**ASSIGNMENT3 : WORKING WITH LISTS**

To iterate over a list in Python, you can use different types of loops, such as for loops or while loops. Here's how you can do it:

**Using a for loop:**

A for loop is the most common way to iterate over each element in a list.

Example:

# Sample list

my\_list = [10, 20, 30, 40, 50]

# Iterating over the list using a for loop

for item in my\_list:

print(item)

**Output:**

10

20

30

40

50

**Using range() with a for loop:**

You can also use range() to iterate over the indices of the list.

Example:

# Sample list

my\_list = [10, 20, 30, 40, 50]

# Iterating over the list using range() to access indices

for i in range(len(my\_list)):

print(f"Index {i}: {my\_list[i]}")

**Output:**

Index 0: 10

Index 1: 20

Index 2: 30

Index 3: 40

Index 4: 50

**Using a while loop:**

You can use a while loop if you prefer working with indices or when you need more control over the iteration.

Example:

# Sample list

my\_list = [10, 20, 30, 40, 50]

# Iterating over the list using a while loop

i = 0

while i < len(my\_list):

print(my\_list[i])

i += 1

**Output:**

10

20

30

40

50

**Using List Comprehension:**

If you need to perform some operation on the list while iterating, you can use list comprehension. It's a concise way to generate new lists.

Example:

# Sample list

my\_list = [10, 20, 30, 40, 50]

# List comprehension to double each element

doubled = [item \* 2 for item in my\_list]

print(doubled)

**Output:**

[20, 40, 60, 80, 100]

These methods allow you to effectively iterate over lists depending on your requirements.

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

### 1. sort():

* The sort() method sorts the list **in place**, meaning it modifies the original list. By default, it sorts in ascending order.
* You can also use the reverse=True parameter to sort in descending order.

**Example of sort() (ascending and descending):**

# Sample list

my\_list = [50, 20, 30, 40, 10]

# Sorting in ascending order (default)

my\_list.sort()

print("Sorted in ascending order:", my\_list)

# Sorting in descending order

my\_list.sort(reverse=True)

print("Sorted in descending order:", my\_list)

**Output:**

Sorted in ascending order: [10, 20, 30, 40, 50]

Sorted in descending order: [50, 40, 30, 20, 10]

### 2. sorted():

* The sorted() function returns a **new sorted list** and does not modify the original list. Like sort(), it can also sort in ascending or descending order using the reverse parameter.

**Example of sorted() (ascending and descending):**

# Sample list

my\_list = [50, 20, 30, 40, 10]

# Sorting in ascending order

sorted\_list = sorted(my\_list)

print("Sorted in ascending order:", sorted\_list)

# Sorting in descending order

sorted\_list\_desc = sorted(my\_list, reverse=True)

print("Sorted in descending order:", sorted\_list\_desc)

# Original list remains unchanged

print("Original list:", my\_list)

**Output:**

Sorted in ascending order: [10, 20, 30, 40, 50]

Sorted in descending order: [50, 40, 30, 20, 10]

Original list: [50, 20, 30, 40, 10]

### 3. reverse():

* The reverse() method reverses the order of the elements in the list **in place**, meaning the original list is modified.

**Example of reverse():**

# Sample list

my\_list = [10, 20, 30, 40, 50]

# Reversing the list

my\_list.reverse()

print("Reversed list:", my\_list)

**Output:**

Reversed list: [50, 40, 30, 20, 10]

### 1. ****Addition****:

You can add items to a list using the following methods:

* **Using append()**: Adds an item to the end of the list.
* **Using insert()**: Adds an item at a specific index.
* **Using extend()**: Adds multiple items (another list) to the end of the current list.

**Example of Addition**:

# Initial list

my\_list = [10, 20, 30]

# Using append()

my\_list.append(40)

print("After append:", my\_list)

# Using insert() to add 25 at index 2

my\_list.insert(2, 25)

print("After insert:", my\_list)

# Using extend() to add multiple items

my\_list.extend([50, 60])

print("After extend:", my\_list)

**Output:**

After append: [10, 20, 30, 40]

After insert: [10, 20, 25, 30, 40]

After extend: [10, 20, 25, 30, 40, 50, 60]

### 2. ****Deletion****:

You can delete items from a list using the following methods:

* **Using remove()**: Removes the first occurrence of a specific item.
* **Using pop()**: Removes an item at a specific index (or the last item if no index is provided) and returns it.
* **Using del**: Deletes an item or a range of items (by index).
* **Using clear()**: Removes all items from the list.

**Example of Deletion**:

# Initial list

my\_list = [10, 20, 30, 40, 50]

# Using remove() to delete the first occurrence of 30

my\_list.remove(30)

print("After remove:", my\_list)

# Using pop() to remove and return the item at index 1

popped\_item = my\_list.pop(1)

print("Popped item:", popped\_item)

print("After pop:", my\_list)

# Using del to delete the item at index 2

del my\_list[2]

print("After del:", my\_list)

# Using clear() to remove all items

my\_list.clear()

print("After clear:", my\_list)

**Output:**

After remove: [10, 20, 40, 50]

Popped item: 20

After pop: [10, 40, 50]

After del: [10, 50]

After clear: []

### 3. ****Updating****:

You can update a list by assigning a new value to a specific index or by using slicing to update multiple values.

**Example of Updating**:

# Initial list

my\_list = [10, 20, 30, 40]

# Updating a specific item

my\_list[1] = 25 # Update index 1 (second item)

print("After updating index 1:", my\_list)

# Using slicing to update multiple items

my\_list[2:4] = [35, 45] # Replace the items from index 2 to index 3

print("After updating slice:", my\_list)

**Output:**

After updating index 1: [10, 25, 30, 40]

After updating slice: [10, 25, 35, 45]

### 4. ****Slicing****:

Slicing allows you to access a part of the list (sublist) without modifying the original list.

* **Syntax**: list[start:end:step]
  + start is the index where the slice starts (inclusive).
  + end is the index where the slice ends (exclusive).
  + step is the step size (optional).

**Example of Slicing**:

# Initial list

my\_list = [10, 20, 30, 40, 50]

# Slicing the list (from index 1 to index 3)

sub\_list = my\_list[1:4]

print("Sliced list (1 to 3):", sub\_list)

# Slicing with a step (every second item)

sub\_list\_step = my\_list[::2]

print("Sliced list with step:", sub\_list\_step)

# Slicing with negative indices (last two items)

sub\_list\_negative = my\_list[-2:]

print("Sliced list (last 2 items):", sub\_list\_negative)

**Output:**

Sliced list (1 to 3): [20, 30, 40]

Sliced list with step: [10, 30, 50]

Sliced list (last 2 items): [40, 50]