Theory: Sorting a list

18 minutes 6 / 8 problems solved

After you have learned how to create and perform basic operations with lists, it is time to explore how to put their items in the order you need. There are two similar ways to do this: the method `list.sort()` and the function `sorted(list)`. They have much in common and take the same arguments, so let's see how they work.

§1. The sorting

Invoking `list.sort()` as well as `sorted(list)` sorts the list in the *ascending order* according to the natural order of stored elements.

```
# Invoking list.sort()
numbers = [3, 2, 5, 4, 1]
numbers.sort()
print(numbers)  # [1, 2, 3, 4, 5]

# Invoking sorted(list)
numbers = [3, 2, 5, 4, 1]
print(sorted(numbers))  # [1, 2, 3, 4, 5]
```

Note that the `sort()` method performs an *in-place* sorting, changes the original list, and returns `None`. `sorted(list)`, on the other hand, creates a new sorted list while the original one remains unmodified.

That is why an attempt to use `print(numbers.sort())` will lead to the `None` result while the same attempt with `sorted(list)` gives you the result you need. This is the important distinction between `sorted(list)` and `list.sort()`.

```
print(numbers.sort()) # None
```

The `sorted` function has one more feature: it works not only with lists but also with other collections, so you can also sort sets, dictionaries, tuples, and so on. The `list.sort()` method, on the contrary, works only with lists.

To sort a list in the *descending order*, you need to specify the `reverse` argument as `True` (it's `False` by default).

```
numbers = [3, 2, 5, 4, 1]
print(sorted(numbers, reverse=True))  # [5, 4, 3, 2, 1]

# the same with list.sort()
numbers.sort(reverse=True)
print(numbers)  # [5, 4, 3, 2, 1]
```

Both `sorted(list)` and `list.sort()` can sort strings according to their lexicographic order (as in a dictionary).

```
strings = ['aa', 'b', 'aaa', 'q', 'qq']
strings.sort()
print(strings)  # ['aa', 'aaa', 'b', 'q', 'qq']
```

However, if your list has uncomparable types (like strings and integers together), an error will occur as well in `list.sort()` and `sorted(list)`:

```
TypeError: '<' not supported between instances of 'str' and 'int'
```

§2. The key

There is one more argument that can be used with both functions. You can pass the `key` function, this function will be applied to each element in the list, and the list will be sorted depending on the results. For example, you

Current topic:
- Sorting a list

Topic depends on:
- List 13★ Stage 1
- Arguments 7★ Stage 1
- Lambda functions
- Iterators
- Ordering and total order

can use `key=len` to sort words by their length.

```
names = ['Mary', 'James', 'Tom', 'Katarina', 'John']
names.sort(key=len)
print(names)  # ['Tom', 'Mary', 'John', 'James', 'Katarina']
```

You can as well use lambda functions as a `key`. For example, if you need to sort numbers by the remainder of dividing by two, you can use a lambda function and the `%` operator. In the example below, 4 is moved to the first place (the remainder is 0), followed by the three other digits with the remainder 1. Note that the relative order of 4, 7, and 5 in the sorted list remains as it was the initial one.

```
nums = [7, 4, 1, 5]
print(sorted(nums, key=lambda x: x % 2))  # [4, 7, 1, 5]
```

What is more, you can use custom functions for the value of the `key` argument. Let's say you have a list of float numbers and you want to sort these numbers by their fractional part. To do so, you can create a function that takes a number as its input and returns the fractional part of this number. The function below performs it by subtracting an integer part from the given number.

```
def my_sorted(x):
    return x - int(x)

numbers = [1.5, 3.2, 4.3]
print(sorted(numbers, key=my_sorted))  # [3.2, 4.3, 1.5]
```

§3. The reverse

There is one more way to modify the order of elements in a list; you can reverse them. It can be done with the help of either the `list.reverse()` method or by using the `reversed()` function.

Invoking `list.reverse()` reverses the order of elements in the list. Just as `list.sort()`, it operates *in place* and returns `None`, so you can't assign the result to another variable, instead, the initial list will be changed.

```
initial_list = [1, 2, 3, 4, 5]
reversed_list = initial_list.reverse()
print(reversed_list)  # None
print(initial_list)   # [5, 4, 3, 2, 1]
```

The function `reversed()` returns a reverse *iterator*, bypassing which you can get the elements of the input sequence in the reverse order. To get access to the reversed list, you can write the following:

```
numbers = [1, 2, 3, 4, 5]
for number in reversed(numbers):
    print(number)
# 5
# 4
# 3
# 2
# 1
```

Pay attention that if you try to print the iterator without using a loop, you will not get the needed result:

```

print(reversed(numbers))  # <list_reverseiterator object at 0x7fe25e718b70>
```

§4. Conclusion

In this topic, we have learned how to sort lists using the function `sorted()` and the method `list.sort()`, also the difference between them, and their arguments. We have also learned to reverse elements in a list with the help of the function and the method.

Table of contents:

To sum up:

- `list.sort()` and `list.reverse()` work only with list objects; they change the given list and return you nothing;
- `sorted()` takes any collection as an argument and creates and returns a new item from the given one;
- `reversed()` can also take any collection as an argument but returns an iterator through the reversed object rather than the modified object itself;
- `list.sort()` and `sorted()` can take the boolean parameter `reverse` to change the order of sorting from the ascending one to the descending one, as well as the parameter `key` to specify the sorting function.

For additional information about sorting methods, you can check out the Official Python documentation.

Report a typo

63 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion