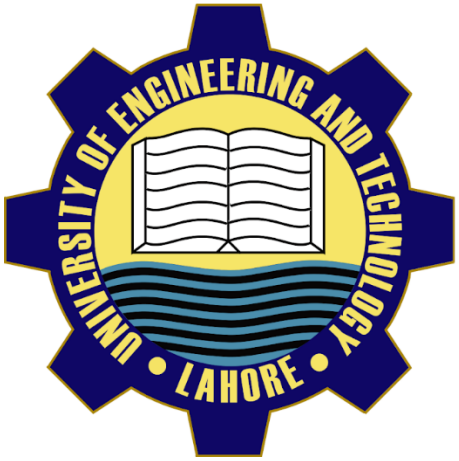
MCT-243 : COMPUTER PROGRAMMING-II

using Python 3.9



Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

Lab 29: Anonymous Functions & Sorting

Lambda Expression or Anonymous Functions:

Lambda Expression or **Anonymous Functions** are the functions with no names. They are used to implement very simple, single expression functions which are not meant to be used quite often. Let's start with simple example.
If we want a function that will take one integer as input and will return the unit digit, we can write it in normal way as:

```
def uDigit(x):
    return x%10
```

As the function is evaluating only a single expression, hence it's better to write it as **Lambda Expression** or **Anonymous Function** which is given as under:

```
lambda x:x%10
```

`lambda` is just a keyword and the function has no name. Variable before `:` e.g. `x` in above case is the input argument and the expression after `:` is evaluated and returned by the function.

How to use lambda function?

To use above lambda function for some value say `4395`, we must write the Lambda Expression as:

```
a=(lambda x:x%10)(4395)
```

The lambda expression written inside `()` is evaluated for `4395` and results is assigned to `a`. Just to see the data type and value of `a` we can write:

```
print(type(a))
print(a)
```

And it will give:

```
<class 'int'>
5
```

But is there any advantage of using lambda expression in above case? Why not just:

```
a=4395%10
```

The answer is that the above method is not the best way these Lambda Expression are used. The best use will be explored soon. Let's see that it is also possible to assign Lambda Expression to a variable and then use the Lambda Function through that variable. For example:

```
a=lambda x:x%10
print(type(a))
```

Now we are not evaluating the function. And if we see the output where data type of `a` is printed, we will see that it as `function` datatype:

```
<class 'function'>
```

And now we can use it like any other function as shown:

```
a=lambda x:x%10
```

```
x=a(4395)
y=a(876)
z=a(556737)
print(x,y,z)
```

This will have the output:

```
5 6 7
```

Here apparently the function is no more anonymous, rather it has been assigned to a variable with name. Actually, it still is anonymous but is assigned to an identifier **a**.

So, why should we use these Lambda Expressions?

There can be three possible uses of Lambda Expressions:

- When a function is of one single expression, writing it as Lambda Expression saves lines of code (not a very big advantage though).
- When we need to pass a function as input argument of another function, Lambda Expressions are simpler. (Explained in example below)
- The best use of Lambda Expressions is using it with some other built-in functions like **sort**, **map**, **filter**, **reduce** etc. (Explained in next topic)

The second point is explained here:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11)) #Will print 22
print(mytripler(11)) #Will print 33
```

myfunc is a regular function but has a Lambda Expression in **return** statement that involves one input argument as **a**. Now in **mydoubler = myfunc(2)**; **n** gets the values **2** and later when **mydoubler** is used with **11** as input argument; **a** gets the value **11** and **22** will be printed. Basically, we can create multiple functions by setting different values of **n** from the original function.

Even in above example, one can think of writing a function with two input arguments and pass **2** as second argument to get doubler or pass **3** as second argument to get tripler. It's all about personal preference but these Lambda Expressions are used mostly with other function e.g. **sort**, **map**, **filter**, **reduce** and is explained in next topic.

At this stage you must get familiar with syntax of these Lambda Expressions. Before moving to next section, note that we can have no or more than one input arguments for Lambda Expressions. One example is shown here:

```
z=(lambda x,y: (x+y)/2)(10,20)
print(z) #Will print 15.0
```

Tasks:

[1] Consider that information of a student is defined as a dictionary. One example student is as under:

```
student1={
    'Reg':'2018-MC-01',
    'Name':'Muhammad Usman',
    'Sec':'A',
    'Courses':['CP2','ES']}
```

Write a Lambda Expression that will take two student variables as input and will return a list of subjects registered by both students. Use the Lambda Expression on two students to verify the output.

SORTING DATA

Sorting of data is generally needed quite often in all kinds of data applications ranging from small datasets to quite large ones. Let's see sorting techniques in different data types.

Sorting a List:

There are two ways to sort a list:

1. Using Python built-in function named as **`sorted()`**. While using this function a list is provided as input argument and the result must be stored in some variable. This function will not change the original list. Example is shown here:

```
nums=[9,12,3,19,2,5,7]
s=sorted(nums)
print(s) #Prints sorted list
print(nums) #Prints original unchanged list
```

This method is preferred when we don't want to change the original list.

2. The second method is to use **`sort()`** method within list class. This method is applied on a list and the list itself is changed to a sorted list. The original unsorted list will no longer be available. An example is show here:

```
nums=[9,12,3,19,2,5,7]
nums.sort()
print(nums) #Prints the sorted list
```

To sort a list in reverse order we must pass in another parameter named as **`reversed`** as **`True`**. Default value of **`reversed`** in **`False`** so when we do not pass this parameter, it is taken as **`False`** and list is sorted in ascending order. Examples of both methods given here:

```
nums=[9,12,3,19,2,5,7]
s=sorted(nums, reverse=True)
print(s) #Prints sorted list in reverse order
print(nums) #Prints original unchanged list
nums.sort(reverse=True)
```

```
print(nums) #Prints sorted list in reverse order
```

Sorting a Tuple:

Tuple is immutable and unchangeable data type in Python. When it is unchangeable, can we apply sorting on it? Let's apply the built-in `sort()` function of Python on a tuple and see the results:

```
nums=(9,12,3,19,2,5,7)
s=sorted(nums)
print(s)
print(type(nums))
print(type(s))
```

The code has following output:

```
[2, 3, 5, 7, 9, 12, 19]
<class 'tuple'>
<class 'list'>
```

You can clearly see from the output that `sort()` method can be applied on a tuple but the result will be a sorted list. See carefully the data types of the two variables.

Also note that the original Tuple is unchanged. Now what if we want to have result as sorted tuple? We can use Python casting feature to cast a list to a tuple as described here:

```
nums=(9,12,3,19,2,5,7)
s=tuple(sorted(nums))#s will be tuple now
print(s)
print(type(nums))
print(type(s))
```

This will have the output:

```
(2, 3, 5, 7, 9, 12, 19)
<class 'tuple'>
<class 'tuple'>
```

To sort a tuple in reverse order, we can use `reversed` argument as we used in case of list.

Is there class method in tuple as in case of list?

The best way to answer the above question is to view all class methods of tuple class. We can see class methods of any class using `dir()` method and passing in the name of the class or any variable of that class. To see tuple class methods, we can write:

```
print(dir(tuple))
```

It will have this output:

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'count', 'index']
```

Ignore all methods surrounded between double underscores as they are special methods and we will study those later in the course. Other than those special methods, there are only two class methods for tuple; **`count`** and **`index`**.

Hence there is no sorting method available in the tuple class. If you are required sort the tuple by overwriting the original form of it, we can use the default sort function and assign the result to same variable as:

```
nums=(9,12,3,19,2,5,7)
nums=tuple(sorted(nums))
print(nums)
```

You might be wondering here that original Tuple has changed here violating the unchangeability feature of Tuple. Basically when we use assignment, as being used in above case, it creates a new object and if there was any other object using the same identifier, that is simply lost.

Finally, if someone is really fan of the list sort method, one can always cast a tuple to list, use list sort method and cast back the result to tuple as shown here:

```
nums=(9,12,3,19,2,5,7)
list(nums).sort()
tuple(nums)
print(nums)
```

Sorting a set:

On similar pattern, one can verify sort method for set datatype and it is basically exactly same as we have explore in case of tuple i.e. we get a sorted list of the set elements. See the similar code here:

```
nums={9,12,3,19,2,5,7}
s=sorted(nums)
print(s)
print(type(nums))
print(type(s))
```

Again, there is no sorting method in set class and you can always use different castings as explained in the case of tuple.

Sorting a Dictionary:

A dictionary is a collection of key-value pairs. We can apply python built-in **`sort()`** function on dictionary and the results will be a list of sorted keys. See this code:

```
products={'walnuts':1500,'cashew':1800,'almond':2000,'pine nuts':8000
}
s=sorted(products)
print(s)
```

The output is:

```
['almond', 'cashew', 'pine nuts', 'walnuts']
```

We can get the sorted value as well instead of the keys by using **`values()`** method of dictionary as:

```
s=sorted(products.values())
```

There is also a way to get both keys and value of a dictionary by using `items()` method of the dictionary class. The output will be a list of tuples where each tuple has two elements; first key and second value and list will be sorted based on keys. See the following code:

```
products={'walnuts':1500,'cashew':1800,'almond':2000,'pine nuts':8000
}
s=sorted(products.items())
print(s)
```

The output is:

```
[('almond', 2000), ('cashew', 1800), ('pine nuts', 8000), ('walnuts',
1500)]
```

Can we sort a dictionary items as tuples but based on the values instead of keys?

There is no direct way but we will see a method using the `key` parameter of the `sorted` function.

Sorting a list of Tuple, Set, Dictionary or Inner lists:

In many cases we encounter a situation where we have a list of other data types. We can apply sort method on that list as well and one such case of list of tuples is done in previous example of tuples which get sorted based on first element of the tuples and hence is not explained further.

List of inner Lists:

A list of inner lists can be sorted using sort method. The sorting will be based on first element of that list.

```
aList=[[10,4,3],[2,4,1,8],[3,2,1]]
aList.sort()
print(aList)
```

The output is:

```
[[2, 4, 1, 8], [3, 2, 1], [10, 4, 3]]
```

Just to see a case where we have different data types in inner lists as:

```
aList=[[10,4,3],[2,4,1,8],['Python',2,1]]
aList.sort()
```

This is going to generate an error because a string in third list cannot be compared with integers. You will see this error:

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

However for following case where the string element is not the first element of any list, this will work:

```
aList=[[10,4,3],[2,4,1,8],[1,'Python',2,1]]
aList.sort()
print(aList)
```

The output will be:

```
[[1, 'Python', 2, 1], [2, 4, 1, 8], [10, 4, 3]]
```

List of Sets:

We can apply sort method on a list of sets but because of indeterministic order nature of a set it makes no sense to sort a list of sets.

List of Dictionaries:

Now let's consider a list of dictionary as:

```
student1={
    'Reg':'2018-MC-01',
    'Name':'Muhammad Usman',
    'Sec':'A',
    'Courses':['CP','ED','EM']
    }
student2={
    'Reg':'2018-MC-02',
    'Name':'Tahir Mehmood',
    'Sec':'A',
    'Courses':['CP','DLD','EM']
    }
student3={
    'Reg':'2018-MC-03',
    'Name':'Muhammad Bilal',
    'Sec':'A',
    'Courses':['VCA','ED','EM']
    }
allStudents=[student1,student2,student3]
```

Here **`allStudents`** is a list having three dictionaries in it. If we try to apply list **`sort()`** method on this list as:

```
allStudents.sort()
```

This will generate an error:

```
TypeError: '<' not supported between instances of 'dict' and 'dict'
```

The error says that less than operation is not supported in dictionary. The interpreter doesn't know which dictionary element is lesser or greater than other.
Hence, we cannot apply sort method on a list of dictionary. But if we really need to sort it on the bases of a particular key of dictionary e.g. **`'Reg'`**, then there are ways which we will explore in next section.

Using **`key`** in sorting:

Suppose that we have a list of few strings and we are sorting the list:

```
a=['Computer','Programming','Python','Pakistan','UET','Mechatronics']
b=sorted(a)
print(b)
```

This will be the output:

```
['Computer', 'Mechatronics', 'Pakistan', 'Programming', 'Python',
'UET']
```

You can see the list of strings is sorted alphabetically. Now suppose that our requirement is to sort the list based on the number of characters in the strings i.e., from shortest string to longest string.

How can we do this?

For above task of sorting the list of strings based on the length of the list, there could be some quite lengthy approach of first getting a list of length of all strings, then zipping the original list with the list of lengths and applying the sorting then and finally taking out just the string form the sorted result.

But there is a much simpler way and that is by using an optional parameter within sort function; available both in default Python library and list class method. Let's see the help of default **`sorted`** method in Python as:

```
help(sorted)
```

This will be the output:

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    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.
```

Hopefully you understand the use of **`/`** and **`*`** in function definition. See carefully that there is an optional parameter named as **`key`** and sort function will sort the iterable based on the **function** provided as key. In simple worlds, Python will apply the function provided as **`key`** on all elements and will sort the elements based on the result of that function.

So, for the giving problem of sorting the list based on the length of the string, we know that we have a built-in function **`len`** that gives the length of the string when applied on it. We can provide this function as **`key`** and will get the desired result.

```
a=['Computer','Programming','Python','Pakistan','UET','Mechatronics']
b=sorted(a,key=len)
print(b)
```

Note that we just need to provide the name of the function as key without the parenthesis. We will get this result:

```
['UET', 'Python', 'Computer', 'Pakistan', 'Programming',
'Mechatronics']
```

We can still use the optional argument **`reverse`** to sort it descending order.

Before we explore more complex criterion, we can achieve using the **key** argument, let's first see another simple example.
Let's consider an e
xample where we have a list of numbers as:

```
nums=[5,-3,8,-2,0,7]
```

And we want to sort the list based on the magnitude of the numbers i.e. without considering the sign of the numbers. This is what we want at the end:

```
[0,-2,-3,5,7,8]
```

See carefully that signs of numbers are not removed but the sorting is based on magnitude.

We know that we have a built-in function **abs** that returns the magnitude of a number. Hence, we can achieve the required result as:

```
nums=[5,-3,8,-2,0,7]
a=sorted(nums,key=abs)
print(a)
```

The output is:

```
[0, -2, -3, 5, 7, 8]
```

The same is done here with sort function within list class:

```
nums=[5,-3,8,-2,0,7]
nums.sort(key=abs)
print(nums)
```

More Examples:

For a list containing different numbers, sort it based on unit digit of the number. For example, if we have list as:

```
a=[2314,432,675,9983]
```

Then it should get sorted as:

```
[432, 9983, 2314, 675]
```

For this we can first create a function that returns unit digit of a number and then pass it as **key** while sorting. The code is given here:

```
def uDigit(x):
        return x%10

a=[2314,432,675,9983]
a.sort(key=uDigit) #or a=sorted(a,key=uDigit)
print(a)
```

This is the situation where we should prefer to use Lambda Expression as given below:

```
a=[2314,432,675,9983]
a.sort(key=lambda x: x%10) #or a=sorted(a,key=lambda x: x%10)
print(a)
```

In last section we checked that sorting a list of dictionaries is not possible. The code we tried is given here once again:

```
student1={
    'Reg':'2018-MC-01',
    'Name':'Muhammad Usman',
    'Sec':'A',
    'Courses':['CP','ED','EM']
    }
student2={
    'Reg':'2018-MC-02',
    'Name':'Tahir Mehmood',
    'Sec':'A',
    'Courses':['CP','DLD','EM']
    }
student3={
    'Reg':'2018-MC-03',
    'Name':'Muhammad Bilal',
    'Sec':'A',
    'Courses':['VCA','ED','EM']
    }
allStudents=[student1,student2,student3]
allStudents.sort() # Will generate an error
```

Firstly, don't get confused with term **key** as it is same for the argument in sort function and also used to describe key of a dictionary. The problem in above program is that the **`sort()`** function don't know by which dictionary key or value it needs to sort the list. We can use the **`key`** argument within sort and specify that e.g., to sort the students based on their names we can write code as shown here:

```
s=sorted(allStudents,key=lambda x: x['Name'])
for e in s:
        print(e)
```

It will have following output which is sorted based on Name key:

```
{'Reg': '2018-MC-03', 'Name': 'Muhammad Bilal', 'Sec': 'A',
'Courses': ['VCA', 'ED', 'EM']}
{'Reg': '2018-MC-01', 'Name': 'Muhammad Usman', 'Sec': 'A',
'Courses': ['CP', 'ED', 'EM']}
{'Reg': '2018-MC-02', 'Name': 'Tahir Mehmood', 'Sec': 'A', 'Courses':
['CP', 'DLD', 'EM']}
```

Tasks:

[2] The task is related to same list of student dictionary as in previous example but before starting the task, add a couple of more students and vary the number of subjects registered by students. Then sort the list based on number of subjects registered by the student in reverse order.

[3] Write program that will sort a list of inner lists based on the second element of the inner list. For example if the original list is:

```
numList=[[4,2,1,3,7],[3,1,7],[8,5,9,11],[9,7]]
```

The final list after sort will be:

```
[[3,1,7], [4,2,1,3,7], [8,5,9,11], [9,7]]
```

[4] Recall the task related to **Survey for favorite Beverages** we did in previous course and given here again:

Write a program that performs a survey tally on beverages. The program will ask user to enter favorite beverage of first person and then should prompt for the next person until a value 0 is entered to terminate the program. Each person participating in the survey should choose their favorite beverage from the following list:

1. Coffee 2. Tea 3.Coke 4. Orange Juice

After the survey is complete indicated by entering a 0, the program should display the result in form of total number of persons participating in the survey and number of votes against each beverage in **ascending order.**

Sample output is:

```
************************************************************
1. Coffee        2. Tea         3.Coke         4. Orange Juice
************************************************************
This is survey for the favorite Beverage.

Choose (1-4) from the above menu or 0 to exit the program.

Please input the favorite beverage of person #1: 4
Please input the favorite beverage of person #2: 1
Please input the favorite beverage of person #3: 3
Please input the favorite beverage of person #4: 1
Please input the favorite beverage of person #5: 1
Please input the favorite beverage of person #6: 0


The results are as follows:
The total number of people participated: 5

Beverage         Number of Votes
*******************************
Coffee           3
Coke             1
Orange Juice     1
Tea              0
```

It was asked to print the beverage vote count in ascending order but later we said it is quite difficult to sort and ignored this condition. Now redo this task by incorporating the Dictionary in proper way and display the results in sorted form.

Multi-Level Sorting:

You were given the task of sorting in the quiz. There I realize that there should be second level of sorting as well. For example, if number of **Credit Hours** or the **GPA** of two students is exactly same then they can be sorted according to registration number or names.

The solution is very simple. If the function needed as input argument for the `sort()` method has two outputs, the elements of the iterable will get sorted based on first output argument and if first output for two elements is same, they will get sorted based on second output. We can have three, four or any higher-level sorting. Very easy to understand examples are shown here:

```
a=[[1,2,3],[5,2,8,7],[1,3,1,7],[5,3,2],[8,2,1,5,1],[5,2,1]]

#Sort a based on second element of inner list
b=sorted(a,key=lambda x: (x[1]))
print(b)

#If second element is same, sort based on total length of inner list
c=sorted(a,key=lambda x: (x[1],len(x)))
print(c)

#If length is also same, sort based on third element of inner list
d=sorted(a,key=lambda x: (x[1],len(x),x[2]))
print(d)

#If length of inner list needed to be considered in descending order
d=sorted(a,key=lambda x: (x[1],-len(x),x[2]))
print(d)
```