**Module-2 Theory**

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

Ans:- In python list is a mutable,ordered sequence of elements. Lists are created by enclosing a comma -separated sequence of item within square bracket, list has a different data types also.

List\_1 = [1, 2, 8, 9, 10, 5]

Fruit= ["apple", "Mango", "Raspberry"]

mixed\_list = [10, "hello", True, 3.14]

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

Ans:- List indexing in Python refers to the process of accessing individual elements within a list using their numerical position, known as an index.

It indicates first element as 0 and second element as 1 and last element -1.

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

Ans:- Slicing a list in Python allows for accessing a range of elements to create a new sub-list. This is achieved using the slice operator, which is the colon (:).

Example:-

Fruits=[10,20,30,40,50,60,70]

A=Fruits[1:5] # start from 1 and not including 4.

A=Fruits[:3] # start from 0 to and not including 3.

A=Fruits[2:] # start from 2 to end of list.

A=fruits[1:7:2] *# Elements from index 0 to 6, taking every second element*

A=fruits[::-1] # Reverse the entire list

1. Common list operations: concatenation, repetition.

Ans:- The + operator, when used with two lists, performs concatenation. This means it combines the elements of the two lists into a single new list. The elements from the list on the left side of the + operator appear first in the new list, followed by the elements from the list on the right side.

Example:-

Fruits=[‘apple’,’banana’,’Mango’,’Raspberry’]

Number=[10,20,25,36]

A=Fruits+Number

Print(A)

Output=[‘apple’,’banana’,’Mango’,’Raspberry’,10,20,25,36]

The \* operator, when used with a list and an integer, performs repetition. It creates a new list by repeating the original list a specified number of times. The integer operand determines how many copies of the original list are concatenated together to form the new list.

Example:-

List\_1=[10,20,30]

A= List\_1 \*3

Print(A)

Output=>[10,20,30,10,20,30,10,20,30]

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

Ans:-

Append(item): This method adds a single item to the end of the list. It modifies the list in-place and does not return a value.

Insert(index, item): This method inserts an item at a specified index within the list. Existing elements from that index onwards are shifted to the right.

remove(value): This method removes the first occurrence of a specified value from the list. If the value is not found, it raises a Value Error.

pop(index=-1): This method removes and returns the element at a specified index. If no index is provided, it removes and returns the last element of the list.

1. Iterating over a list using loops.

Ans:- Using a for loop (most common): This method directly iterates through each element in the list. Examples:-

Hardik=[‘Hardik’,’Jayesh’,’Ramesh’,Divyesh’]

For i in Hardik:

Print(i)

Output:- Hardik

Jayesh

Ramesh

Divyesh

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

Ans:-

Sort:-This is a method of the list object itself.

It sorts the list in-place, meaning it modifies the original list directly and does not return a new list.

By default, it sorts in ascending order.

To sort in descending order, use the reverse=True argument:Listname.sort(reverse=True).

Example:-

Hardik=['Hardik','Jayesh','Ramesh','Divyesh']

Hardik.sort()

print(Hardik)

output= ['Divyesh', 'Hardik', 'Jayesh', 'Ramesh'] Arrange in AtoZ format

**Sorted:-**

This is a built-in Python function that can take any iterable (like a list, tuple, or string) as input.

It returns a new sorted list without modifying the original iterable.

By default, it sorts in ascending order.

To sort in descending order, use the reverse=True argument: sorted(my\_iterable, reverse=True).

Reverse:-

This is a method of the list object.

It reverses the elements of the list in-place, modifying the original list directly and not returning a new list.

1. Basic list manipulations: addition, deletion, updating, and slicing.
2. Introduction to tuples, immutability.
3. Creating and accessing elements in a tuple.
4. Basic operations with tuples: concatenation, repetition, membership.
5. Accessing tuple elements using positive and negative indexing.
6. Slicing a tuple to access ranges of elements.
7. Introduction to dictionaries: key-value pairs.
8. Accessing, adding, updating, and deleting dictionary elements,
9. Dictionary methods like keys(), values(), and items().
10. Iterating over a dictionary using loops.
11. Merging two lists into a dictionary using loops or zip().
12. Counting occurrences of characters in a string using dictionaries.
13. Defining functions in Python.
14. Different types of functions: with/without parameters, with/without return values.
15. Anonymous functions (lambda functions).
16. Introduction to Python modules and importing modules.
17. Standard library modules: math, random.
18. Creating custom modules.
19. Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').
20. Using the open() function to create and access files.
21. Closing files using close().
22. Reading from a file using read(), readline(), readlines().
23. Writing to a file using write() and writelines().
24. Introduction to exceptions and how to handle them using try, except, and finally.
25. Understanding multiple exceptions and custom exceptions.