Introduction to Lists

- A list is an ordered, mutable collection of items.
- Lists can hold mixed data types including numbers, strings, and other lists.
- Defined using square brackets `[]` or the `list()` constructor.
- Lists can be nested and dynamically resized.
- Example & Output:

Python

```python
my_list = [1, "apple", 3.14, [2, 3]]
print(my_list)
```

Output:

```
[1, 'apple', 3.14, [2, 3]]
```

Characteristics of Lists

- Lists maintain insertion order.
- They are mutable and allow changes after creation.
- Indexing starts at 0 and supports negative indices.
- Lists can contain duplicates and mixed types.
- Example & Output:

Python

```python
data = [10, 20, 30]
data[1] = 25
print(data)
```

Output:

```
[10, 25, 30]
```

🛠️ Creating a List

- Use `[]` or `list()` to create a list.
- Lists can be empty or pre-filled.
- You can use repetition to initialize with default values.
- Lists can be created from other iterables.
- Example & Output:

Python

```python
fruits = ["apple", "banana", "cherry"]
print(fruits)
```

Output:

```
['apple', 'banana', 'cherry']
```

🏗️ `list()` Constructor

- The `list()` constructor creates a list from any iterable like strings, tuples, or sets.
- It's useful when converting other data types into list format.
- You can use it to split strings into characters or unpack tuples.
- It returns a new list object containing the elements of the iterable.
- Example & Output:

Python

```python
my_tuple = (1, 2, 3)
converted_list = list(my_tuple)
print(converted_list)
```

Output:

```
[1, 2, 3]
```

🔄 Updating and Accessing Elements

- Use indexing to access or update elements.
- Slicing returns a sublist.
- Nested lists are accessed with multiple indices.
- Negative indices count from the end.
- Example & Output:

Python

```python
colors = ["red", "blue", "green"]
colors[1] = "yellow"
print(colors[1:3])
```

Output:

```
['yellow', 'green']
```

Traversing a List

- Use `for` loop to iterate over items.
- `enumerate()` gives index and value.
- List comprehensions offer concise traversal.
- Useful for filtering and transformations.
- Example & Output:

Python

```python
for item in ["a", "b", "c"]:
    print(item)
```

Output:

Code

```
a
b
c
```

🗑️ Deleting Elements from a List

- `del` removes by index.
- `remove()` deletes first matching value.
- `pop()` removes and returns item by index.
- All modify the list in-place.
- Example & Output:

Python

```python
nums = [1, 2, 3, 4]
nums.remove(3)
print(nums)
```

Output:

```
[1, 2, 4]
```

1. `del` Statement

- Deletes an element at a specific index.
- It does **not** return the deleted item.
- Useful for removing items without needing the value.
- Example & Output:

Python

```python
nums = [10, 20, 30, 40]
del nums[2]
print(nums)
```

Output:

```
[10, 20, 40]
```

2. `pop()` Method

- Removes and **returns** the item at the given index.
- If no index is provided, it removes the last item.
- Useful when you need the removed value.
- Example & Output:

Python

```python
nums = [5, 15, 25, 35]
removed = nums.pop(1)
print("Removed:", removed)
print("Updated List:", nums)
```

Output:

Code

```
Removed: 15
Updated List: [5, 25, 35]
```

List Operations

- `+` concatenates lists.
- `*` repeats list elements.
- `in` checks for membership.
- `len()` returns number of items.
- Example & Output:

Python

```python
a = [1, 2]
b = [3, 4]
print(a + b)
```

Output:

```
[1, 2, 3, 4]
```

🔁 Repeat (`*`) Operator

- The `*` operator repeats the elements of a list a specified number of times.
- It's useful for initializing lists with default values.
- The result is a new list with repeated elements.
- Works with any integer value.
- Example & Output:

Python

```python
repeated = ["hi"] * 3
print(repeated)
```

Output:

```
['hi', 'hi', 'hi']
```

🔍 Membership (`in`) Operator

- The `in` operator checks if an item exists in a list.
- Returns `True` if found, `False` otherwise.
- Useful for conditionals and filtering.
- Can be used with `if` statements or expressions.
- Example & Output:

Python

```python
fruits = ["apple", "banana", "cherry"]
print("banana" in fruits)
```

Output:

```
True
```

Built-in List Functions

Function	Description	Example	Output
`len()`	Number of items	`len([1, 2, 3])`	`3`
`max()`	Largest item	`max([1, 5, 3])`	`5`
`min()`	Smallest item	`min([1, 5, 3])`	`1`
`sum()`	Sum of items	`sum([1, 2, 3])`	`6`
`clear()`	Removes all items	`a = [1,2]; a.clear(); print(a)`	`[]`

List Methods (15 Total)

Method	Description	Example	Output
`append()`	Adds item to end	`a = [1]; a.append(2); print(a)`	`[1, 2]`
`extend()`	Adds multiple items	`a = [1]; a.extend([2,3]); print(a)`	`[1, 2, 3]`
`insert()`	Inserts at index	`a = [1,3]; a.insert(1,2); print(a)`	`[1, 2, 3]`
`remove()`	Removes first match	`a = [1,2,2]; a.remove(2); print(a)`	`[1, 2]`
`pop()`	Removes by index	`a = [1,2]; a.pop(0); print(a)`	`[2]`
`index()`	Finds index of value	`a = [1,2]; print(a.index(2))`	`1`
`count()`	Counts occurrences	`a = [1,2,2]; print(a.count(2))`	`2`
`sort()`	Sorts list	`a = [3,1,2]; a.sort(); print(a)`	`[1, 2, 3]`
`reverse()`	Reverses list	`a = [1,2]; a.reverse(); print(a)`	`[2, 1]`