4-Lists القوائم

In Python, **lists** are mutable (modifiable)), ordered collections of items. Lists can hold items of any data type (integers, strings, floats, or even other lists) and can change in size as needed. Lists are one of the most versatile and frequently used data structures in Python.

Below is a comprehensive guide to Python lists with explanations, examples, and outputs.

1. Creating a List انشاء القائمة

Lists are created by placing all the elements (items) inside square brackets [], separated by commas.

Example:

Python Code

```
# Creating different types of lists
numbers = [1, 2, 3, 4, 5]              # List of integers
fruits = ["apple", "banana", "cherry"]   # List of strings
mixed = [1, "apple", 3.14, True]         # Mixed data types

print(numbers)
print(fruits)
print(mixed)
```

Output:

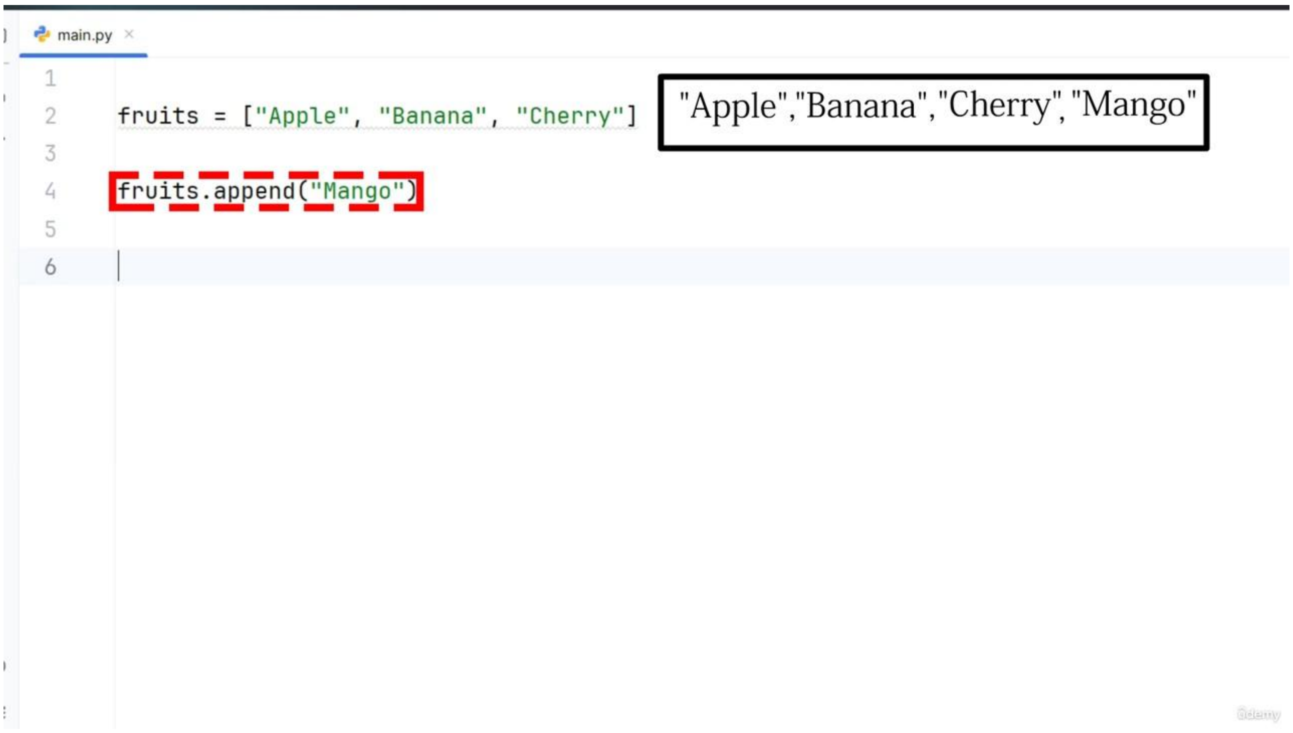
```
[1, 2, 3, 4, 5]
['apple', 'banana', 'cherry']
[1, 'apple', 3.14, True]
```

Explanation:

- numbers: A list of integers.
- fruits: A list of strings.
- mixed: A list containing multiple data types (integers, strings, floats, booleans).

2. Accessing List Elements

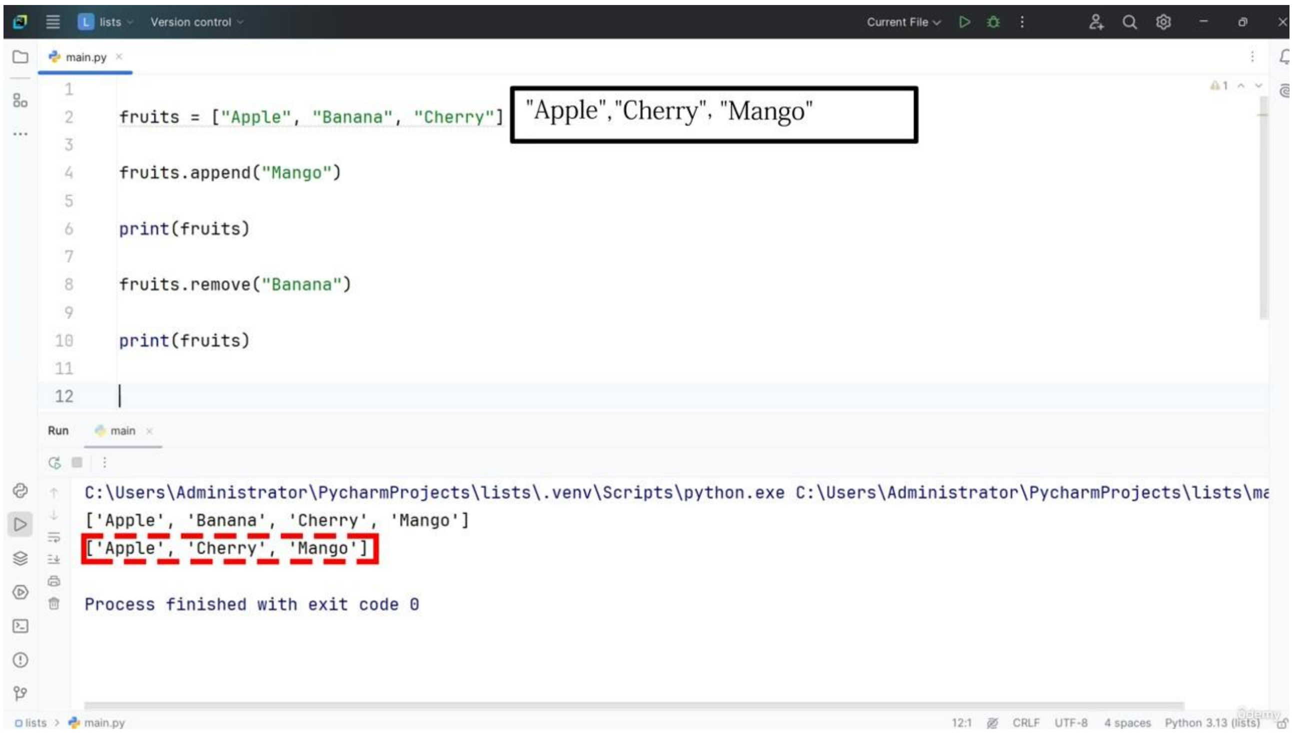
You can access elements of a list by referring to their **index**. Python uses **zero-based indexing**, meaning the first element has index 0.



main.py

```python
fruits = ["Apple", "Banana", "Cherry"]

fruits.append("Mango")

```

"Apple","Banana","Cherry","Mango"



lists Version control Current File

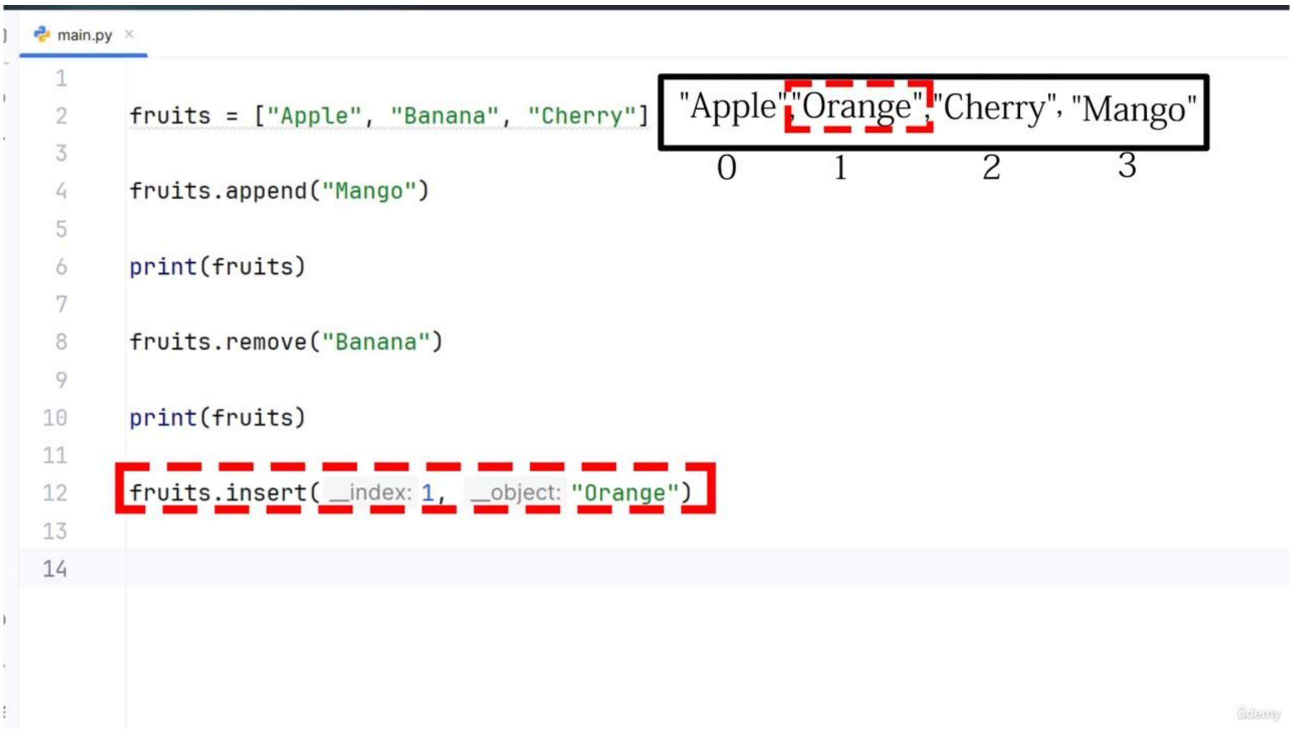
main.py

```python
fruits = ["Apple", "Banana", "Cherry"]

fruits.append("Mango")

print(fruits)

fruits.remove("Banana")

print(fruits)

```

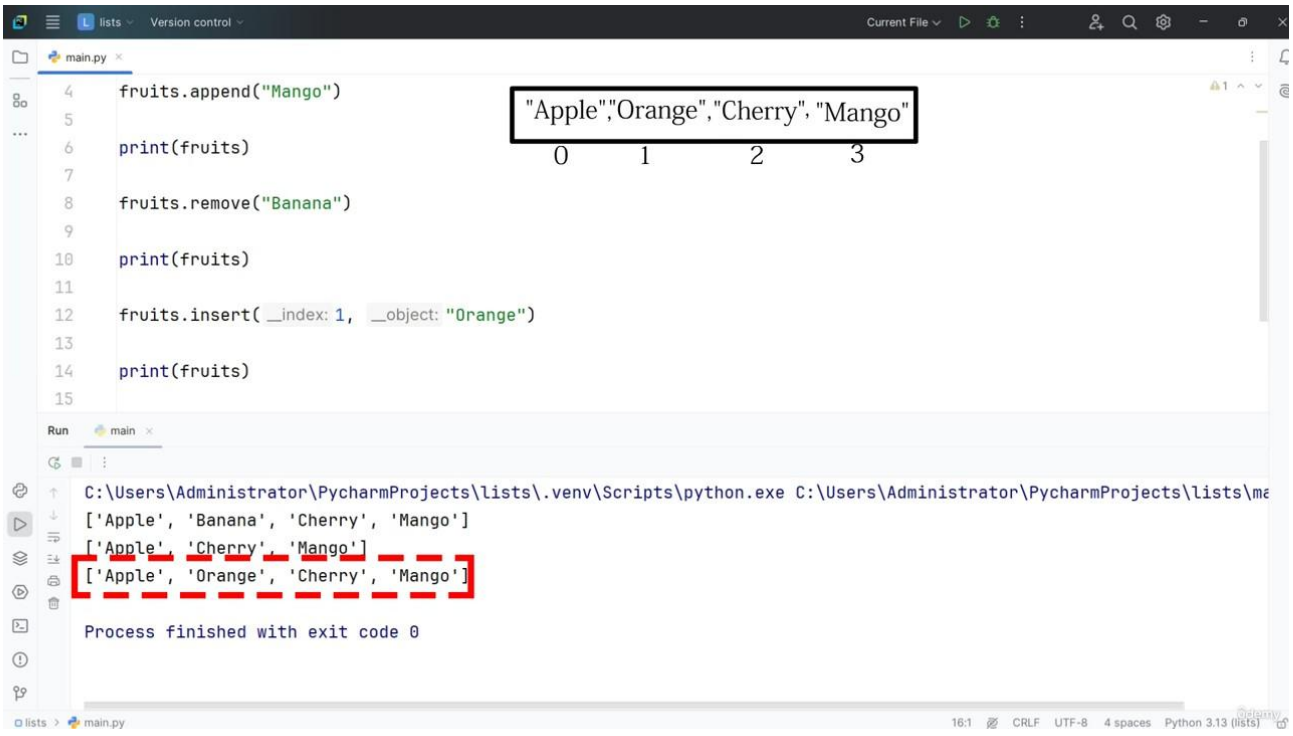
"Apple","Cherry", "Mango"

Run main

```
C:\Users\Administrator\PycharmProjects\lists\.venv\Scripts\python.exe C:\Users\Administrator\PycharmProjects\lists\ma
['Apple', 'Banana', 'Cherry', 'Mango']
['Apple', 'Cherry', 'Mango']

Process finished with exit code 0
```



```
fruits = ["Apple", "Banana", "Cherry"]

fruits.append("Mango")

print(fruits)

fruits.remove("Banana")

print(fruits)

fruits.insert( __index: 1,  __object: "Orange")
```



```
fruits.append("Mango")

print(fruits)

fruits.remove("Banana")

print(fruits)

fruits.insert( __index: 1,  __object: "Orange")

print(fruits)
```

```
C:\Users\Administrator\PycharmProjects\lists\.venv\Scripts\python.exe C:\Users\Administrator\PycharmProjects\lists\ma
['Apple', 'Banana', 'Cherry', 'Mango']
['Apple', 'Cherry', 'Mango']
['Apple', 'Orange', 'Cherry', 'Mango']

Process finished with exit code 0
```

"Apple","Orange","Cherry", "Mango"
0 1 2 3
lists
Version control
Current File
main.py
Run
main
16:1 CRLF UTF-8 4 spaces Python 3.13 (lists)

Example:

Python Code
fruits = ["apple", "banana", "cherry"]

print(fruits[0]) # First item
print(fruits[1]) # Second item

```
print(fruits[-1]) # Last item (negative indexing)
```

Output:

```
apple
banana
cherry
```

Explanation:

- The index 0 accesses the first item "apple".
- The index -1 accesses the last item "cherry".

3. Modifying List Elements

Lists are **mutable**, so you can change or update items after the list is created.

Example:

Python Code

```
fruits = ["apple", "banana", "cherry"]
fruits[1] = "blueberry"  # Change the second item

print(fruits)
```

Output:

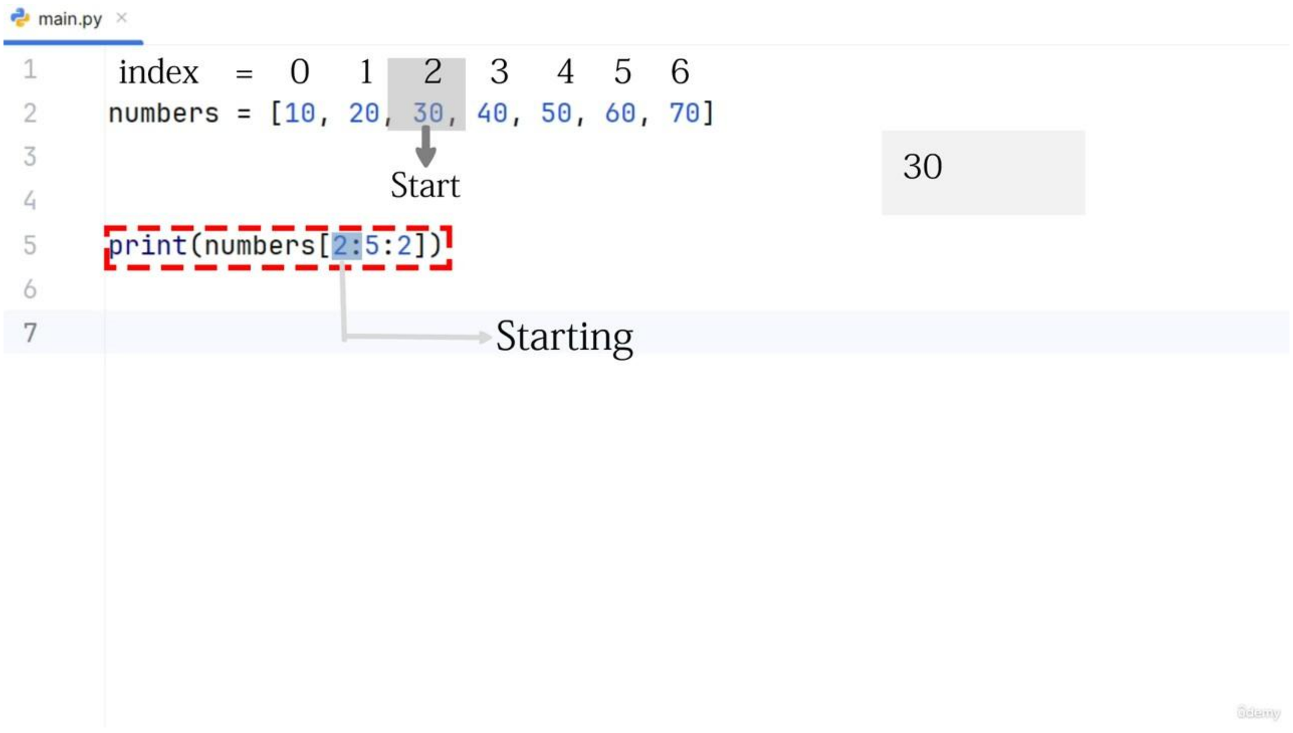
```
 ['apple', 'blueberry', 'cherry']
```

Explanation:

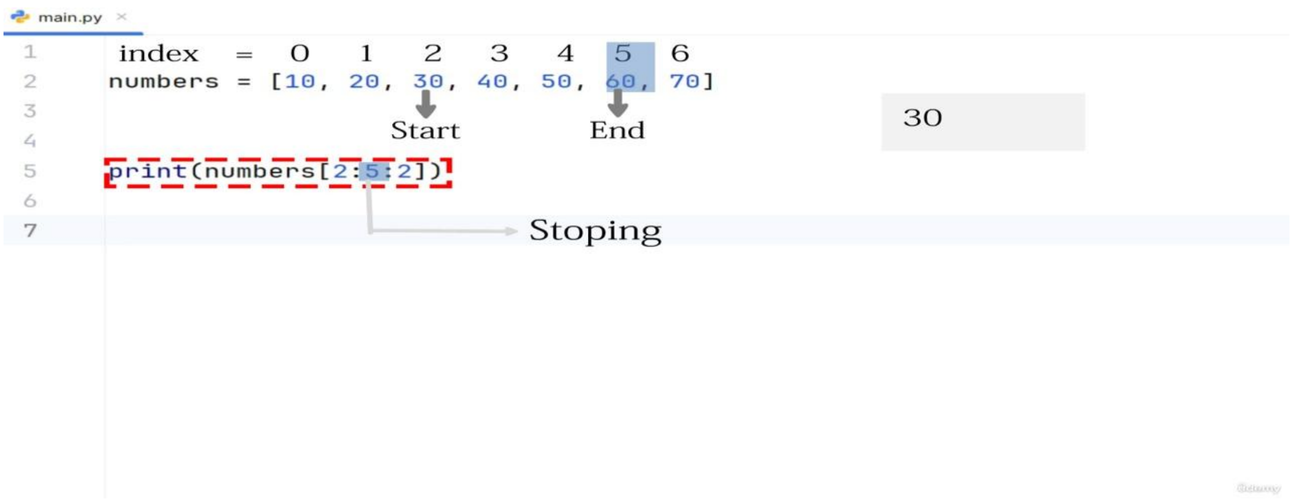
- The second item "banana" is replaced with "blueberry" using index 1.

4. Slicing Lists

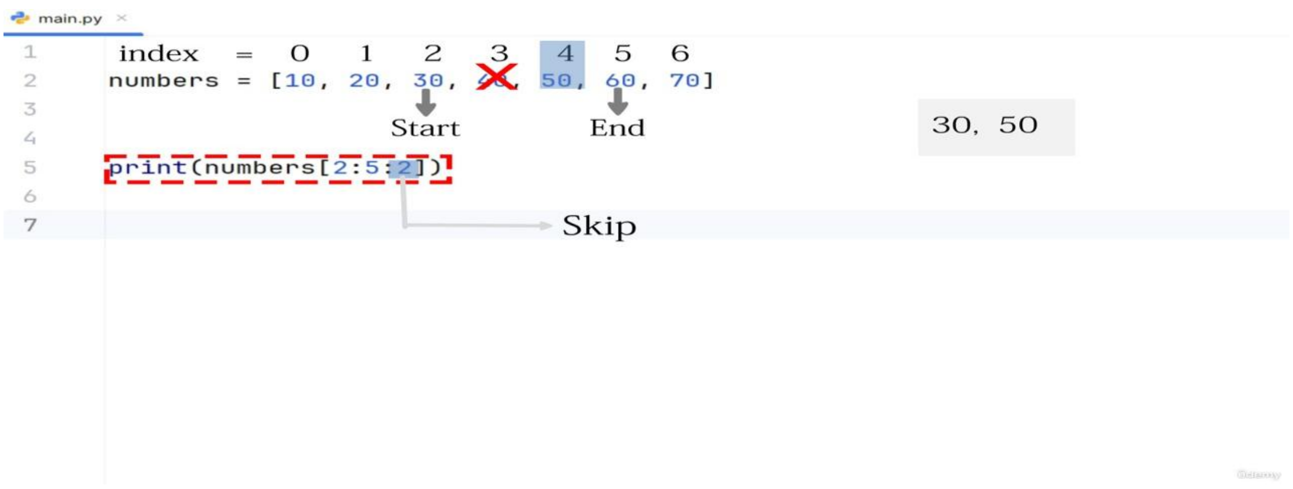
You can slice a list to retrieve a range of elements using the syntax list[start:end], where start is inclusive and end is exclusive.



```
 index   =  0    1    2    3    4    5    6
numbers = [10, 20, 30, 40, 50, 60, 70]


print(numbers[2:5:2])
```



```
 index   =  0    1    2    3    4    5    6
numbers = [10, 20, 30, 40, 50, 60, 70]


print(numbers[2:5:2])
```



```
 index   =  0    1    2    3    4    5    6
numbers = [10, 20, 30, 40, 50, 60, 70]


print(numbers[2:5:2])
```

Example:

Python Code

```
numbers = [1, 2, 3, 4, 5, 6, 7]

print(numbers[2:5])  # Elements from index 2 to 4
print(numbers[:3])   # Elements from the start to index 2
print(numbers[3:])   # Elements from index 3 to the end
print(numbers[::2])  # Every second element
```

Output:

```
 [3, 4, 5]
[1, 2, 3]
[4, 5, 6, 7]
[1, 3, 5, 7]
```

Explanation:

- numbers[2:5]: Slices from index 2 to 4 (index 5 is excluded).
- numbers[::2]: Returns every second element (step of 2).

5. Adding Elements to a List

You can add new elements to a list using append(), insert(), or extend() methods.

Example 1: append()

Code

```
fruits = ["apple", "banana"]
fruits.append("cherry")  # Add 'cherry' to the end of the list

print(fruits)
```

Output:

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

Explanation:

- append() adds a single element to the end of the list.

Example 2: insert()

Code

```
fruits = ["apple", "banana"]
fruits.insert(1, "blueberry")  # Insert 'blueberry' at index 1

print(fruits)
```

Output:

```
 ['apple', 'blueberry', 'banana']
```

Explanation:

- insert() allows you to add an element at a specific index.

Example 3: extend()

Python Code

```
fruits = ["apple", "banana"]
tropical_fruits = ["pineapple", "mango"]

fruits.extend(tropical_fruits)  # Extend 'fruits' with another list

print(fruits)
```

Output:

```
['apple', 'banana', 'pineapple', 'mango']
```

Explanation:

- extend() adds all elements from tropical_fruits to the fruits list.

6. Removing Elements from a List

You can remove elements using remove(), pop(), or del.

Example 1: remove()

Code

```
fruits = ["apple", "banana", "cherry"]
fruits.remove("banana")  # Remove 'banana'

print(fruits)
```

Output:

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

Explanation:

- remove() removes the first occurrence of the specified value ("banana").

Example 2: pop()

Code

```
fruits = ["apple", "banana", "cherry"]
removed_fruit = fruits.pop(1)  # Remove item at index 1

print(fruits)
print("Removed:", removed_fruit)
```

Output:

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

Explanation:

- pop() removes the element at the specified index and returns it.

Example 3: del

Python Code

```
fruits = ["apple", "banana", "cherry"]
del fruits[0]  # Delete item at index 0

print(fruits)
```

Output:

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

Explanation:

- The del statement removes an element at the specified index.

7. List Comprehension

List comprehension provides a concise way to create lists by iterating over an iterable and optionally applying a condition.

Example:

Python Code

```
# Creating a list of squares
squares = [x ** 2 for x in range(1, 6)]
print(squares)

# Filtering even numbers
even_numbers = [x for x in range(10) if x % 2 == 0]
print(even_numbers)
```

Output:

```
 [1, 4, 9, 16, 25]
[0, 2, 4, 6, 8]
```

Explanation:

- The list comprehension [x ** 2 for x in range(1, 6)] creates a list of squares from 1 to 5.
- The list comprehension [x for x in range(10) if x % 2 == 0] filters out even numbers from the range.

8. Sorting and Reversing Lists

You can sort and reverse lists using the sort(), sorted(), and reverse() methods.

Example 1: sort()

Python Code

```
numbers = [4, 2, 9, 1, 5]
numbers.sort()  # Sort the list in ascending order

print(numbers)
```

Output:

```
 [1, 2, 4, 5, 9]
```

Explanation:

- sort() sorts the list in place in ascending order.

Example 2: sorted()

Python Code

```
numbers = [4, 2, 9, 1, 5]
```

```
sorted_numbers = sorted(numbers)  # Return a sorted copy of the list

print(sorted_numbers)
print(numbers)  # The original list remains unchanged
```

Output:

```
[1, 2, 4, 5, 9]
[4, 2, 9, 1, 5]
```

Explanation:

- sorted() returns a new sorted list without modifying the original list.

Example 3: reverse()

Python Code

```
numbers = [4, 2, 9, 1, 5]
numbers.reverse()  # Reverse the list

print(numbers)
```

Output:

```
[5, 1, 9, 2, 4]
```

Explanation:

- reverse() reverses the elements of the list in place.

9. Copying Lists

You can copy a list using the copy() method or list slicing.

Example 1: copy()

Python Code

```
fruits = ["apple", "banana", "cherry"]
new_fruits = fruits.copy()  # Create a copy of the list

print(new_fruits)
```

Output:

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

Example 2: List Slicing

Python Code

```
fruits = ["apple", "banana", "cherry"]
new_fruits = fruits[:]  # Copy the list using slicing

print(new_fruits)
```

Output:

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

10. Checking for List Membership

You can check whether an element exists in a list using the in keyword.

Example:

Python Code

```
fruits = ["apple", "banana", "cherry"]

if "banana" in fruits:
    print("Banana is in the list!")
else:
    print("Banana is not in the list.")
```

Output:

```
Banana is in the list!
```

11. Nested Lists

Lists can contain other lists (nested lists), which allows for the creation of 2D arrays or matrices.

Example:

Python Code

```
# Creating a 2D list (matrix)
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Accessing elements in a nested list
print(matrix[0][1])  # Access the element in the first row, second column
print(matrix[2][0])  # Access the element in the third row, first column
```

Output:

```
2
7
```

Explanation:

- matrix[0][1]: Accesses the second element in the first row.
- matrix[2][0]: Accesses the first element in the third row.

12. List Length

You can find the length of a list (number of elements) using the len() function.

Example:

python

Code

```
fruits = ["apple", "banana", "cherry"]
print("Number of fruits:", len(fruits))
```

Output:

javascript

Code

```
Number of fruits: 3
```

Explanation:

- len() returns the number of elements in the list.

13. List Functions and Methods Summary

Here is a list of common list methods and functions:

Method/Function	**Description**
append()	Adds an element to the end of the list.
insert()	Inserts an element at a specific index.
extend()	Adds elements of another list to the list.
remove()	Removes the first occurrence of a value.
pop()	Removes and returns the element at the specified index.
clear()	Removes all elements from the list.
index()	Returns the index of the first occurrence of a value.
count()	Returns the number of occurrences of a value.
sort()	Sorts the list in ascending order.
reverse()	Reverses the list.
copy()	Returns a shallow copy of the list.
len()	Returns the number of elements in the list.

Conclusion

Lists in Python are a flexible and powerful data structure that allows you to store and manipulate ordered collections of items. You can use lists for various purposes, such as storing a sequence of numbers, strings, or other data types. The wide variety of built-in methods makes list manipulation efficient and straightforward