Python Fundamentals day 58

Today's Agenda

- Sorting
- Sorting algorithms

Sorting

Sorting is a process of arranging items systematically, be it in ascending order or descending order. In python we can sort built-in objects as well as user defined objects.

Let us understand with the basic example first

```
lst=[20,50,40,30,10,60]
print(lst)
lst.sort()
print(lst)
```

Output:

```
In [1]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
[20, 50, 40, 30, 10, 60]
[10, 20, 30, 40, 50, 60]
```

By default the elements in the list get sorted in ascending order. What to do to get it in descending order let us see below

```
lst=[20,50,40,30,10,60]
print(lst)
lst.sort(reverse=True)
print(lst)
```

Output:

```
In [2]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
[20, 50, 40, 30, 10, 60]
[60, 50, 40, 30, 20, 10]
```

Next let us try with strings

```
lst=['python','java','c','abc','ruby']
print(lst)
lst.sort()
print(lst)
```

Output:

```
In [3]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
['python', 'java', 'c', 'abc', 'ruby']
['abc', 'c', 'java', 'python', 'ruby']
```

The string values are sorted in alphabetical order. And just like numbers we can have the reverse order of strings as well

```
lst=['python','java','c','abc','ruby']
print(lst)
lst.sort(reverse=True)
print(lst)
```

Output:

```
In [4]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
['python', 'java', 'c', 'abc', 'ruby']
['ruby', 'python', 'java', 'c', 'abc']
```

So far we have sorted homogeneous data, what if we provide heterogeneous data? Combination of multiple datatypes. Let us see

```
lst=['python',30,10,'java','c',20,'abc',40,'ruby']
print(lst)
lst.sort(reverse=True)
print(lst)
```

Output:

```
  File "C:/Users/rooman/OneDrive/Desktop/python/test.py",
line 3, in <module>
    lst.sort(reverse=True)

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

So as we can see heterogeneous sorting cannot happen, because we cannot measure two different set of data.

Sorting Algorithms

There are different algorithms (step by step procedure) used for sorting, below are few algorithms mentioned

- Bubble sort
- Mere sort
- Insertion sort
- Quick sort
- Selection sort
- Heap sort
- Radix sort
- Bucket sort

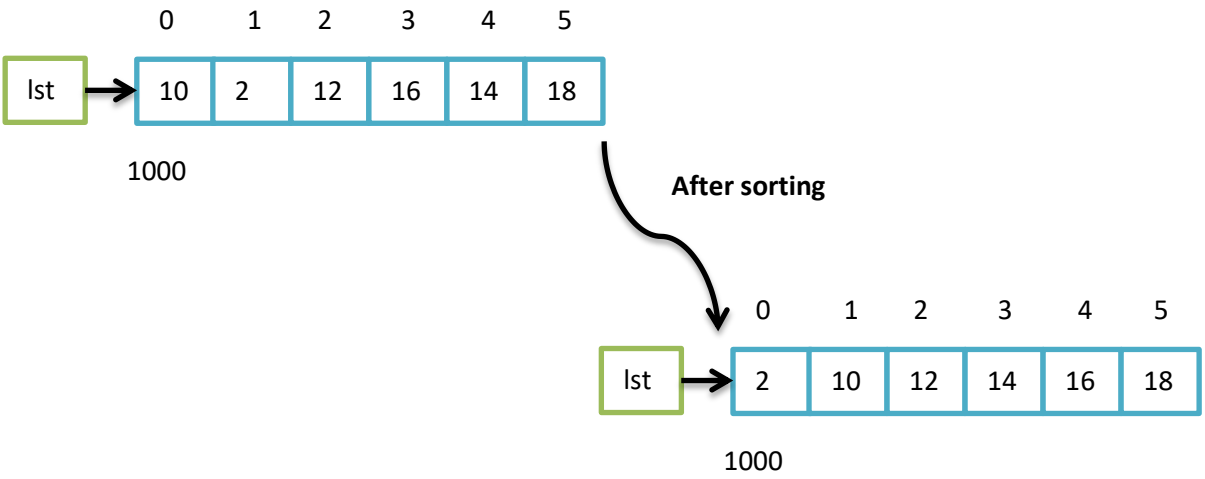
Amongst all let us select selection sort and know about it.

Selection sort:

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1)The subarray which is already sorted.
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.



Boundary condition

```
i : range(0,len(lst)-1)
j : range(i+1, len(lst))
```

Condition

```
if lst[j]<lst[i]:
    lst[i],lst[j]=lst[j],lst[i]
```

Let us try to implement the above code:

```
lst=[18,12,2,16,10,14]
print(lst)

for i in range(0,len(lst)-1):
    for j in range(i+1, len(lst)):
        if lst[j]<lst[i]:
            lst[i],lst[j]=lst[j],lst[i]

print(lst)
```

Output:

```
In [6]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
[18, 12, 2, 16, 10, 14]
[2, 10, 12, 14, 16, 18]
```

Next let us see the selection sort for user defined objects. For that first we have to create the user defined objects and then shall proceed with sorting

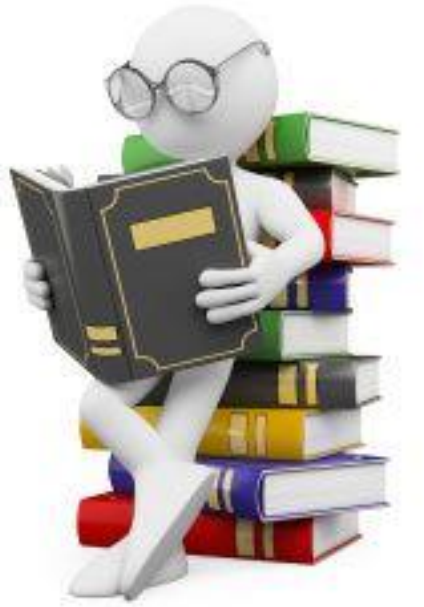
```
class Footballer:

    def __init__(self,name,goals,asist):
        self.name=name
        self.goals=goals
        self.asist=asist

    def display(self):
        print(self.__dict__)

def main():
    f1=Footballer('Messi',650,359)
    f2=Footballer('Cristiano',750,250)
    f1.display()
    f2.display()

main()
```

Output:

```
In [7]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
{'name': 'Messi', 'goals': 650, 'asist': 359}
{'name': 'Cristiano', 'goals': 750, 'asist': 250}
```

The user defined objects are ready, but can we just sort them by using any operators we used for built-in objects? Certainly not! So what to do?

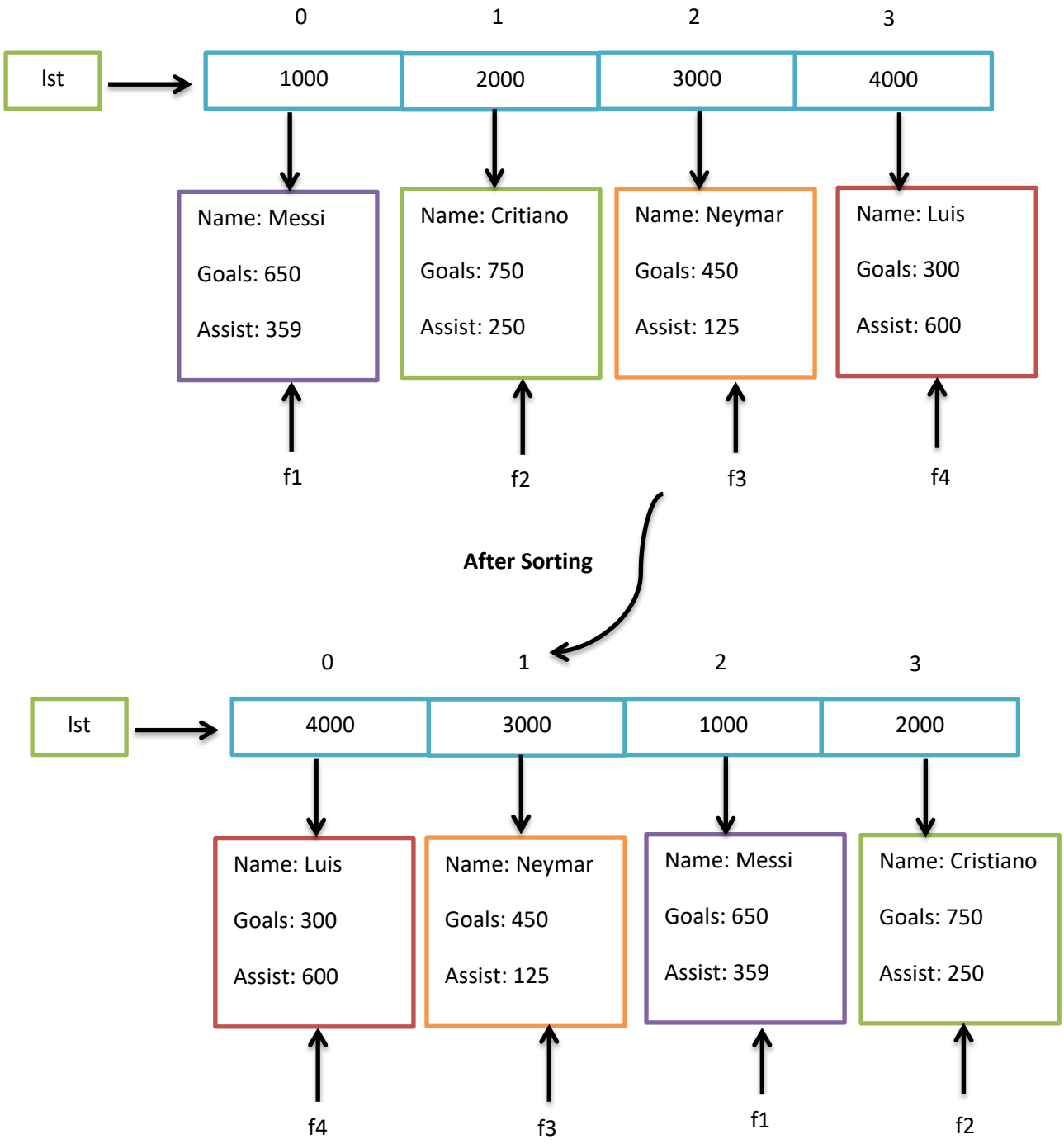
```
class Footballer:

    def __init__(self,name,goals,asist):
        self.name=name
        self.goals=goals
        self.asist=asist

    def display(self):
        print(self.__dict__)

    def __lt__(self,other):
        if self.goals < other.goals:
            return True
        else:
            return False

    def __gt__(self,other):
        if self.goals > other.goals:
            return True
        else:
            return False

def main():
    f1=Footballer('Messi',650,359)
    f2=Footballer('Cristiano',750,250)
    f1.display()
    f2.display()
    print(f1<f2) #f1.__lt__f2
    print(f1>f2) #f1.__gt__f2

main()
```



Output:

```
In [8]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
{'name': 'Messi', 'goals': 650, 'asist': 359}
{'name': 'Cristiano', 'goals': 750, 'asist': 250}
True
False
```

Now that we know how to compare user defined objects, let us see how to sort them



```
class Footballer:

    def __init__(self,name,goals,asist):
        self.name=name
        self.goals=goals
        self.asist=asist

    def display(self):
        print(self.__dict__)

    def __lt__(self,other):
        if self.goals < other.goals:
            return True
        else:
            return False

    def __gt__(self,other):
        if self.goals > other.goals:
            return True
        else:
            return False

    def __str__(self):
        return f'{self.name} {self.goals} {self.asist}'

def sort_footballer(lst):
    for i in range(0,len(lst)-1):
        for j in range(i+1,len(lst)):
            if lst[j]<lst[i]:
                lst[i],lst[j]=lst[j],lst[i]
```

```
def main():
    f1=Footballer('Messi',650,359)
    f2=Footballer('Cristiano',750,250)
    f3=Footballer('Neymar',450,125)
    f4=Footballer('Luis',300,600)
    lst=[f1,f2,f3,f4]
    sort_footballer(lst)

    for i in lst:
        print(i)

main()
```

Output:

```
In [13]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
Luis 300 600
Neymar 450 125
Messi 650 359
Cristiano 750 250
```

Great! But is this the optimized code? Definitely not

```
class Footballer:

    def __init__(self,name,goals,asist):
        self.name=name
        self.goals=goals
        self.asist=asist

    def __lt__(self,other):
        return self.goals < other.goals

    def __gt__(self,other):
        return self.goals > other.goals

    def __str__(self):
        return f'{self.name} {self.goals} {self.asist}'

def main():
    f1=Footballer('Messi',650,359)
    f2=Footballer('Cristiano',750,250)
    f3=Footballer('Neymar',450,125)
    f4=Footballer('Luis',300,600)
    lst=[f1,f2,f3,f4]
    lst.sort()

    for i in lst:
        print(i)

main()
```

Output:

```
In [14]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
Luis 300 600
Neymar 450 125
Messi 650 359
Cristiano 750 250
```