Sorting Data in lists and sets

Let us go through the details about sorting data in lists and sets.

Contents

Task 1 - Sort employees based upon their salary

- We can use `sorted` function to sort any collection - list, set, dict or tuple. For now, let's focus on list and set.
- `sorted` always returns a new list. We typically assign it to a new variable to process the newly created sorted list further.
- On top of `list`, we can also invoke `sort` function. While `sorted` creates new list, `sort` will update the existing list. The sorting done by `sort` is also known as **inplace** sorting.
- While `sorted` returns a new list, `sort` on top of list returns nothing.
- Both `sorted` and `sort` takes same arguments.
- We can use `reverse` to sort in reverse order.
- We can also sort the data based upon comparison logic passed using `key` argument.
- We use `sorted` more often than `list.sort` for following reasons.
    - `sorted` can be used on all types of collections - list, set, dict, tuple or any other collection type.
    - The original collection will not be touched.
    - We can pass `sorted` function to other functions as part of chained calling. For example if we would like to get unique records after sorting, we can say `set(sorted(l))`. It is not possible with `list.sort`

```
# Run this to see the syntax of sorted
sorted?
```

```
Signature: sorted(iterable, /, *, key=None, reverse=False)
Docstring:
Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customize the sort order, and the
reverse flag can be set to request the result in descending order.
Type:      builtin_function_or_method
```

```
# Run this to see the syntax of sort function
list.sort?
```

```
Signature: list.sort(self, /, *, key=None, reverse=False)
Docstring: Stable sort *IN PLACE*.
Type:      method_descriptor
```

- Sorting a simple list using `sorted`

```
l = [1, 3, 2, 6, 4]
```

```
sorted(l)
```

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

```
type(sorted(l))
```

```
list
```

```
# l did not change
l
```

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

```
# Typical usage for further processing
l_sorted = sorted(l)
```

```
type(l_sorted)
```

```
list
```

```
l_sorted
```

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

- Sorting a simple list using `sort`

```
l = [1, 3, 2, 6, 4]
```

```
# We typically don't assign to another variable.
l.sort()
```

```
type(l.sort())
```

```
NoneType
```

```
l
```

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

- Sorting a set using `sort`

```
s = {1, 4, 2}
```

```
# This will fail as sort is available only on top of list but not set
s.sort()
```

```
sorted(s)
```

```
[1, 2, 4]
```

- Reverse sorting of a list or a set using `sorted`. Similar process can be followed for `list.sort` as well.

```
l = [1, 3, 2, 6, 4]
```

```
sorted(l, reverse=True)
```

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

Task 1 - Sort employees based upon their salary

Let us perform a task to sort employees based upon their salary.

- We will create employee list in the form strings.
- Employee list will contain employee id, email and salary.

```
employees = [
    '1,ktrett0@independent.co.uk,6998.95',
    '2,khaddock1@deviantart.com,10572.4',
    '3,ecraft2@dell.com,3967.35',
    '4,drussam3@t-online.de,17672.44',
    '5,graigatt4@github.io,11660.67',
    '6,bjaxon5@salon.com,18614.93',
    '7,araulston6@list-manage.com,11550.75',
    '8,mcobb7@mozilla.com,17016.15',
    '9,grobardley8@unesco.org,14141.25',
    '10,bbuye9@vkontakte.ru,12193.2'
]
```

- We need to sort the data by comparing salaries between employees.
- We can define custom comparitor using `key` argument.
- Each element or record in the list is comma seperated.
- We need to extract the salary as float for right comparison.
- Here is how we can extract the salary.

```
# Reading first element
employees[0]
```

```
'1,ktrett0@independent.co.uk,6998.95'
```

```
emp = employees[0]
```

```
type(emp)
```

```
str
```

```
# We can use split with ',' as delimiter.
# It will create a list of strings.
# The list contains 3 elements - id, email and salary
# All 3 will be of type string
emp.split(',')
```

```
['1', 'ktrett0@independent.co.uk', '6998.95']
```

```
emp_list = emp.split(',')
```

```
type(emp_list)
```

```
list
```

```
for e in emp_list:
    print(f'Data type of {e} is {type(e)}')
```

```
Data type of 1 is <class 'str'>
Data type of ktrett0@independent.co.uk is <class 'str'>
Data type of 6998.95 is <class 'str'>
```

```
# Getting salary
emp_list[2]
```

```
'6998.95'
```

```
# We can also -1 to read from the last
emp_list[-1]
```

```
'6998.95'
```

```
# We need to change the data type to float or decimal for right comparison.
float(emp_list[-1])
```

```
6998.95
```

```
# Complete logic
float(emp.split(',')[-1])
```

```
6998.95
```

```
# We can pass the comparison logic to key function in sorted
# You can see the output. It is sorted in ascending order by salary.
sorted(employees, key=lambda emp: float(emp.split(',')[-1]))
```

```
['3,ecraft2@dell.com,3967.35',
 '1,ktrett0@independent.co.uk,6998.95',
 '2,khaddock1@deviantart.com,10572.4',
 '7,araulston6@list-manage.com,11550.75',
 '5,graigatt4@github.io,11660.67',
 '10,bbuye9@vkontakte.ru,12193.2',
 '9,grobardley8@unesco.org,14141.25',
 '8,mcobb7@mozilla.com,17016.15',
 '4,drussam3@t-online.de,17672.44',
 '6,bjaxon5@salon.com,18614.93']
```

```
# You can reverse by using reverse keywork argument
# reverse will be applied on custom comparison passed as part of key
sorted(employees, key=lambda emp: float(emp.split(',')[-1]), reverse=True)
```

```
['6,bjaxon5@salon.com,18614.93',
 '4,drussam3@t-online.de,17672.44',
 '8,mcobb7@mozilla.com,17016.15',
 '9,grobardley8@unesco.org,14141.25',
 '10,bbuye9@vkontakte.ru,12193.2',
 '5,graigatt4@github.io,11660.67',
 '7,araulston6@list-manage.com,11550.75',
 '2,khaddock1@deviantart.com,10572.4',
 '1,ktrett0@independent.co.uk,6998.95',
 '3,ecraft2@dell.com,3967.35']
```

By Durga Gadiraju