Lists in Python

Data Structures - Lists in Python

A list in Python is a built-in data structure that can store a collection of items. These items can be of any type — numbers, strings, other lists, objects, etc. Lists are ordered, changeable (mutable), and allow duplicate elements.

You can think of a list as a container that holds multiple items inside square brackets [ ].

Key features of a list:

- **Ordered:** Items have a defined order, and that order will not change unless you explicitly reorder the list.
- **Mutable:** You can change, add, or remove items after the list is created.
- **Allows duplicates:** The same value can appear multiple times in a list.

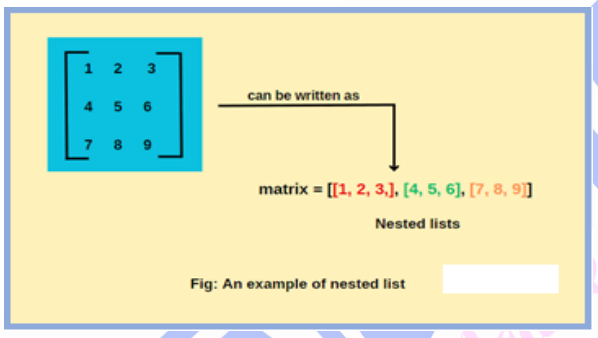

Fig: An example of nested list

Why we used list in Python?

1. **Store multiple values in one variable:**
Instead of creating separate variables for each item (like item1, item2, item3), a list lets you group all related items together.
2. **Ordered data:**
Lists keep the order of items, so you can access elements by their position (index). This is useful when order matters, like in sequences or when processing data step-by-step.
3. **Dynamic size:**
Lists can grow or shrink — you can add or remove items anytime. This flexibility is great for programs where the amount of data isn't fixed.
4. **Support various data types:**
A list can contain different types of data at once — numbers, strings, even other lists or objects.
5. **Easy to loop through:**
You can easily iterate over a list to perform operations on each item, like processing a list of names or calculating values.

Lists in Python

6. **Useful built-in methods:**
Python provides many handy methods (like .append(), .remove(), .sort(), .reverse()) that make working with lists simple.

Syntax:
my_list = [item1, item2, item3, ...]

Solved Example 1: Creating a List

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

Output:
['apple', 'banana', 'cherry']

Explanation:
This creates a list of fruits with three elements. The list is printed as it is.
Output: ['apple', 'banana', 'cherry']

Solved Example 2: Lists with Different Data Types

```
mix_list = [25, "Abhyas", 3.14, True]
print(mix_list)
```

Output:
[25, 'Abhyas', 3.14, True]

Explanation:
A list can store numbers, text, decimal, and Boolean values together.

Solved Example 3: Accessing List Elements (Indexing)

Each item in a list has a position called an **index**, starting from 0.

```
colors = ["red", "green", "blue"]
print(colors[0])  # First element
print(colors[2])  # Third element
```

Output:
red
blue
You can also use **negative indexing**:

```
print(colors[-1])  # last element
```

Output:
blue

Explanation:
- colors[0] gives first item → 'red'
- colors[2] gives third item → 'blue'
- colors[-1] gives last item → 'blue'

Solved Example 4: Changing List Items

Lists are **mutable**, meaning you can change their contents.

```
numbers = [10, 20, 30, 40]
numbers[2] = 99
print(numbers)
```

Output:
[10, 20, 99, 40]

Explanation:
List elements can be changed. Here, 3rd element 30 is replaced by 99.

Solved Example 5: Adding Elements

You can add new items using:
- append() – adds an item at the end
- insert() – adds at a specific position
- extend() – adds multiple items

```
fruits = ["apple", "banana"]
fruits.append("cherry")
fruits.insert(1, "orange")
fruits.extend(["grape", "mango"])
print(fruits)
```

Explanation:
- append() adds at end → "cherry"
- insert(1, "orange") adds "orange" at index 1
- extend() joins another list

Output:
['apple', 'orange', 'banana', 'cherry', 'grape', 'mango']

Solved Example 6: Removing Elements
- remove(item) → removes by value

- pop(index) → removes by position
- del list[index] → deletes by index
- clear() → empties the list

```
nums = [1, 2, 3, 4, 5]
nums.remove(3)
nums.pop(2)
del nums[0]
nums.clear()
print(nums)
```

Explanation:
- remove(3) deletes value 3
- pop(2) removes item at index 2
- del nums[0] deletes first element
- clear() empties the list

Output:
```
[]
```

Solved Example 7: Looping Through a List

```
animals = ["dog", "cat", "rabbit"]
for a in animals:
    print(a)
```

Explanation:
A for loop prints each item one by one.

Output:
```
dog
cat
rabbit
```

Solved Example 8: Checking if an Item Exists

```
cities = ["Delhi", "Mumbai", "Chennai"]
if "Delhi" in cities:
    print("Yes, Delhi is in the list.")
```

Explanation:
The in keyword checks if "Delhi" exists in the list.

Output:

Yes, Delhi is in the list.

Solved Example 9: Length of List

```
marks = [89, 76, 90, 85]
print(len(marks))
```

Explanation:
len() gives the total number of items in the list.

Output:
4

Solved Example 10: List Slicing

You can access parts of a list using **slicing**.

```
nums = [10, 20, 30, 40, 50]
print(nums[1:4])
```

Explanation:
nums[1:4] returns items from index 1 to 3 (not including 4).

Output:
[20, 30, 40]

Solved Example 11: Combining Two Lists

```
list1 = [1, 2, 3]
list2 = [4, 5]
combined = list1 + list2
print(combined)
```

Explanation:
Using + joins both lists into one.
Output: [1, 2, 3, 4, 5]

Solved Example 12: Common List Functions

Function	Description	Example	Output
len()	Counts elements	len([1,2,3])	3
max()	Finds largest	max([1,9,2])	9
min()	Finds smallest	min([1,9,2])	1

Lists in Python

Function	Description	Example	Output
sum()	Adds all values	sum([1,2,3])	6
list()	Converts to list	list("abc")	['a','b','c']

Explanation:
These built-in functions make working with lists faster and easier — you can quickly calculate totals, find max/min, or convert other data into lists.

Assignment

Ques 1: Create a list named fruits containing 5 fruit names.
Print the full list and then print the first and last fruit using indexing.

Ques 2: Accessing List Elements
Create a list colors = ["red", "green", "blue", "yellow", "pink"].
Print:
1. The second and fourth elements.
2. The last element using negative indexing.

Ques 3: Adding and Removing Items
Start with a list numbers = [10, 20, 30].
Perform the following:
1. Add 40 using append().
2. Insert 15 at index 1 using insert().
3. Remove 30 from the list using remove().
4. Print the final list.