TOPS TECHNOLOGY

Python – Collections, functions and Modules

Presented By:
Nandni Vala

Working with Lists

1.Iterating over a list using loops

- **Using a for Loop :**
- The simplest way to iterate through a list
- Example :
- fruits = ["apple", "banana", "cherry"]
- for fruit in fruits:
- print(fruit)
- **Output:**
- Apple
- Banana
- cherry

- **Using for Loop with range() :**
- If you need the index of each element, you can combine a for loop with range().
- Example :
- fruits = ["apple", "banana", "cherry"]
- for i in range(len(fruits)):
- print(f"Index {i}: {fruits[i]}")
- **Output:**
- Index 0: apple
- Index 1: banana
- Index 2: cherry

- **Using a while Loop :**
- You can use a while loop for iteration, though it’s less common. This is useful when the stopping condition is dynamic.
- Example :
- fruits = ["apple", "banana", "cherry"]
- i = 0
- while i < len(fruits):
- print(fruits[i])
- i += 1
- **Output:**
- Apple
- Banana
- cherry

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

- **Using sort()**
- The **sort()** method sorts a list **in place** (modifies the original list).
- By default, it sorts in **ascending order**.
- Use the reverse=True parameter for **descending order**.
- # Example 1 :
- numbers = [4, 2, 8, 1]
- numbers.sort()
- print(numbers)
- Output: [1, 2, 4, 8]

- **Using sorted() :**
- The **sorted()** function returns a **new list** that is sorted.
- The original list remains unchanged.
- You can sort in **ascending** or **descending order**.
- Examples:
- numbers = [4, 2, 8, 1]
- sorted_numbers = sorted(numbers)
- print(sorted_numbers)
- Output: [1, 2, 4, 8]

- **Using reverse() :**
- The **reverse()** method reverses the order of the elements in the list **in place**.
- It does not sort the list, but simply reverses the current order.
- Example :
- numbers = [4, 2, 8, 1]
- numbers.reverse()
- print(numbers)
- Output: [1, 8, 2, 4]

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

- **Addition**
- Adding elements to a list can be done using methods like append(), extend(), or by concatenation.
- a) Using append():
- Adds a single element to the **end** of the list.
- Example
- numbers = [1, 2, 3]
- numbers.append(4)
- print(numbers)
- Output: [1, 2, 3, 4]

- b) Using extend():
- Adds multiple elements (from another list or iterable) to the end of the list.
- Example :
- numbers = [1, 2, 3]
- numbers.extend([4, 5, 6])
- print(numbers)
- Output: [1, 2, 3, 4, 5, 6]
- c) Using Concatenation (+):
- Combines two lists and creates a **new list**.
- Example :
- numbers = [1, 2, 3]
- new_numbers = numbers + [4, 5]
- print(new_numbers)
- Output: [1, 2, 3, 4, 5]

- **Deletion**
- You can delete elements using remove(), pop(), del, or clear().
- a) Using remove():
- Removes the first occurrence of a specified value.
- Example :
- numbers = [1, 2, 3, 2]
- numbers.remove(2)
- print(numbers)
- Output: [1, 3, 2]
- b)Using pop():
- Removes an element by **index** and returns it. If no index is provided, it removes the last element.

- Example :
- numbers = [1, 2, 3]
- removed = numbers.pop(1)
- print(removed)
- Output: 2
- c)Using del:
- Deletes an element or a slice of the list.
- numbers = [1, 2, 3, 4]
- del numbers[1]
- print(numbers)
- Output: [1, 3, 4]

- d) Using clear():
- Removes all elements from the list.
- numbers = [1, 2, 3]
- numbers.clear()
- print(numbers)
- Output: []
- **3. Updating**
- You can update list elements by accessing them directly using their index.
- a) Updating a Single Element:
- numbers = [1, 2, 3]
- numbers[1] = 5
- print(numbers)
- Output: [1, 5, 3]

- b) Updating a Slice:
- numbers = [1, 2, 3, 4]
- numbers[1:3] = [8, 9]
- print(numbers)
- Output: [1, 8, 9, 4]
- **Slicing**
- Slicing is used to access a range of elements from a list.
- a) Basic Slicing:
- numbers = [10, 20, 30, 40, 50]
- print(numbers[1:4]) # Output: [20, 30, 40]