Sorting methods in Python - sort vs sorted

Sorting is an topic that is very important in the world of computer science. There are many algorithms available for sorting arrays of data such as merge sort, quick sort, bubble sort, selection sort, insertion sort, etc. All these algorithms have different time and space complexities, but in general, merge sort and quick sort are considered the best. Since sorting is a very common operation, programming languages contain in-build functions to sort data. And, Python is one of them.

But, there are two functions for sorting in Python, namely, `sort` and `sorted`. In this article, we will learn about the difference between these two sorting functions.

`sort` function in Python

The `sort` function in Python sorts the list `in place` by swapping the values at it indexes. When we call the `sort` function over a list, a new list is not returned, rather the same list is sorted. `sort` function can be used to sort lists.

Refer to the following code for an example.

```
myList = [1, 5, 2 ,10, 33, 3, 9]
print(myList)
myList.sort()
print(myList)
```

Output

```
[1, 5, 2, 10, 33, 3, 9]
[1, 2, 3, 5, 9, 10, 33]
```

As we can see, the same list was sorted when we called the `sort` function over it.

If we wish to sort the list in reverse order, we can do that as well by setting the `reverse` argument to `True`.

Refer to the following code for the same.

```
myList = [1, 5, 2 ,10, 33, 3, 9]
print(myList)
myList.sort(reverse = True)
print(myList)
```

Output

```

```

```
[1, 5, 2, 10, 33, 3, 9]
[33, 10, 9, 5, 3, 2, 1]
```

The `sort` function has yet another parameter, namely, `key`, which can be used to define what value to consider for sorting. For example, if we have a list of pairs and a we wish to sort the pairs considering the second value, we can do that using the `key` parameter.

Refer to the following code for an example.

```
def secondElement(element):
    return element[1]

data = [(1, 2), (3, 4), (5, 3), (6, 6), (3, 0), (10, 1)]
print("Before sorting:", data)
data.sort(key = secondElement)
print("After sorting:", data)
```

`Output`

```
Before sorting: [(1, 2), (3, 4), (5, 3), (6, 6), (3, 0), (10, 1)]
After sorting: [(3, 0), (10, 1), (1, 2), (5, 3), (3, 4), (6, 6)]
```

As we can see, the list is sorted based on the second number inside the pairs.

We can also sort a list of dictionaries using the same approach. Refer to the following code for the same.

```
def byName(entry):
    return entry["name"]

def byAge(entry):
    return entry["age"]

def byRank(entry):
    return entry["rank"]

data = [{
        "name": "Vaibhav",
        "age": 18,
        "rank": 1,
    }, {
        "name": "Steve",
        "age": 24,
        "rank": 5,
    }, {
        "name": "Olive",
        "age": 12,
```

```
        "rank": 2,
    },{
        "name": "Allison",
        "age": 17,
        "rank": 4,
    }]

print("Original Data")
print(data)
print("Sorted by Name")
data.sort(key = byName)
print(data)
print("Sorted by Age")
data.sort(key = byAge)
print(data)
print("Sorted by Rank")
data.sort(key = byRank)
print(data)
```

Output

```
Original Data
[{'name': 'Vaibhav', 'age': 18, 'rank': 1}, {'name': 'Steve', 'age': 24, 'rank':
5}, {'name': 'Olive', 'age': 12, 'rank': 2}, {'name': 'Allison', 'age': 17,
'rank': 4}]
Sorted by Name
[{'name': 'Allison', 'age': 17, 'rank': 4}, {'name': 'Olive', 'age': 12, 'rank':
2}, {'name': 'Steve', 'age': 24, 'rank': 5}, {'name': 'Vaibhav', 'age': 18,
'rank': 1}]
Sorted by Age
[{'name': 'Olive', 'age': 12, 'rank': 2}, {'name': 'Allison', 'age': 17, 'rank':
4}, {'name': 'Vaibhav', 'age': 18, 'rank': 1}, {'name': 'Steve', 'age': 24,
'rank': 5}]
Sorted by Rank
[{'name': 'Vaibhav', 'age': 18, 'rank': 1}, {'name': 'Olive', 'age': 12, 'rank':
2}, {'name': 'Allison', 'age': 17, 'rank': 4}, {'name': 'Steve', 'age': 24,
'rank': 5}]
```

As we can see, the entries inside the list of dictionaries are sorted based on the key we defined.

`sorted` function in Python

Like the `sort` function, `sorted` is also used to sort lists. But, the `sorted` function returns a new list instead of sorting the same list in place. This is the major difference between the two function. Let's understand this with an example.

```
myList = [1, 5, 2 ,10, 33, 3, 9]
print("Before called sorted():", myList)
newList = sorted(myList)
```

```
print("After called sorted():", myList)
print("New list:", newList)
```

Output

```
Before called sorted(): [1, 5, 2, 10, 33, 3, 9]
After called sorted(): [1, 5, 2, 10, 33, 3, 9]
New list: [1, 2, 3, 5, 9, 10, 33]
```

As we can see, the list we passed to the `sorted` function remains the same, and a new list was returned which is stored inside the `newList` variable. The two parameters, namely, `reverse` and `key`, are also available in this function. Using the two, we can get a sorted list in reverse order, and sort the list based on a custom key. The following examples depict the same.

Example for the `reverse` parameter.

```
myList = [1, 5, 2 ,10, 33, 3, 9]
print("Before called sorted():", myList)
newList = sorted(myList, reverse = True)
print("After called sorted():", myList)
print("New list:", newList)
```

Output

```
Before called sorted(): [1, 5, 2, 10, 33, 3, 9]
After called sorted(): [1, 5, 2, 10, 33, 3, 9]
New list: [33, 10, 9, 5, 3, 2, 1]
```

And, an example for the `key` parameter.

```
def byName(entry):
    return entry["name"]

def byAge(entry):
    return entry["age"]

def byRank(entry):
    return entry["rank"]

data = [{
        "name": "Vaibhav",
        "age": 18,
        "rank": 1,
    }, {
        "name": "Steve",
```

```
        "age": 24,
        "rank": 5,
    }, {
        "name": "Olive",
        "age": 12,
        "rank": 2,
    },{
        "name": "Allison",
        "age": 17,
        "rank": 4,
    }]

print("Original Data")
print(data)
print("Sorted by Name")
nameSorted = sorted(data, key = byName)
print(nameSorted)
print("Sorted by Age")
ageSorted = sorted(data, key = byAge)
print(ageSorted)
print("Sorted by Rank")
rankSorted = sorted(data, key = byRank)
print(rankSorted)
print("Data after all operations")
print(data)
```

Output

```
Original Data
[{'name': 'Vaibhav', 'age': 18, 'rank': 1}, {'name': 'Steve', 'age': 24, 'rank':
5}, {'name': 'Olive', 'age': 12, 'rank': 2}, {'name': 'Allison', 'age': 17,
'rank': 4}]
Sorted by Name
[{'name': 'Allison', 'age': 17, 'rank': 4}, {'name': 'Olive', 'age': 12, 'rank':
2}, {'name': 'Steve', 'age': 24, 'rank': 5}, {'name': 'Vaibhav', 'age': 18,
'rank': 1}]
Sorted by Age
[{'name': 'Olive', 'age': 12, 'rank': 2}, {'name': 'Allison', 'age': 17, 'rank':
4}, {'name': 'Vaibhav', 'age': 18, 'rank': 1}, {'name': 'Steve', 'age': 24,
'rank': 5}]
Sorted by Rank
[{'name': 'Vaibhav', 'age': 18, 'rank': 1}, {'name': 'Olive', 'age': 12, 'rank':
2}, {'name': 'Allison', 'age': 17, 'rank': 4}, {'name': 'Steve', 'age': 24,
'rank': 5}]
Data after all operations
[{'name': 'Vaibhav', 'age': 18, 'rank': 1}, {'name': 'Steve', 'age': 24, 'rank':
5}, {'name': 'Olive', 'age': 12, 'rank': 2}, {'name': 'Allison', 'age': 17,
'rank': 4}]
```