a specified number of times.
* Syntax: list2 = list1 \* n
* Ex:

list1 = [1, 2, 3]

repeated = list1 \* 2

print(repeated) # Output: [1, 2, 3, 1, 2, 3]

* **Membership:**
* Checks whether an element exists in a list.
* **Ex:**
* list1 = [1, 2, 3]
* print(2 in list1) # Output: True
* print(4 in list1) # Output: False

1. Understanding list methods like append(), insert(), remove(), pop().

* append(): Adds an element to the end of the list.
* insert(): Inserts an element at a specific index.
* remove(): Removes the first occurrence of an element.
* pop(): Removes an element at a specific index.
* Working with Lists

1. Iterating over a list using loops.

* Iterating means going through each element in the list to perform operations like printing, modifying, or summing up elements. This can be done using **for loops** or **while loops**.
* **For Loop**: It is commonly used to iterate over all elements in the list.

1. Sorting and reversing a list using sort(), sorted(), and reverse().

* sort(): Sorts in-place.
* sorted(): Returns a new sorted list.
* reverse(): Reverses the list in-place.

1. Basic list manipulations: addition, deletion, updating, and slicing.

* **Addition**:
* Refers to adding new elements to a list. This can be done in two ways:
  + Using **append()**: Adds a single element to the end of the list.
  + Using **extend()**: Adds multiple elements to the list by extending it.
  + Using **concatenation (+)**: Combines two or more lists into one.
* **Deletion**:
* Refers to removing elements from a list. Common methods include:
  + **remove()**: Removes the first occurrence of a specific element.
  + **pop()**: Removes an element at a specific index and returns it.
  + **del**: Deletes an element or a slice of elements.
  + **clear()**: Removes all elements from the list.
* **Updating**:
* Refers to modifying an existing element in the list by accessing it via its index.
* Syntax: list[index] = new\_value
* **Slicing**:
* Refers to extracting a portion of the list. Slicing is done using the syntax:
  + list[start:stop:step]
  + start: Starting index (inclusive).
  + stop: Ending index (exclusive).
  + step: Step size (optional).
* Tuples

1. Introduction to tuples, immutability.

* **Tuple** is an ordered collection of elements, similar to a list, but it is **immutable**, meaning its elements cannot be changed, added, or removed after creation.
* Tuples are faster than lists because they are immutable and are often used for fixed collections of data.
* Syntax: Tuples are defined using parentheses ().

1. **Creating and Accessing Elements in a Tuple**

* Tuples can hold multiple data types.
* Individual elements can be accessed using **indexing**.
* Example:

my\_tuple = (1, "hello", 3.14)

print(my\_tuple[0]) # Accessing the first element

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

* **Concatenation**: Combines two tuples into one.

t1 = (1, 2)

t2 = (3, 4)

t3 = t1 + t2

print(t3) # Output: (1, 2, 3, 4)

* **Repetition**: Repeats a tuple a specified number of times

t = (1, 2)

print(t \* 3) # Output: (1, 2, 1, 2, 1, 2)

* **Membership**: Checks if an element exists in a tuple.

t = (1, 2, 3)

print(2 in t) # Output: True

* Accessing Tuples

1. Accessing tuple elements using positive and negative indexing.

* **Positive and Negative Indexing**:
* **Positive Indexing**: Starts from 0 and goes left to right.
* **Negative Indexing**: Starts from -1 and goes right to left.
* Example:

t = (10, 20, 30, 40)

print(t[1]) # Positive indexing, Output: 20

print(t[-1]) # Negative indexing, Output: 40

* **Slicing**:
* Extracts portions of a tuple.
* Syntax: tuple[start:stop:step]
* Example:

t = (10, 20, 30, 40, 50)

print(t[1:4]) # Output: (20, 30, 40)

print(t[::2]) # Output: (10, 30, 50)

1. Slicing a tuple to access ranges of elements.

* **Slicing** is the process of extracting a portion of a tuple by specifying a range of indices.
* Tuples are immutable, so slicing creates a new tuple without modifying the original one.
* Example:

tuple[start:stop:step]

* Dictionaries

1. Introduction to dictionaries: key-value pairs.

* A dictionary is an unordered collection of **key-value pairs**.
* Keys must be unique and immutable, while values can be of any data type.
* Syntax: {key: value}

1. Accessing, adding, updating, and deleting dictionary elements.

* **Accessing**: Use the key to get its value.

d = {"a": 1, "b": 2}

print(d["a"]) # Output: 1

* **Adding**: Assign a new key-value pair.

d["c"] = 3

* **Updating**: Change the value of an existing key

d["a"] = 10

* **Deleting**: Use del or pop() to remove an element

del d["a"]

1. **Dictionary Methods**:

* keys(): Returns all keys.
* values(): Returns all values.
* items(): Returns key-value pairs as tuples.
* Working with Dictionaries

1. Iterating over a dictionary using loops.

* Use a loop to go through keys or key-value pairs.

d = {"a": 1, "b": 2}

for key in d:

print(key, d[key])

2. **Merging Two Lists into a Dictionary:**

* Use the zip() function or a loop.

keys = ["a", "b"]

values = [1, 2]

d = dict(zip(keys, values))

print(d) # Output: {'a': 1, 'b': 2}

3. **Counting Occurrences Using Dictionaries:**

s = "hello"

count = {}

for char in s:

count[char] = count.get(char, 0) + 1

print(count) # Output: {'h': 1, 'e': 1, 'l': 2, 'o': 1}

* Functions

1. Defining functions in Python.

* A function in Python is defined using the def keyword, followed by the function name and parentheses ().
* Ex:

def greet():

print("Hello, World!")

* A function is called using its name followed by ().

2. Different types of functions: with/without parameters, with/without return values.

* Function without parameters and without return value:

def say\_hello():

print("Hello!")

say\_hello()

* Function with parameters but without return value:

def greet(name):

print(f"Hello, {name}!")

greet("Diya")

* Function with parameters and with return value:

def add(a, b):

return a + b

result = add(5, 3)

print(result)

* **Function without parameters but with return value:**

def get\_pi():

return 3.14159

print(get\_pi())

3. Anonymous functions (lambda functions).

* A lambda function is a small, anonymous function defined using the lambda keyword.
* Ex:

square = lambda x: x \* x

print(square(5)) # Output: 25

* Lambda functions are mostly used for short, simple operations.
* Modules

1. Introduction to Python modules and importing modules.

* A **module** in Python is a file containing Python code (functions, classes, or variables) that can be reused in other programs.
* We use the import statement to use a module.

2. Standard library modules: math, random.

* **math Module:** Provides mathematical functions such as sqrt(), ceil(), floor(), etc.
* **random Module:** Used for generating random numbers.

3. Creating Custom Modules:

* We can create a module by writing Python code in a .py file and importing it into another file.
* Example: Create mymodule.py:

def greet(name):

return f"Hello, {name}!"

* Importing the module:

import mymodule

print(mymodule.greet("Diya"))