Here is the information about the sorted() function and the .sort() method in Python:

Both sorted() and .sort() are used for sorting data, but they have key differences:

sorted() Function:

- **Creates a New List:** The sorted() function takes any iterable (e.g., list, tuple, string) as input and returns a new sorted list, leaving the original iterable unchanged.
- **Versatile:** It can be used with any iterable, not just lists.
- **Syntax:** sorted(iterable, key=..., reverse=...)
- **Return Value:** Returns a new sorted list.

.sort() Method:

- **Modifies In-Place:** The .sort() method is a list method that sorts the list in place, meaning it changes the original list directly.
- **List-Specific:** It can only be used with lists.
- **Syntax:** list.sort(key=..., reverse=...)
- **Return Value:** Returns None (because it modifies the list directly).

Key Differences Summarized:

Feature	sorted()	.sort()
Modifies	Creates a new sorted list	Modifies the original list in place
Applicable To	Any iterable (list, tuple, string, etc.)	Only lists
Return Value	New sorted list	None

Common Parameters:

- key: A function that specifies how to extract the comparison key from each element in the iterable.
- reverse: A boolean value that indicates whether to sort in ascending (default) or descending order.

Example:

```
my_list = [3, 1, 4, 1, 5, 9, 2, 6]

# Using sorted()
sorted_list = sorted(my_list)
print(f"Original list: {my_list}")  # Output: Original list: [3, 1,
4, 1, 5, 9, 2, 6]
print(f"Sorted list (using sorted()): {sorted_list}") # Output:
Sorted list (using sorted()): [1, 1, 2, 3, 4, 5, 6, 9]

# Using .sort()
my_list.sort()
print(f"Sorted list (using .sort()): {my_list}") # Output: Sorted
list (using .sort()): [1, 1, 2, 3, 4, 5, 6, 9]
```

When to Use Which:

- Use sorted() when you need to preserve the original data and want a new sorted list.
- Use .sort() when you want to modify the original list directly and don't need to keep a copy of the unsorted version.