TOPSTechnologies

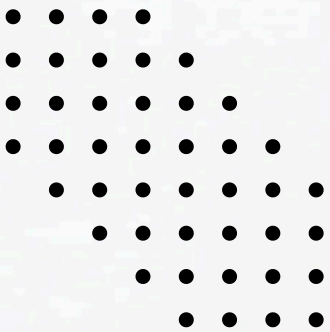
Working with Lists

Presented for :

TOPs Technologies

Presented by :

Sumit B Yadav



Python - Collections , Module, Functions

Que 1

Iterating over a list using loops is a fundamental concept in programming.

1. Using a for loop
The simplest and most common way to iterate over a list:

syntax-

```
my_list = [1, 2, 3, 4, 5]

for item in my_list:
    print(item)
```

2. Using for with enumerate
If you need both the index and the value, use enumerate:

syntax-

```
for index, value in enumerate(my_list):
    print(f"Index: {index}, Value: {value}")
```

3. Using a while loop

You can iterate using a while loop by maintaining an index variable:

syntax-

```
index = 0
while index < len(my_list):
print(my_list[index])
index += 1
```

4. List comprehension (for simple transformations)

Though not strictly a "loop," list comprehensions are a concise way to iterate and create new lists:

syntax-

```
squared = [x ** 2 for x in my_list]
print(squared)
```

5. Iterating with conditions

You can add conditions to filter elements during iteration:

syntax-

```
for item in my_list:
if item % 2 == 0:
print(item)
```

6. Using a for loop with a function
You can pass each element to a function within the loop:

synatx-

```
def greet(name):
print(f"Hello, {name}!")

names = ["Alice", "Bob", "Charlie"]

for name in names:
    greet(name)
```

7. Using break and continue
Control the loop's flow with break (to exit the loop) and continue (to skip the current iteration):

python
CopyEdit

synatx-

```
for item in my_list:
    if item == 3:
        break
print(item)

for item in my_list:
    if item % 2 != 0:
        continue
print(item)
```

Que. 2

1. Using sort()

The sort() method modifies the list in place and sorts it in ascending order by default.

synatax-

```
my_list = [5, 2, 8, 1, 9]

# Sorting in ascending order
my_list.sort()
print(my_list)
```

2. Using sorted()

The sorted() function returns a new sorted list, leaving the original list unchanged.

syntax-

```
my_list = [5, 2, 8, 1, 9]

# Sorting in ascending order
sorted_list = sorted(my_list)
print(sorted_list)  # New sorted list
print(my_list)      # Original list remains unchanged
```

3. Reversing a list using reverse()

The reverse() method modifies the list in place by reversing the order of its elements.

syntax-

```
my_list = [5, 2, 8, 1, 9]

# Reversing the list
my_list.reverse()
print(my_list)
```

4. Reversing using slicing

A more concise way to reverse a list (without modifying the original) is using slicing:

syntax-

```
my_list = [5, 2, 8, 1, 9]

reversed_list = my_list[::-1]
print(reversed_list)
```

5. Using key for custom sorting

Both sort() and sorted() allow you to provide a key function for custom sorting logic.

3. Reversing a list using reverse()

The reverse() method modifies the list in place by reversing the order of its elements.

syntax-

```
my_list = [5, 2, 8, 1, 9]

# Reversing the list
my_list.reverse()
print(my_list)
```

4. Reversing using slicing

A more concise way to reverse a list (without modifying the original) is using slicing:

syntax-

```
my_list = [5, 2, 8, 1, 9]

reversed_list = my_list[::-1]
print(reversed_list)
```

5. Using key for custom sorting

Both sort() and sorted() allow you to provide a key function for custom sorting logic.

Que. 3

1. Adding Elements to a List

Using append()

Adds a single element to the end of the list.

syntax-

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)
```

2. Deleting Elements from a List

Using remove()

Removes the first occurrence of a specified value.

syntax-

```
my_list = [1, 2, 3, 2]
my_list.remove(2)  # Removes the first 2
print(my_list)
```

3. Updating Elements in a List

You can update elements by accessing them directly using their index.

syntax-

```
my_list = [1, 2, 3]
my_list[1] = 10  # Change the second element to 10
print(my_list)
```

4. Slicing a List

Slicing allows you to access parts of a list.

syntax-

```
my_list = [1, 2, 3, 4, 5]
print(my_list[1:4])  # Elements from index 1 to 3
```