MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9

Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

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

LAB 16: LIST: CLO-4

Recall the task we did in one lab for entering marks of five subjects and calculating the mean of those. Firstly, we did it by declaring 5 variables and later we did it using a **`for`** loop and summing up all values. What if you want to store the marks of the whole class comprising of 50 students? Declaring 50 variables for this would not be a good solution. Secondly, using a **`for`** loop even for the case of 5 subject marks, the individual marks were never saved, we just kept adding entered marks and could not access the individual entries.

The solution for such cases is the use of **`list`** which is defined as single variable but can hold multiple values in it. The entries of **`list`** are generally known as elements or items.

Defining and printing a list:

A list can be defined by specifying its elements inside **`[]`** as shown here:

```
a=[5,300,-20]
```

Here **`a`** is the name of the list having three numbers. Another example with string elements is shown here:

```
b=['Python','Computer','Programming','Course']
```

Interestingly, we can have elements of a list with different data types as shown below:

```
c=[10,20,'Programming',100]
```

Printing a list is as simple as printing any variable. Use **`print(a)`** for the first list and respective list names for other two to see the displayed output.

Accessing and Updating list elements:

We can always access the individual elements of a list using the index of the element inside **`[]`**. The index of the list elements starts from **`0`**. So, if we want to access second element of a list it can be done as:

```
a=[5,300,-20]
print(a[1])
```

Of course, other than printing **`a[1]`** we can always use it as any other variable for example; **`x=a[1]+10`** or storing the user input.

We can also update the value of any element as shown here:

```
a=[5,300,-20]
a[2]=20
print(a)
```

You will see in the output that -20 is changed to 20.

If you will try to access some element beyond the index range of that list, you will get this error:

```
IndexError: list index out of range
```

Can the index be negative?

In most of the programming languages it cannot be. But Python has the feature of negative indices in reverse direction. Each element can be accessed through two indices. One, the positive index in forward direction as seen in above examples and the other is a negative index in reverse direction. The last element of the list has index -1, second last has -2 and so on till the first element. So, a list shown below has two set of indices:

```
myList=[30,45,"Hello","Python",950]
```

List Elements	30	45	Hello	Python	950
Positive Indices	0	1	2	3	4
Negative Indices	-5	-4	-3	-2	-1

List Slicing

Other than one element from of the list, we can access multiple elements at a time, known as **List Slicing**. Recall how we used `range()` function earlier and we can provide start, stop and step parameter. Likewise, as List indices, we can specify those by providing start, stop and step values.

Consider these examples:

```
a=[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9] Positive Indices
#[-10,-9,-8,-7,-6,-5,-4,-3,-2,-1] Negative Indices

print(a[0:4])        # Will print [1, 2, 3, 4]
print(a[2:4])        # Will print [3, 4]
print(a[0:9:2])      # Will print [1, 3, 5, 7, 9]
# We can also specify the last index as -1
# So that we do not need to know total number of List elements.
print(a[0:-1:2])     # Will print [1, 3, 5, 7, 9]
print(a[2:-2])       # Will print [3, 4, 5, 6, 7, 8]
```

Traversing/Scanning a List:

If we need to traverse a complete list, say we want the product of each element of list, we can use a `for` loop for that. For example, for a list named `myList` with five elements, we can do it like:

```
myList=[30,45.2,33,42,15.8]
p=1
for i in range(5):
    p*=myList[i]
print(f"The product of list numbers is: {p}")
```

The above method is good to scan a list of five elements. For list of some other size we will use that number in range parameter. But a much better way is to use a Python built-in function **`len()`** that gives the length i.e. size of the list that is passed as the input parameter. For example, **`len(myList)`** will return 5 in above case. Therefore, to scan a complete list, the better approach will be:

```
myList=[30,45.2,33,42,33,42,15.8]
p=1
for i in range(len(myList)):
    p*=myList[i]
print(f"The product of list numbers is: {p}")
```

List Membership Test using "in" Operator:

Like we have used the **`in`** operator with **`range()`** function, we can use it on list elements as well. It can be used to check if an element is inside a list or not. See this example:

```
myList=[30,45.2,33,42,33,42,15.8]
print(30 in myList) # Will print True
print(32 in myList) # Will print False
```

Even a better way to use **`in`** operator is directly using it on list elements. For example, like:

```
for i in myList:
```

In such case, the loop variable **`i`** is not the index but it is the list element starting from first element in the first iteration to the last element in last iteration. Hence the task of finding the product of all list numbers can be done as:

```
myList=[30,45.2,33,42,33,42,15.8]
p=1
for i in myList:
    p*=i
print(f"The product of list numbers is: {p}")
```

You can see this way is simple and we do not need to access each element through indexing, the loop variable itself is the element of the list. We will be using this format as much possible but at times we may need to access the element by index.

Tasks:

[1] Write a program that will find the average of the numbers inside a list. The list may contain any number of values. In your program, define a list with a few entries as shown in above example, then use a **`for`** loop to find the sum of numbers in that list. After the loop, use that sum to find and display the average.
Note: There is a built-in Python function named as **`sum()`** which can be applied on a list to find the sum of elements but at present do not use it and do this by finding the sum by scanning the whole list.

[2] Write a program that will replace all list entries which are greater than 10 with 10 and negative entries with 0. You have to do it as:

- **a.** Start the program by defining a list with few entries similar to above task.
- **b.** Display the complete list.
- **c.** Use a `for` loop to scan the complete list. Within `for` loop you will check if list element is greater than `10` then update that element to `10`. Likewise, the entries less than `0` should be replaced with `0`.
- **d.** Display the complete list after the entries have been updated.

Sample Output is:

```
Initial List: [5, 6, -2.3, 12, 4, 13.8, 100, 5, -2]
Final List: [5, 6, 0, 10, 4, 10, 10, 5, 0]
```

[3] Write a program that will display number of primes in a list.

Sample Output is:

```
List Elements: [5, 6, 8, 12, 4, 13, 100, 5, 21]
There are 3 prime numbers in above list
```

You will do it as:

- **a.** Start the program by defining a list with few positive integers similar to the above task.
- **b.** Scan the list using a `for` loop.
- **c.** Within `for` loop, use another `for` loop to find whether that list element is prime or not. If it is prime, increment the count variable.
- **d.** Do not define function to determine if the number is prime or not.
- **e.** Display the count.

[4] Repeat above task but now to find whether a number is prime or not, define and use a function named as `isPrime` as we did earlier. Use this function inside the list scanning `for` loop.

[5] Finding the maximum value in a list

In this program define a list with a few entries and then program must find and display the maximum value from list entries.

How to do that:

Suppose here is our initial list:

```
myList=[30,45,33,82,33,42,15]
```

The logic to find maximum out of above list goes something like:

- Start with declaring first value as the maximum value, like: `highest=myList[0]`
- Start scanning the array from second entry till the last one; if the current index value is greater than highest declared, update the highest value as:

```
if(myList[i])>highest):

    highest=myList[i]
```

[6] Update above task so that both the maximum and the minimum value is inside the list is found and displayed. You have to find both values within the same `for` loop.

List Operations:

The two list operations known as list concatenation (+ Operator) and repetition (* Operator) along with list comparison are explained in the table below:

Expression	Result
`[5,3,"Hello"]+[2,4]`	`[5,3,"Hello",2,4]`
`[1]*5`	`[1,1,1,1,1]`
`[1,5,2]*3`	`[1,5,2,1,5,2,1,5,2]`
`[2,5,3]==[2,5,3]`	`True`
`[2,5]>[2,3,7]`	`True`

Empty List:

We can also define an empty list i.e. a list with no element inside it. The use of such list will be clear soon. There are two ways to define an empty list.

```
a=[]
b=list()
```

Built-in List Methods:

There are few built-in very useful methods in list class for its processing and operations. We will explore these step by step:

Method	Description	Example	Output

`append(x)`	Appends x at the end of the list.	`a=[5,4]`
		`a.append(6)`
		`print(a)`
			`[5, 4, 6]`

`insert(i,x)`	Inserts x at i-th index	`a=[5,4,1]`
		`a.insert(1,6)`
		`print(a)`
			`[5, 6, 4, 1]`

`pop(i)`	Removes the list element at index i.	`a=[5,4,1]`
		`a.pop(1)`
		`print(a)`
			`[5, 1]`

`remove(x)`	Removes the element x from list.	`a=[5,4,1]`
		`a.remove(1)`
		`print(a)`
			`[5, 4]`

`count(x)`	Gives the number of time x I present in the list	`a=[5,4,4,1]`
		`b=a.count(4)`
		`print(b)`
			2

`index(x)`	Returns the index of x inside list. If there are multiple x in list it will return the index of first one.	`a = [3,6,10,6]`
		`print(a.index(6))`
			`1`

`clear()`	Clears all entries of a list	`a = [3,6,10,6]`
		`a.clear()`
		`print(a)`
			`[]`

`sort`	Sorts list in ascending order.	`a = [3,4,1,7,8,2]`
		`a.sort()`
		`print(a)`
			`[1, 2, 3, 4, 7, 8]`

	Can also sort list in reverse order.	`a = [3,4,1,7,8,2]`
		`a.sort(reverse=True)`
		`print(a)`
			`[8, 7, 4, 3, 2, 1]`

`reverse()`	Reverses a list	`a = [3,6,10,7,8,2]`
		`a.reverse()`
		`print(a)`
			`[2, 8, 7, 10, 6, 3]`

`extend`	Extends a list at the end of another list similar to + operator	`a = [3,6,10]`
		`b=[4,5]`
		`b.extend(a)`
		`print(b)`
			`[4, 5, 3, 6, 10]`

Built-in Python Functions for List:

Some Python built-in functions for list are given here:

Method	Description	Example	Output

`del`	Another way to delete list element via its index	`a=[5,4,1]`
		`del a[2]`
		`print(a)`
			`[5, 4]`

`len(a)`	Returns the length of the list.	`a = [3,6,10,6]`
		`b=len(a)`
		`print(b)`
			`4`

`max(a)`	Returns the maximum element of the list	`a = [3,6,10,6]`
		`print(max(a))`
			`10`

`min(a)`	Returns the minimum element of the list	`a = [3,6,10,6]`
		`print(min(a))`
			`3`

`sorted(a)`	a is the list which will get sorted in ascending order	`a = [3,4,1,7,8,2]`
		`b=sorted(a)`
		`print(b)`
			`[1, 2, 3, 4, 7, 8]`

	List a can also be sorted in reverse order.	`a = [3,4,1,7,8,2]`
		`b=sorted(a,reverse=True)`
		`print(b)`
			`[8, 7, 4, 3, 2, 1]`

`sum(a)`	All elements of list a are summed up.	`a = [3,4,1,7,8,2]`
		`b=sum(a)`
		`print(b)`
			`25`

`all(a)`	Returns True if all values inside the list are True. Only 0 and False are treated as not True, any other value is treated as True. In other words, if there is one or more False or 0 inside the list, it will return False.	`a=[2,3,5,'Python']`
		`b=[1,2,0]`
		`print(all(a))`
		`print(all(b))`
			`True`
			`False`

`any(a)`	Returns True if there is at least one True (any thing other than 0 or False) in the list. Returns False when all values inside the list are False or 0.	`a=[0,3,5,False]`
		`b=[0,False]`
		`print(any(a))`
		`print(any(b))`
			`True`
			`False`

Filling list with user's input values:

You can see the use of **append()** method as adding an element at the end of the existing list. One possible use is shown in the code below where user is allowed to enter as many numbers as he wants and enters -1 to finish the process. An empty list is defined first, and each new entry is appended to it until -1 is entered.

```
numList=[]
while True:
    x=eval(input("Enter Next number (-1 to exit):"))
    if x==-1:
        break
    numList.append(x)

print("You entered these numbers:")
print(numList)
```

Tasks:

[7] Write a program that will ask user to enter numbers and will then find and display the average, the highest and the lowest values. User can enter as many numbers as he wants and will enter **-1** to end the process. Use one **while** loop to take in the entries as shown above and then use **only** one **for** loop to calculate the three outputs.

[8] Write a program that will first create a list of 20 random numbers (1 to 50). One `for` or `while` loop will do this part. Then the program should create a new list that will contain all prime numbers in the first list. Using a function `isPrime` is up to you.

Sample Output is:

```
A random List: [5, 16, 28, 12, 44, 13, 40, 35, 21, 37, 24, 31,
27, 13, 34, 19, 41, 33, 29, 34]
List of Prime numbers: [5, 13, 37, 31, 13, 19, 41, 29]
```

[9] We will define following list of 20 elements in our program:

```
a = [7,3,6,10,6,7,-2,7,5,-8,23,12,-22,3,6,-5,7,5,10,-20]
```

And will do the following tasks on it. We have seen detail of filling a list with user's input numbers. But for this task we will not take user's data and will define the above list in program. The purpose is to understand and use the methods and functions given in last lab session.

Secondly, you are not allowed to use `for` loop or `while` loop for the below tasks. The first task is solved for your guidance.

a. **Find the index of maximum value inside the list.**
   From table of last lab session, we can see that we have `max()` function for the maximum value of list and `index()` method to find index of any element. We can use both of them as shown here:

   ```
   print(a.index(max(a)))
   ```

   This will display 10 which is the index of 23, the maximum value in the list.

b. **Find how many numbers are less than 7 in the list.**
   - Sort the list. The original list must not be overwritten i.e. the sorted list must be a new list keeping the original list as there.
   - Find the index of 7
   - Relate index of 7 in sorted list with the number of elements less than 7

c. **Find how many numbers are greater than 7 in the list.**
   First Approach:
   - Sort list in reverse order. The original list must not be overwritten i.e. the sorted list must be a new list keeping the original list as there.
   - Find the index of 7
   - Relate index of 7 in reverse sorted list with the number of elements greater than 7

   Second Approach:

   - Find number of elements less than 7 as in part b.
   - Count number of times 7 is in the list.

    - Find total number of elements i.e., length of the list.
    - Relate above three number to find the number of elements greater than 7

d. **Find the index of second largest number in the list. If there are two or more instances of the largest value, the second largest can be considered same as the first largest. (later, we will see the case where the second largest means the one after all occurrences of first largest).**
    - Sort the list. The original list must not be overwritten i.e., the sorted list must be a new list keeping the original list as there.
    - Find the second last number in the sorted list.
    - Find the index of above number from original list.

e. **Replace the minimum value (only the first instance in case there are multiple) by the maximum value.**
    - Find the minimum and the maximum value.
    - Find index of the minimum value
    - Replace that index value with maximum value.