List Sorting

Table of contents

01

Introduction

Introduction

Sorting is a fundamental operation in Python that allows arranging elements in a **specific order** (ascending or descending).

Python provides **two** primary ways to sort lists:

- **Using sort() method** → Modifies the original list **in place**.
- **Using sorted() function** → Returns a **new sorted list** without modifying the original one.

02

sort() Method

Using sort() Method

The .sort() method sorts a list **in place**, meaning the original list is modified and no new list is created.

Syntax

Python

```
list_name.sort(key=None, reverse=False)
```

Explanation:

- **list_name** → The list to be sorted.
- **key (optional)** → A function that defines the sorting criterion. Default is None (normal sorting).
- **reverse (optional)** → If True, sorts in **descending order**;
  If False, sorts in **ascending order** (default).

Example 1:

Objective:

Sorting a List of Strings (Alphabetically)

Python

```
fruits = ["orange", "mango", "kiwi", "pineapple", "banana"]
print("Before sorting:", fruits)
fruits.sort()
print("After sorting:", fruits)
```

Output

```
Before sorting: ['orange', 'mango', 'kiwi', 'pineapple', 'banana']
After sorting: ['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

Strings are sorted **lexicographically** (dictionary order).

Example 2:

Objective:

Sorting a List of Numbers (Numerically)

Python

```
numbers = [100, 50, 65, 82, 23]
print("Before sorting:", numbers)
numbers.sort()
print("After sorting:", numbers)
```

Output

```
Before sorting: [100, 50, 65, 82, 23]
After sorting: [23, 50, 65, 82, 100]
```

- Numbers are sorted from **smallest to largest**.

Sorting Lists in Descending Order

To sort a list in descending order, set **reverse=True**.

Example 1:

Objective:

Sorting Strings in Reverse Order

Python

```
fruits = ["orange", "mango", "kiwi", "pineapple", "banana"]
fruits.sort(reverse=True)
print(fruits)
```

Output

```
['pineapple', 'orange', 'mango', 'kiwi', 'banana']
```

- The list is sorted in reverse alphabetical order.

Example 2:

Objective:

Sorting Numbers in Descending Order

Python

```
numbers = [100, 50, 65, 82, 23]
numbers.sort(reverse=True)
print(numbers)
```

Output

```
[100, 82, 65, 50, 23]
```

- Numbers are sorted from largest to smallest.

03

sorted() Function

Using sorted() Function

The **sorted**() function returns a new sorted list while keeping the original list unchanged.

To sort a list in descending order, set **reverse=True**.

Syntax

Python

```
sorted(iterable, key=None, reverse=False)
```

Explanation:

- **iterable** → The list (or iterable) to be sorted.
- **key (optional)** → Function for custom sorting.
- **reverse (optional)** → If True, sorts in descending order.

Example 1:

Objective:

Sorting Without Changing the Original List

Python

```
numbers = [3, 1, 4, 1, 5, 9, 2]
sorted_numbers = sorted(numbers)  # Creates a new sorted list
print("Original list:", numbers)
print("Sorted list:", sorted_numbers)
```

Output

```
Original list: [3, 1, 4, 1, 5, 9, 2]
Sorted list: [1, 1, 2, 3, 4, 5, 9]
```

- The original list remains unchanged.

Example 2:

Objective:

Sorting in Descending Order

Python

```
ages = [25, 45, 32, 18, 90]
sorted_ages_desc = sorted(ages, reverse=True)
print(sorted_ages_desc)
```

Output

```
[90, 45, 32, 25, 18]
```

04

Custom sorting

Custom Sorting Using “key” Parameter

The **key** parameter in **sort**() and **sorted**() allows us to define custom sorting logic.

This function is applied to each element before sorting.

Example 1:

Objective:

Sorting based on length of word (using **.sort()** method)

Python

```
words = ["apple", "banana", "cherry", "date"]

words.sort(key=len)
print(words)
```

Output

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

Example 2:

Objective:

Sorting based on length of word (using **sorted**() function)

Python

```
words = ["apple", "banana", "cherry", "date"]

# Using sorted() with a key
sorted_words = sorted(words, key=len)
print(sorted_words)  # Output:
```

Output

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

Example 3:

Objective:

Sorting numbers based on their distance from 50.

Python

```
def fun(n):
    return n % 10

numbers = [100, 50, 65, 82, 23]
numbers.sort(key=fun)
print(numbers)
```

Output

```
[100, 50, 82, 23, 65]
```

Example 4:

Objective:

Case-Insensitive Sorting

Python

```
words = ["banana", "Orange", "Kiwi", "cherry"]
words.sort(key=str.lower)
print(words)
```

Output

```
['banana', 'cherry', 'Kiwi', 'Orange']
```

05

reverse() Method

Reversing a List Using .reverse()

The **.reverse()** method **flips** the order of elements but does not sort them.

Example:

Objective:

Reversing Without Sorting

Python

```
names = ["A", "d", "f", "E"]
names.reverse()
print(names)
```

Output

```
['E', 'f', 'd', 'A']
```

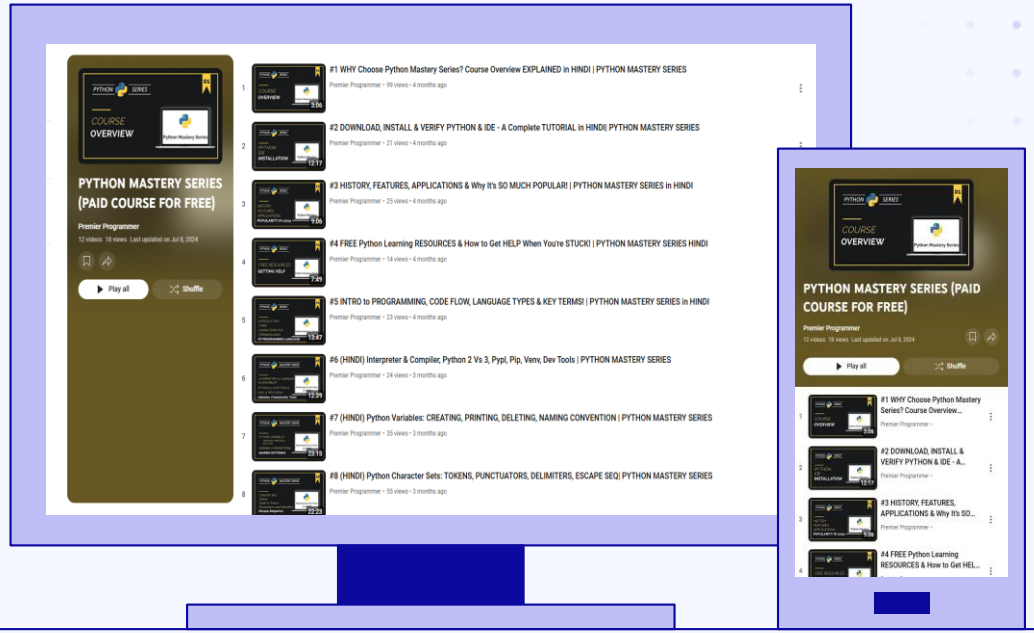
The order is **reversed**, but the list is **not sorted**.

Summary

- Use **.sort()** when you want to modify the original list.
- Use **sorted()** when you need a new sorted list without changing the original.
- Use **reverse=True** to sort in descending order.
- Use **key** parameter to define custom sorting logic.
- Use **.reverse()** if you only want to flip the order of elements.

WATCH

Level up your coding with each episode in this focused Python series.



Next Video!

Combining List