List in Python

A list in Python is a collection of values that are stored in a single object, in a specific order. Lists allow you to store multiple items in a single variable, and access those items by referring to their indices. Lists are defined using square brackets [] and can contain elements of any data type, including other lists.

In addition to accessing individual elements, you can perform various operations on lists in Python, such as adding or removing elements, concatenating two lists, slicing, and more. Lists are mutable, which means you can change their contents by adding, removing, or modifying elements.

Overall, lists are a powerful and flexible data structure that you can use to store and manipulate collections of data in your Python programs.

Note: List allows duplicate values as well.

In [2]:
```
#creating a list, elements in list are seperated using comma
myFriendsList = ["Nasir","Hamza","GM","Chan","Jillani","Jibran","Awais","Jahantab",]
#register yourself if you're not in list :)
#lets print the first element of list
friend = myFriendsList[0] #we can access an element using its index
print(friend)
```

```
Nasir
```

In [3]:
```
#since list are mutable, mutable means we can change the element of list, lets change element at index 0
print("before changing list is:\n{}".format(myFriendsList))
#changing value of index 0
print("Changing Value")
myFriendsList[0]="M Sabir"
#print list again
print("after changing list is:\n{}".format(myFriendsList))
```

```
before changing list is:
['Nasir', 'Hamza', 'GM', 'Chan', 'Jillani', 'Jibran', 'Awais', 'Jahantab']
Changing Value
after changing list is:
['M Sabir', 'Hamza', 'GM', 'Chan', 'Jillani', 'Jibran', 'Awais', 'Jahantab']
```

In [7]:
```
# a list can contain another list as well
studentsList = [
    ["MUHAMMAD NASIR","SP20M2BB030","BSCS 7th E"],
    ["GHULAM JILLANI","SP20M2BB008","BSCS 7th E"],
    ["M AWAIS","SP20M2BB073","BSCS 7th E"],
    ["JAHANTAB SIDDIQUE","SP20M2BB079","BSCS 7th E"],
]
#a student list is given which contains 4 sub lists, and each sub-list correspond to a individual student
#information
print(studentsList[0])
print(studentsList[1])
print(studentsList[2])
print(studentsList[3])
print("\n")
print(studentsList)
```

```
['MUHAMMAD NASIR', 'SP20M2BB030', 'BSCS 7th E']
['GHULAM JILLANI', 'SP20M2BB008', 'BSCS 7th E']
['M AWAIS', 'SP20M2BB073', 'BSCS 7th E']
['JAHANTAB SIDDIQUE', 'SP20M2BB079', 'BSCS 7th E']


[['MUHAMMAD NASIR', 'SP20M2BB030', 'BSCS 7th E'], ['GHULAM JILLANI', 'SP20M2BB008', 'BSCS 7th E'], ['M AWAIS', 'SP2
0M2BB073', 'BSCS 7th E'], ['JAHANTAB SIDDIQUE', 'SP20M2BB079', 'BSCS 7th E']]
```

Slicing with list

We already seen slicing in strings and previous topics, but here I just want to mention again. Slicing in Python allows you to extract a portion of a sequence (such as a list or a tuple) and create a new sequence from it. variable[start:end] #end index will not be included in results like variable[0:4] means include 0,1,2,3 but not 4

In [92]:
```
studentsList = [
    ["MUHAMMAD NASIR","SP20M2BB030","BSCS 7th E"],
    ["GHULAM JILLANI","SP20M2BB008","BSCS 7th E"],
    ["M AWAIS","SP20M2BB073","BSCS 7th E"],
    ["JAHANTAB SIDDIQUE","SP20M2BB079","BSCS 7th E"],
]
studentList2 = studentsList[0:3] #it will return students at index 0, 1 and 2
print("Length of newly generated list using studentsList[0:3] is = {}".format(len(studentList2)))
print("")
print(studentList2)
```

```
Length of newly generated list using studentsList[0:3] is = 3

[['MUHAMMAD NASIR', 'SP20M2BB030', 'BSCS 7th E'], ['GHULAM JILLANI', 'SP20M2BB008', 'BSCS 7th E'], ['M AWAIS', 'SP2
0M2BB073', 'BSCS 7th E']]
```

LIST BASIC OPERATIONS

append() and extend() methods

```
append():
this method will append the given element (can by of any type) at the end of list
```

In [10]:
```
numList = [1,3,5,7]
numList.append(99)
print(numList)
```

```
[1, 3, 5, 7, 99]
```

```
extend():
This method extends the list by appending(adding) all the elements from the given iterable.
```

In [11]:
```
#lets create another list
numList2 = [111,333,555,777,999]
#lets extend numList with numList2
numList.extend(numList2)
#lets print the extended list numList
print(numList)
```

```
[1, 3, 5, 7, 99, 111, 333, 555, 777, 999]
```

```
as shown above all the element of numList2 are added in numList.
we can also add numList2 using append but with that the list object will be appended in
numList not the individual element as shown above output. let see the difference of append and extend
```

In [14]:
```
numList = [1,3,5,7]
numList2 = [111,333,555,777,999]
numListAppended = numList.append(numList2)
numListExtended = numList.extend(numList2)
print("Appended List Output: {}".format(numListAppended))
print("Extended List Output: {}".format(numListExtended))
```

```
Appended List Output: None
Extended List Output: None
```

```
Both variables contains None (None is a special data type which is assigned to a variable when method
returns nothin) so in our case both methods are returning null because they are of void type this is the reason
both variables contains None.
These methods basicaaly modifies the list object using which they are accessed.
we can use a copy() method which will return a copy of list and we can store it in new list object
```

In [74]:
```
numList = [1,3,5,7]
numList2 = [111,333,555,777,999]
numList.append(numList2)
numListAppended = numList.copy()#we are copying the modified numList in numListAppended
#the copy method will return a new list object instead of same
#list object, so changes made in numList will not be reflect in numListAppended
#new object to numList which is different from the one it holding before
numList = [1,3,5,7]
numList.extend(numList2)
numListExtended = numList.copy()#we are copying the modified numList in numListExtended
print("Appended List Output: {}".format(numListAppended))
print("Extended List Output: {}".format(numListExtended))
```

```
Appended List Output: [1, 3, 5, 7, [111, 333, 555, 777, 999]]
Extended List Output: [1, 3, 5, 7, 111, 333, 555, 777, 999]
```

```
as shown above the appended method will append the entire list object as a single element in list, while
the extend method take each eleemnt from the provided list and put each element in list as individual item
```

In [83]:
```
list1 = [1,2,3]
list2 = [4,5,6]
print("List1:{}".format(list1))
print("List2:{}".format(list2))
list3 = list1 + list2
print("New list, list3 generated by adding two lists as list1 + list2:\n{}".format(list3))
#the + operator will perform same action as the extend method but there is a difference, the
#extend method is of type void and returns null and also it modified the original list as
#list1.extend(list2) will modify the first list and add all eleemnts of list 2 into list2 but
#the + operator on the other hand does not modify the original list instead it keep the lists as they are and create
#a new list object which contains all the object of list1 + list2. let se an example
list1 = [1,2,3]
list2 = [4,5,6]
list1.extend(list2)
print("\nOld list object of list1, which is modified using list1.extend(list2):\n{}".format(list1))
```

```
List1:[1, 2, 3]
List2:[4, 5, 6]
New list, list3 generated by adding two lists as list1 + list2:
[1, 2, 3, 4, 5, 6]

Old list object of list1, which is modified using list1.extend(list2):
[1, 2, 3, 4, 5, 6]
```

Inserting element at specified index

```
we can use insert method to insert given element at specific given index, let see an example to understand,
```

In [19]:
```
numList = [1,3,5,7]
print("length of list:{}".format(len(numList)))
```

```
numList.insert(2,99) #insert element 2 at index 2, keep in mind one thing that inesrt function will not override
#the value of index 2 which now is 5 to 99, instead what it will do is that it will move the element one index
#further from index 2 and add given value which is 99 at given index which is 2 and the element 5 will move on
#to index 3. let's print the result
print(numList)
print("length of list:{}".format(len(numList)))
```

```
length of list:4
[1, 3, 99, 5, 7]
length of list:5
```

As shown above before inserting the element the length of list is 4 and after inserting element 99 at index 2 the length of list increases to 5.

Removing element from list using remove() and pop()

To remove a specific element from list simply pass that element to remove() method as,

In [22]:
```
numList = [1,3,5,7]
numList.remove(1) #here 1 is element not index, so the element 1 which is on index 0 will be removed from list
print(numList)
```

```
[3, 5, 7]
```

Note: If given element not exist in list the remove() method will generate error that element not found in list
to remove a element using its index use pop() method as,

In [58]:
```
numList = [1,3,5,7]
numList.pop(1) #remove element at index 1 which is 3, the index passed in pop will always be integer
print(numList)
```

```
[1, 5, 7]
```

In [59]:
```
#del keyword can be used to delete the element at specified index, syntax is del list[indexOfElement]
del numList[2] #it will remove the element at index 2 which is 7
print(numList)
```

```
[1, 5]
```

In [60]:
```
#clear() method is used to clear all the element of list
print("list before clear:{}".format(numList))
print("Clearing the list")
numList.clear()
print("list after clear:{}".format(numList))
```

```
list before clear:[1, 5]
Clearing the list
list after clear:[]
```

Finding index of an element

In [85]:
```
#to find a index of specific element pass that element to the method named list.index() as,
numList = ["Nasir","Sabir","Hamza","Chan"]
indexOfHamza = numList.index("Hamza") #it will return the index of element "Hamza"
print(indexOfHamza)
```

```
2
```

The index of element "Hamza" is 2

Counting a number of occurence of specific element in list

To count the number of occurence of specific element in list we can use count() method. Example

In [87]:
```
awardList = ["SP20M2BB008","SP20M2BB033","SP20M2BB032","SP20M2BB008","SP20M2BB030","SP20M2BB030"]
#lets count the num of occurence of "SP20M2BB030"
count = awardList.count("SP20M2BB030")
print(count)
```

```
2
```

Sorting a list

To sort a list in ascending order we can use a method named sort() of list, which will sort the list items automatically. It a declarative programming we just ask to sort how we don't know

In [33]:
```
awardList = ["SP20M2BB008","SP20M2BB033","SP20M2BB032","SP20M2BB008","SP20M2BB030","SP20M2BB030"]
awardList.sort() #lets sort the list
print(awardList)#print the sorted list
```

```
['SP20M2BB008', 'SP20M2BB008', 'SP20M2BB030', 'SP20M2BB030', 'SP20M2BB032', 'SP20M2BB033']
```

In [34]:
```
#lets reverse the order of list using reverse() method
awardList.reverse() #reversing the items of list
print(awardList) #print the items of list
```

```
['SP20M2BB033', 'SP20M2BB032', 'SP20M2BB030', 'SP20M2BB030', 'SP20M2BB008', 'SP20M2BB008']
```

Max, Min and Sum of List

```
max() method is used to find the maximum element of list
min() is used to find the minimum element of list
sum() is used to calculate the sum / total of elements of list
These are global function which takes list as argument,
Function is a global it don't belongs to any specific class and can be accessed anywhere without any object
Method belongs to specific class and can only be accessible using the object of that class
```

In [36]:
```
print(max(awardList))
```

```
SP20M2BB033
```

In [38]:
```
listOfNumber = [89434,56,546,23,2,1,4,54,6,757,45]
print("max:{}".format(max(listOfNumber)))
print("min:{}".format(min(listOfNumber)))
print("sum:{}".format(sum(listOfNumber))) #sum function will only work with integer and float numbers, if we use
#sum with other data types we get error, and the initial value for sum = 0, it add all the values in sum and return
#the sum value, we can also provide the initial value as  sum(list,10), now it will start adding values in 10
#instead of 0
print("unsorted list: {}".format(listOfNumber))
listOfNumber.sort()
print("sorted list: {}".format(listOfNumber))
```

```
max:89434
min:1
sum:90928
unsorted list: [89434, 56, 546, 23, 2, 1, 4, 54, 6, 757, 45]
sorted list: [1, 2, 4, 6, 23, 45, 54, 56, 546, 757, 89434]
```

How string are sorted in list

```
If a list contains a strings elements still we can sort them in ascending or descending order. the strings
are sorted according to the charcters they contains, let suppose we have two strings as,
"Nasir" and "Habibi" now the first letter of both strings will be matches.
The string with the smaller character in the first non-matching(a position where character of
both strings are not same) position comes first.
```

In [63]:
```
listOfString = ["Nasir", "Habibi"]
listOfString.sort()
print(listOfString)
```

```
['Habibi', 'Nasir']
```

```
Now what will happen if the first character of both string is same, already defined in above description that
first-non-matching character will be compared which means if there are two strings whose first letter are even
more than one letter is same, then the letter no matter at which position they are will be matched in string and
according to their value the string will be sorted for example,
```

In [64]:
```
listOfString = ["Muneeb", "Mujeeb"]
listOfString.sort()
print(listOfString)
```

```
['Mujeeb', 'Muneeb']
```

```
Now in above example the first two letter are same as Mu and Mu so the third letter which is n and j will be compare
and dependig on their evaluation the string will be sorted.
Now what if list contains two strings with all characters same as,
"Nasir", "Nasir"
in that case these strings will be compared with all other strings and any strings which contains less value
 character will be sorted before them and they will be placed in there own and they any string with value greater
than then will be sorted after them for example,
 ["Ahmad", "Nasir", "Amjad","Basit","Saqib","Zaigham","Nasir",]
 from above list all the letter which comes before  as A,B will be sorted before Nasir and S and Z containing
 strings will be sorted later as shown below,
```

In [65]:
```
nameList =["Ahmad", "Nasir", "Amjad","Basit","Saqib","Zaigham","Nasir",]
nameList.sort()
print(nameList)
```

```
['Ahmad', 'Amjad', 'Basit', 'Nasir', 'Nasir', 'Saqib', 'Zaigham']
```

In [68]:
```
#to sort in reverse use
nameList.sort(reverse=True) #set argument reverse to True which by default is false
print(nameList)
```

```
['Zaigham', 'Saqib', 'Nasir', 'Nasir', 'Basit', 'Amjad', 'Ahmad']
```

In [70]:
```
nameList2 = list(nameList)
nameList3 = nameList.copy()
nameList[0]="OK"
print(nameList)
print(nameList2)
print(nameList3)
print(nameList2 is nameList)
print(nameList3 is nameList)
```

```
['OK', 'Saqib', 'Nasir', 'Nasir', 'Basit', 'Amjad', 'Ahmad']
['Zaigham', 'Saqib', 'Nasir', 'Nasir', 'Basit', 'Amjad', 'Ahmad']
['Zaigham', 'Saqib', 'Nasir', 'Nasir', 'Basit', 'Amjad', 'Ahmad']
False
False
```

In [ ]:
```
```