Module 14) Python – Collections, functions and Modules

Accessing List

Theory:

Understanding how to create and access elements in a list.

Indexing in lists (positive and negative indexing).

**Ans -Indexing in Lists**

Each element in a list is assigned a position called an **index**, which helps access elements.

**1. Positive Indexing**

The index starts from 0 for the first element.

The second element is at index 1, and so on.

For example, in the list numbers = [10, 20, 30, 40, 50]:

numbers[0] refers to 10

numbers[1] refers to 20

numbers[4] refers to 50

**2. Negative Indexing**

Python also allows accessing elements from the end using negative indices.

The last element is indexed as -1, the second last as -2, and so on.

For example, in the same list numbers = [10, 20, 30, 40, 50]:

numbers[-1] refers to 50

numbers[-2] refers to 40

numbers[-5] refers to 10

Slicing a list: accessing a range of elements.

**Ans** - Slicing allows access to a specific range of elements in a list using a **start index, end index, and step size**. The start index is **inclusive**, while the end index is **exclusive**.

**Theory:**

•

Common list operations: concatenation, repetition, membership.

**Ans -** 1. Concatenation (+)

Concatenation is the process of joining two or more lists into a single list using the + operator. This creates a new list without modifying the original ones.

**Key Points:**

Works only with lists (not other data types like integers or strings).

Returns a new list containing elements of both lists.

**2. Repetition (\*)**

Repetition is used to create multiple copies of a list using the \* operator.

**Key Points:**

The original list remains unchanged.

The result is a new list where elements are repeated the specified number of times.

**3. Membership (in, not in)**

Membership operators are used to check whether an element exists in a list or not.

**in operator:** Returns True if the element is found in the list.

**not in operator:** Returns True if the element is not present in the list.

**Key Points:**

Helps in searching for values within a list.

Works for different data types like numbers, strings, and lists.

•

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

**Ans-** Python provides several built-in methods to modify and manipulate lists dynamically.

**1. append()**

Adds a single element to the end of the list.

Modifies the original list (does not return a new one).

**2. insert()**

Inserts an element at a specific index.

Shifts existing elements to the right.

**3. remove()**

Removes the first occurrence of a specified element.

Raises an error if the element is not found.

**4. pop()**

Removes and returns an element at a given index.

If no index is provided, removes the last element by default.

**Theory:**

•

Iterating over a list using loops.

**Ans**-Iteration allows accessing each element of a list one by one. This can be done using:

**for loop**: Iterates directly over elements.

**while loop**: Uses indexing to access elements.

**Key Points:**

Efficient for processing all elements in a list.

Commonly used for searching, filtering, and applying transformations.

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

**Ans-** Sorting Methods:

**sort()**:

Sorts the list in-place (modifies the original list).

Default sorting is in ascending order.

Use reverse=True for descending order.

**sorted()**:

Returns a new sorted list without modifying the original list.

Can sort in ascending or descending order using reverse=True.

**Reversing Method:**

**reverse()**: Reverses the order of elements in the list in-place.

**Key Points:**

Sorting is useful for ordering data before processing.

Reversing helps when accessing elements from the end.

**•**

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

**Ans**- **Basic List Manipulations**

**1. Addition**

**append(x)** → Adds x to the end.

**insert(i, x)** → Inserts x at index i.

**Concatenation (+)** → Merges two lists.

**2. Deletion**

**remove(x)** → Removes the first occurrence of x.

**pop(i)** → Removes and returns the element at index i (default: last element).

**del list[i]** → Deletes the element at index i.

**3. Updating**

**list[i] = new\_value** → Updates an element at index i.

**4. Slicing**

**list[start:end:step]** → Extracts a sublist.

**Examples:**

list[:3] → First 3 elements

list[2:] → From index 2 to end

list[::-1] → Reversed list

4. Tuple

**Theory:**

Introduction to tuples, immutability.

Ans -  A **tuple** is an **ordered, immutable** collection of elements.

 Once created, elements **cannot be modified, added, or removed**.

 Tuples are **faster** than lists and can be used as dictionary keys.

•

Creating and accessing elements in a tuple.

**Ans -** . Creating & Accessing Elements

**Syntax:** my\_tuple = (1, "apple", 3.14)

**Accessing:**

**Indexing:** my\_tuple[1] → "apple"

**Negative Indexing:** my\_tuple[-1] → 3.14

Basic operations with tuples: concatenation, repetition, membership.

**Ans -**  **Concatenation (+)** → Combines tuples: (1, 2) + (3, 4) → (1, 2, 3, 4)

 **Repetition (\*)** → Repeats elements: (1, 2) \* 3 → (1, 2, 1, 2, 1, 2)

 **Membership (in, not in)** → Checks existence: "apple" in ("apple", "banana") → True

5. Accessing Tuples

**Theory:**

• Accessing tuple elements using positive and negative indexing.

**Ans - Indexing (Positive & Negative)**

**Positive Indexing (0 to n-1)** → tuple[1] (second element).

**Negative Indexing (-1 to -n)** → tuple[-1] (last element).

Slicing a tuple to access ranges of elements.

**Ans - Slicing (**start:end:step**)**

tuple[1:4] → Elements from index 1 to 3.

tuple[:3] → First 3 elements.

tuple[::-1] → Reversed tuple.

6. Dictionaries

**Theory:**

•

Introduction to dictionaries: key-value pairs.

Ans -  A **dictionary** stores data as **key-value pairs** {key: value}.

 Keys are **unique** and **immutable** (strings, numbers, tuples).

Accessing, adding, updating, and deleting dictionary elements.

Ans - **Accessing:** dict[key] or dict.get(key).

**Adding/Updating:** dict[key] = value.

**Deleting:** del dict[key] or dict.pop(key).

Dictionary methods like keys(), values(), and items().

**Ans-**  keys() → Returns all keys.

 values() → Returns all values.

 items() → Returns key-value pairs.

7. Working with Dictionaries

**Theory:**

•

Iterating over a dictionary using loops.

**Ans -Iterating Over a Dictionary**

Keys: for key in dict:

**Values:** for value in dict.values():

**Key-Value Pairs:** for key, value in dict.items():

•

Merging two lists into a dictionary using loops or zip().

**Ans -**Two lists (one for keys, one for values) can be combined using:

**Looping manually** to assign keys and values.

**Using zip()** to pair elements from both lists into key-value pairs.

Counting occurrences of characters in a string using dictionaries.

**Ans -**Create a dictionary where each key is a character, and its value is the number of times it appears in the string.

8. Functions

**Theory:**

Defining functions in Python.

Ans-- Functions in Python are created using the def keyword and allow code reuse.

**Syntax:** A function has a name, parameters (optional), and a return value (optional).

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

**Ans-**  **Without Parameters & Return Value:** Executes an action but does not return any result.

 **With Parameters & Without Return Value:** Accepts input but does not return a value.

 **With Parameters & Return Value:** Takes inputs, processes them, and returns a result.

 **Without Parameters & With Return Value:** Returns a value without requiring input.

• Anonymous functions (lambda functions)

**Ans-**  Small, single-expression functions using lambda.

 Typically used for short, simple operations where defining a full function is unnecessary.

9. Modules

**Theory:**

Introduction to Python modules and importing modules.

**Ans-**  A **module** is a Python file containing functions, variables, or classes.

 Helps organize code and improve reusability.

**2. Importing Modules**

Use import module\_name to access built-in or custom modules.

Standard library modules: math, random.

**Ans-**  **math Module:** Provides mathematical operations like sqrt(), ceil(), floor().

 **random Module:** Generates random numbers using functions like randint(), choice().

Creating custom modules.

**Ans-**  A custom module is a Python file containing reusable code.

 It can be imported into other Python programs using import module\_name.