

Python List

- The list is a sequence data type which is used to store the collection of data.
- In simple language, a list is a collection of things, enclosed in [] and separated by commas.
- Lists are the simplest containers that are an integral part of the Python language. Lists need not be homogeneous always which makes it the most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

Creating a List in Python

Lists in Python can be created by just placing the sequence inside the square brackets []. Unlike Sets, a list doesn't need a built-in function for its creation of a list.

```
# a list of three elements
ages = [19, 26, 29]
print(ages)

# Output: [19, 26, 29]
```

List Items of Different Types

We can store data of different data types in a Python list. For example,

```
# a list containing strings and numbers
student = ['Jack', 32, 'Computer Science']
print(student)

# an empty list
empty_list = []
print(empty_list)
```

Output

```
['Jack', 32, 'Computer Science']  
[]
```

Accessing a Single Element by Index

```
# List of numbers  
numbers = [10, 20, 30, 40, 50]  
  
# Access the first element  
first_element = numbers[0] # Output: 10  
print(first_element)  
  
# Access the third element  
third_element = numbers[2] # Output: 30  
print(third_element)
```

Accessing Elements Using Negative Indexing

```
# Access the last element  
last_element = numbers[-1] # Output: 50  
print(last_element)  
  
# Access the second-to-last element  
second_last_element = numbers[-2] # Output: 40  
print(second_last_element)
```

Iterating Over a List Using a for Loop

- The for loop iterates over each element in the list directly.
- It's the most common and concise way to iterate through a list.

Example:

```
# List of fruits
fruits = ["apple", "banana", "cherry", "date"]

# Iterate using a for loop
for fruit in fruits:
    print(fruit)
```

Output:

```
apple
banana
cherry
date
```

Explanation:

- The for loop goes through each element in the fruits list.
- Each element is assigned to the variable fruit in each iteration, and it gets printed.

Iterating Over a List Using a while Loop

- The while loop uses a counter/index to access elements.
- This approach gives you more control over the loop, but it's more verbose.

Example:

```
# List of fruits
fruits = ["apple", "banana", "cherry", "date"]

# Initialize the index
index = 0

# Iterate using a while loop
while index < len(fruits):
    print(fruits[index])
    index += 1 # Increment the index to move to the next element
```

Output

```
apple
banana
cherry
date
```

Explanation:

- The while loop continues as long as index is less than the length of the fruits list.
- Inside the loop, the element at the current index is printed.
- After printing, the index is incremented by 1 to move to the next element.

Both methods effectively iterate over the elements of the list, but the for loop is typically preferred for its simplicity and readability. Use the while loop when you need more control over the iteration process.

List Methods in Python

Let's look at some different list methods in Python for Python lists:


1	<u>append()</u>	Used for adding elements to the end of the List.
2	<u>copy()</u>	It returns a shallow copy of a list
3	<u>clear()</u>	This method is used for removing all items from the list.
4	<u>count()</u>	These methods count the elements.
5	<u>extend()</u>	Adds each element of an iterable to the end of the List

6	<code>index()</code>	Returns the lowest index where the element appears.
7	<code>insert()</code>	Inserts a given element at a given index in a list.
8	<code>pop()</code>	Removes and returns the last value from the List or the given index value.
9	<code>remove()</code>	Removes a given object from the List.
10	<code>reverse()</code>	Reverses objects of the List in place.
11	<code>sort()</code>	Sort a List in ascending, descending, or user-defined order
12	<code>min()</code>	Calculates the minimum of all the elements of the List
13	<code>max()</code>	Calculates the maximum of all the elements of the List

1. `append()`

Adds an element to the end of the list.

python


 Copy code

```
numbers = [1, 2, 3]
numbers.append(4)
print(numbers) # Output: [1, 2, 3, 4]
```

2. `copy()`

Creates a shallow copy of the list.

python


 Copy code

```
numbers = [1, 2, 3]
numbers_copy = numbers.copy()
print(numbers_copy) # Output: [1, 2, 3]
```

3. `clear()`

Removes all elements from the list.

python


 Copy code

```
numbers = [1, 2, 3]
numbers.clear()
print(numbers) # Output: []
```

4. `count()`

Returns the number of occurrences of a specified element in the list.

python


 Copy code

```
numbers = [1, 2, 2, 3]
count_of_two = numbers.count(2)
print(count_of_two) # Output: 2
```

5. `extend()`

Adds elements of an iterable (like another list) to the end of the list.

python


 Copy code

```
numbers = [1, 2, 3]
numbers.extend([4, 5])
print(numbers) # Output: [1, 2, 3, 4, 5]
```

6. `index()`

Returns the index of the first occurrence of a specified element.

python


 Copy code

```
numbers = [1, 2, 3, 2]
index_of_two = numbers.index(2)
print(index_of_two) # Output: 1
```

7. `insert()`

Inserts an element at a specific position in the list.

python


 Copy code

```
numbers = [1, 2, 4]
numbers.insert(2, 3)
print(numbers) # Output: [1, 2, 3, 4]
```

8. `pop()`

Removes and returns the element at the specified index (or the last element if no index is specified).

python


 Copy code

```
numbers = [1, 2, 3]
last_element = numbers.pop()
print(last_element) # Output: 3
print(numbers) # Output: [1, 2]
```


9. `remove()`

Removes the first occurrence of a specified element.

python


 Copy code

```
numbers = [1, 2, 3, 2]
numbers.remove(2)
print(numbers) # Output: [1, 3, 2]
```

10. `reverse()`

Reverses the elements of the list.

python


 Copy code

```
numbers = [1, 2, 3]
numbers.reverse()
print(numbers) # Output: [3, 2, 1]
```

11. `sort()`

Sorts the list in ascending order by default (can also be sorted in descending order).

python

 Copy code


```
numbers = [3, 1, 4, 2]
numbers.sort()
print(numbers) # Output: [1, 2, 3, 4]

numbers.sort(reverse=True)
print(numbers) # Output: [4, 3, 2, 1]
```

12. `min()`

Returns the smallest element in the list.

python


 Copy code

```
numbers = [3, 1, 4, 2]
minimum = min(numbers)
print(minimum) # Output: 1
```

13. `max()`

Returns the largest element in the list.

python

 Copy code

```
numbers = [3, 1, 4, 2]
maximum = max(numbers)
print(maximum) # Output: 4
```

List Comprehension

List comprehension provides a concise way to create lists in Python. It's a shorthand for looping through an iterable and applying an expression to each item.

Example: Create a list of squares

python

```
# Using list comprehension to create a list of squares
squares = [x**2 for x in range(5)]
print(squares) # Output: [0, 1, 4, 9, 16]
```

Example: Create a list of even numbers

python

```
# Using list comprehension to filter even numbers
evens = [x for x in range(10) if x % 2 == 0]
print(evens) # Output: [0, 2, 4, 6, 8]
```

Nested Lists

A nested list is a list that contains other lists as its elements. You can access elements of a nested list using indexing.

Example: Simple nested list

```
# A nested list
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Access the element in the second row, third column
print(matrix[1][2]) # Output: 6
```

Example: Iterate through a nested list

```
python

# Iterate through each row in the nested list
for row in matrix:
    for element in row:
        print(element, end=" ")
    print()
```

Output:

```
1 2 3
4 5 6
7 8 9
```