1. Creating & Accessing Lists

A list in Python is an ordered, mutable collection of elements. Lists can store elements of different data types.

Syntax:

```
list_name = [element1, element2, element3, ...]
```

Accessing Elements (Indexing)

Positive Indexing: Starts from 0 (left to right).

```
list_name[index]
```

Negative Indexing: Starts from -1 (right to left).

```
list_name[-index]
```

Looping Through a List

Using a `for` loop:

```
for item in list_name:
    # Code to execute
```

Using an index-based loop:

```
for i in range(len(list_name)):
    # Code to execute using list_name[i]
```

---

2. List Slicing

Used to extract portions of a list.

Syntax:

```
list[start_index:end_index:step_size]
```

- `start_index`: Starting position (default is 0).
- `end_index`: Stopping position (excluded).
- `step_size`: Determines increment (default is 1).

Common slicing patterns:

```
list[:]        # Copies entire list
list[start:]   # From start index to the end
list[:end]     # From beginning to end index (exclusive)
list[::step]   # Steps through list based on step_size
list[::-1]     # Reverses the list
```

---

3. Modifying Lists (Mutability)

Lists are **mutable**, meaning their elements can be changed.

Syntax for modifying elements:

```
list_name[index] = new_value
```

---

4. Adding Elements to a List

`append()`: Adds an element at the end of the list.

```
list_name.append(value)
```

`insert()`: Inserts an element at a specified index.

```
list_name.insert(index, value)
```

`extend()`: Adds multiple elements from another iterable.

```
list_name.extend(iterable)
```

---

5. Removing Elements from a List

`remove()`: Removes the first occurrence of a specific value.

```
list_name.remove(value)
```

`pop()`: Removes and returns an element at the given index (default is last element).

```
list_name.pop(index)
```

`clear()`: Removes all elements from the list.

```
list_name.clear()
```

`del` statement: Deletes an element or the entire list.

```
del list_name[index]  # Deletes a specific element
del list_name         # Deletes the entire list
```

---

6. Sorting Lists

`sort()`: Sorts the list in-place (modifies the original list).

```
list_name.sort()         # Ascending order
list_name.sort(reverse=True)  # Descending order
```

`sorted()`: Returns a new sorted list without modifying the original list.

```
new_list = sorted(list_name)
new_list = sorted(list_name, reverse=True)  # Descending order
```

---

7. Reversing a List

Using `reverse()`: Modifies the original list.

```
list_name.reverse()
```

Using slicing (`[::-1]`): Returns a reversed copy of the list.

```
reversed_list = list_name[::-1]
```

Using `reversed()`: Returns an iterator that can be converted to a list.

```
new_list = list(reversed(list_name))
```

---

8. Checking Membership in a List

`in` Operator: Checks if an element exists in the list.

```
if value in list_name:
    # Code to execute
```

`not in` Operator: Checks if an element does not exist in the list.

```
if value not in list_name:
    # Code to execute
```

---

9. Getting List Length

Using `len()`: Returns the number of elements in a list.

```
length = len(list_name)
```

---

10. Iterating Through a List with `enumerate()`

Returns both index and value while iterating.

```
for index, value in enumerate(list_name):
    # Code to execute
```

Additional Examples to Cover Missed Topics

Using `count()` to find occurrences of an element

```
list_name.count(value)
```

1.

Using `index()` to find the first occurrence of an element

```
list_name.index(value)
```

2.

Copying a list using slicing and `copy()`

```
new_list = list_name[:]
new_list = list_name.copy()
```

3.

Creating a list with `range()`

```
list_name = list(range(start, end, step))
```

4. **List comprehension to create a modified list**

   ```
   new_list = [expression for item in list_name]
   ```