3. Working with Lists

- **Conditional statement in python.**

List iteration is the process of sequentially accessing each element in a list. Python provides multiple looping constructs for this purpose, each with specific characteristics and use cases.

Types of List Iteration

1. For Loops (Element-wise Iteration)

Primary Mechanism:

for item in list:

Characteristics:

- Accesses elements directly (not indices).
- Simple and Pythonic.
- Read-only by default (cannot modify list structure during iteration).
- Creates a temporary reference to each element.
- Time complexity: O(n) for complete traversal.

2. **For Loops (Index-based Iteration)**

   Primary Mechanism:
   for i in range(len(list)):

Characteristics:

- Accesses elements via indices.
- Allows modification of elements (list[i] = new_value).
- Enables access to neighboring elements.
- More verbose than element-wise iteration.
- Required when position information is needed.

3. **While Loops**

Primary Mechanism:
while i < len(list):

Characteristics:

- More flexible termination conditions.
- Can modify iteration index manually.
- Risk of infinite loops if not properly managed.
- Useful for complex traversal patterns.
- Often requires manual index management.

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

1. **Sort()**

The sort() method modifies the original list in ascending order (by default). It does not return a new list.

- Sorts the list in-place (modifies the original list).
- By default, it sorts in ascending order.
- You can use reverse=True to sort in descending order.
- reverse = true sorts the list in descending order.

```
list = [5, 3, 9, 1, 7]

list.sort()

print("Sorted list:", list)

# Output: Sorted list: [1, 3, 5, 7, 9]
```

2. sorted()

The sorted() function **returns a new sorted list** without modifying the original list.

- Does not change the original list.
- Default is ascending order.
- Works with any iterable (not just lists).
- More versatile but uses more memory (creates new list)

```
numbers = [4, 1, 3]

new_list = sorted(numbers)



print("Original:", numbers)    # Output: [4, 1, 3]

print("Sorted:", new_list)     # Output: [1, 3, 4]
```

3. **reverse().**

   The reverse() method reverses the order of elements in a list. It does not sort, it only flips the list from back to front.

   - reverse() modifies the original list.
   - It doesn’t sort, just flips the order.
   - Fast and simple if you just need to reverse order.

```
list = ["apple", "banana", "orange"]

list.reverse()

print(list)  # Output: ['orange', 'banana', 'apple']
```

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

1. **Adding Elements to a List**

   Definition:

   Adding elements means inserting new items into the list. You can do this using:

Adds single or multiple items.

Expands the list size.

Maintains order of elements.

- append() – adds at the end.
- Insert() - adds at a specific position.
- Extend() - adds multiple elements.

Example:

```
fruits = ["apple", "banana"]

fruits.append("chickoo")           # Add at end

fruits.insert(1, "mango")         # Add at index 1

fruits.extend(["grape", "orange"])  # Add multiple items


print(fruits)

# Output: ['apple', 'mango', 'banana', 'chickoo', 'grape', 'orange']
```

2. **Deletion (Removing Elements from a List)**

Definition:

Removing elements means deleting one or more items using:

- remove() - deletes by value.
- Pop() - deletes by index.
- del - deletes by index or entire list.

Reduces list size.

Can remove specific value or by position.

```
fruits = ["apple", "banana", "orange", "banana"]

fruits.remove("banana")    # Remove first occurrence

fruits.pop(1)              # Remove element at index 1

del fruits[0]              # Delete by index


print(fruits)

# Output: ['orange']
```

3. Updating (Changing Elements in a List)

- Updating means changing the value of a list item at a specific index.
- Overwrites the old value.
- Accessed by index.
- List must be mutable (which it is in Python).

```
fruits = ["apple", "banana", "orange"]

fruits[1] = "mango"       # Change index 1

print(fruits)

# Output: ['apple', 'mango', 'orange']
```

4. Slicing (Accessing a Range of Elements)

- Method: list[start:stop:step]
- Behavior: Returns new list containing specified elements
- Time Complexity: O(k) for slice of size k
- Modifies Original: No
- Default Values: start=0, stop=len(list), step=1

```
fruits = ["apple", "banana", "cherry", "date", "elderberry"]


print(fruits[1:4])    # Output: ['banana', 'cherry', 'date']

print(fruits[:3])     # Output: ['apple', 'banana', 'cherry']

print(fruits[::2])    # Output: ['apple', 'cherry', 'elderberry']
```