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

1. Accessing List :-

⇒ Theory :-

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

Ans :-

→ You create a list by placing elements inside square brackets [ ], separated by commas.

→ To access an item in a list, use its index inside square brackets [ ].

- Indexing starts at 0, not 1.

- You can also use negative indexes to start from the end.

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

Ans :-

→ Positive Indexing :-

- Starts from the left (the beginning of the list)

- First element is at index 0

- Second element is at index 1, and so on...

→ Negative Indexing :-

- Starts from the right (the end of the list)

- Last element is at index -1

- Second last is -2, and so on...

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

Ans :-

→ Slicing is a way to access a range of elements from a list by specifying a start index and an end index.

→ It creates a new list with the selected elements.

- start → The index to begin the slice (included)

- stop → The index to stop the slice (not included)

2. List Operations :-

⇒ Theory :-

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

Ans :-

→ Concatenation :-

- Concatenation means joining two or more lists together using the + operator.

- It creates a new list that contains elements from both lists.

→ Repetition :-

- Repetition uses the \* operator to repeat the list elements a certain number of times.

- It returns a new list with repeated elements.

→ Membership

- Membership uses the in or not in operator to check whether an element exists in a list.

- Returns True or False.

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

Ans :-

→ append() :-

- Adds an element to the end of the list.

- The original list is modified.

→ insert() :-

- Adds an element at a specific position in the list.

- Shifts the existing elements to the right.

→ remove() :-

- Removes the first occurrence of a given element.

- If the element is not found, it raises an error.

→ pop() :-

- Removes and returns an element from the list.

- If you don’t give an index, it removes the last item.

3. Working with Lists :-

⇒ Theory :-

1. theory Iterating over a list using loops.

Ans :-

→ Iteration over a list means accessing each element of the list one by one, usually to perform some action like printing, modifying, or evaluating them. This is done using loops control structures that repeat a block of code.

→ Iterate Over a List :-

- To perform operations on each item (e.g., calculations)

- To automate tasks on collections of data

- To avoid repeating code manually for each element

→ for loop (direct iteration)

- Simplest and most commonly used method in Python.

- Automatically accesses each element in order.

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

Ans :-

→ sort() Method :-

- Used for sorting a list in-place (modifies the original list).

- Can sort in ascending or descending order.

- Works only with lists.

→ sorted() Function :-

- Returns a new sorted list, without changing the original.

- Works with any iterable (like lists, tuples, strings, etc.).

- Can sort in ascending or descending order.

→ reverse() Method

- Reverses the order of elements in the list in-place.

- It does not sort the list — it just flips the order of the current elements.

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

Ans :-

Addition - append,index,item,extend

Deletion - remove,pop,del

Updating - index assignment

Slicing - [start:stop]

4. Tuple :-

⇒ Theory :-

1. Introduction to tuples, immutability.

Ans :-

→ A tuple in Python is an ordered, immutable collection of elements. It allows storing multiple values in a single variable, and items can be of different data types (e.g., integers, strings, floats).

→ The key feature of a tuple is immutability, which means:

- Once a tuple is created, its elements cannot be changed, added, or removed.

- Any attempt to modify its contents will result in a TypeError.

→ Effects of Immutability:

- Safety: Data remains unchanged, making it reliable in code where consistency is important.

- Hashability: Tuples can be used as keys in dictionaries and elements of sets (only if they contain hashable types).

- Performance: Tuples are generally faster than lists in iteration and processing due to their fixed size.

→ Use Cases:

- Returning multiple values from a function.

- Representing fixed sets of values (e.g., coordinates, days of the week).

2. Creating and accessing elements in a tuple.

Ans :-