Python Collection

In [ ]:
```
Python has 4 built-in data structures that can be used to hold a collection of objects, they are :
1. list
2. tuple
3. set
4. dictionary
```

List

Used to store multiple items(collection of data) in a single variable.Lists are created using square brackets.

Create a List

In [3]:
```
my_list = [1,2,3,4,5]
print("My List :",my_list)
```

```
My List : [1, 2, 3, 4, 5]
```

Characteristics of List

In [ ]:
```
List items are:
1. Ordered                : The items have a defined order, and that order will not change.
                            If you add new items to a list,the new items will be placed at the end of the list.
2. Changeable (Mutable)   : We can change, add, and remove items in a list after it has been created.
3. Allow duplicate values : List can have any number of duplicate values.
4. indexed                : The first item has index 0, the second item has index 1 etc.
```

List Methods

In [ ]:
```
Given below are the predefined methods of a list:

Method              Description

append()               Adds an element at the end of the list
clear()                Removes all the elements from the list
copy()                 Returns a copy of the list
```

```
count()                 Returns the number of elements with the specified value
extend()                Add the elements of a list (or any iterable), to the end of the current list
index()                 Returns the index of the first element with the specified value
insert()                Adds an element at the specified position
pop()                   Removes the element at the specified position
remove()                Removes the item with the specified value
reverse()               Reverses the order of the list
sort()                  Sorts the list
```

In [4]:
```
#Checking whether list allows duplicate values.
my_list = [12,23,12,55,70,23,12]
print(my_list)
```

```
[12, 23, 12, 55, 70, 23, 12]
```

Length of List

In [ ]:
```
len() : Function used to find the length of list. Length of a list is equal to the number of elements in that list.
```

In [5]:
```
#Length of a list
my_list = [1,2,3,4,5,6,7]
l = len(my_list)
print(l)
```

```
7
```

Datatypes

In [ ]:
```
A list can contain different data types.
```

In [6]:
```
#List allows different datatypes.
my_list = [12,"Smith",5.5,"Hello"]
print(my_list)
```

```
[12, 'Smith', 5.5, 'Hello']
```

In [ ]:
```
type() :Function to identify the type of a list.
```

In [7]:
```
#To identify the datatype of given variable.
my_list = [1,2,3,4,5,6,7]
print(type(my_list)) #here , my_list is a object of class list.
```

```
<class 'list'>
```

list() Constructor

In [ ]:
```
A new list can be creat using list() constructor.
function with same name as the class is called a constructor.
here, list() method is a constructor of class list.
```

In [8]:
```
#Create list using list() constructor
my_list = list((1,2,3,4,5))
print(my_list)
print(type(my_list))
```

```
[1, 2, 3, 4, 5]
<class 'list'>
```

Access List Items

In [ ]:
```
List items are indexed and you can access them by referring to the index number.
There are two types of indexing:
    1. Positive indexing
           List index starts at 0. index 0 points to the first item in the list.
           1 refers to the second item in the list an so on.
    2. Negative indexing
           Negative indexing means start from the end.-1 refers to the last item,
           -2 refers to the second last item etc.
```

In [11]:
```
#Access items of a list using Positive index.
my_list = [1,2,3,4,5,6]
print(my_list[0]) #Accessing first item
print(my_list[1]) #Accessing second item
```

```
1
2
```

In [12]:
```
#Access items of a list using Negative index.
my_list = [1,2,3,4,5,6]
print(my_list[-1]) #Accessing last item
print(my_list[-2]) #Accessing second last item
```

```
6
5
```

Range of Positive Indexes

In [ ]:
```
We can specify a range of indexes by specifying where to start and where to end the range.
eg: list1(start:end)
here, start:end is the range of the new list items.start indicates the start index and
end indicates the end index of new list.
When specifying a range, the return value will be a new list with the specified items.
```

In [13]:
```
#new list with the given range using positive indexes
my_list = [1,2,3,4,5,6]
new_list = my_list[1:4] #The search will start at index 1 (included) and end at index 4 (not included).
print (new_list)
```

```
[2, 3, 4]
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list.

In [16]:
```
#new list with the given range using negative indexes
my_list = [1,2,3,4,5,6]
new_list = my_list[-5:-3] # The search will start at index -5 (included) and end at index -3 (not included).
print(new_list)
```

```
[2, 3]
```

Check if Item Exists

In [ ]:
```
in : A keyword used to check whether a specified item is present in the list or not.
```

In [21]:
```
#Check if item (string data) is present or not
my_list = ["apple","banana","grapes","orange"]
if "apple" in my_list:
    print ("yes , item present in the list")
else:
    print("No,item not in the list")
```

```
yes , item present in the list
```

In [22]:
```
#Check if item (integer data) is present or not
my_list = [1,2,3,4,5]
```

```
if 1 in my_list:
    print ("yes , item present in the list")
else:
    print("No,item not in the list")
```

```
yes , item present in the list
```

Change Item Value

To change the value of a specific item, refer to the index number.

In [2]:
```
#ex: To change the item at index 4 in a list.
my_list = [1,2,3,4,5,6]
my_list[3] ="four"
print(my_list)
```

```
[1, 2, 3, 'four', 5, 6]
```

Change a Range of Item Values

To change the range of items in a given list,define a list with the new values, and refer to the range of index numbers where you want to insert the new values. If you insert more items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly. The length of the list will change when the number of items inserted does not match the number of items replaced.

In [3]:
```
#ex: To change the range of items in a list.
my_list = [1,2,3,4,5,6]
my_list[1:4] =["two","three","four"] #changed the items at index 1,2,and 3
print(my_list)
```

```
[1, 'two', 'three', 'four', 5, 6]
```

In [4]:
```
#ex: To change the range of items in a list.(When you insert more items than the no. of items want to be replaced)
my_list = [1,2,3,4,5,6]
my_list[1:4] =["two","three","four","five","six"] #add 5 items instead of 3.
print(my_list)
```

```
[1, 'two', 'three', 'four', 'five', 'six', 5, 6]
```

Insert Items

In [ ]:
```
insert()  :Function to insert a new item to the list without replacing any existing values. The insert() method can
           insert an item at the specified index
```

In [9]:
```
# insert a new item to the list.
my_list = [10,20,40,50,60]
my_list.insert(2,30) #insert item 30 at index 2
print(my_list)
```

```
[10, 20, 30, 40, 50, 60]
```

Append Items

append() : method to add an item to the end of the list.

In [ ]:
```

```

In [10]:
```
# append an item to the list.
my_list = [10,20,40,50,60]
my_list.append(70) # add item 70 at the end of the list
print(my_list)
```

```
[10, 20, 40, 50, 60, 70]
```

Extend List

extend() : 1.Method to append elements from another list to the current list.The elements will be added to the end of the list.

2. The extend() method can append a list,tuple, set or dictinaries at the end of the current list.

In [11]:
```
# extend a list
my_list1 = [1,2,3,4]
my_list2 = [5,6,7,8]
my_list1.extend(my_list2)
print(my_list1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [15]:
```
#The extend() method can append a list,tuple, set or dictinaries at the end of the current list.
my_list1 = [1,2,3,4]
my_list2 = (5,6,7,8) #tuple
my_list3 = {5,6,7,8} #set
```

```
#my_list1.extend(my_list2) # appending a tuple at the end of a list
my_list1.extend(my_list3) # appending a set at the end of a list (order may varry since set is unordered)
print(my_list1)
```

```
[1, 2, 3, 4, 8, 5, 6, 7]
```

Remove List Items

remove() : Method to remove an item from a list. pop() : Method to remove an iten at specified index. If you do not specify the index, the pop() method removes the last item. del : Keyword to remove an item at specified index

In [18]:
```
#remove an item from the list
my_list = [1,2,3,4]
my_list.remove(3) #item 3 will be removed
print(my_list)
```

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

In [20]:
```
#remove an item at specified index from the list
my_list = [1,2,3,4]
my_list.pop(2) #item at index 2 will be removed
print(my_list)
```

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

In [21]:
```
#remove an last item from the list
my_list = [1,2,3,4]
my_list.pop() # pop method with out any argument remove the last element of the list, here 4 will be removed.
print(my_list)
```

```
[1, 2, 3]
```

In [22]:
```
#remove an item at specified index from the list
my_list = [1,2,3,4]
del my_list[2] #item at index 2 will be removed
print(my_list)
```

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

Clear the List

In [ ]:
```
clear() :Method to empties the list.
The list still remains, but it has no content.
```

In [23]:
```
#To empty the list
my_list = [1,2,3,4]
my_list.clear()
print(my_list) #empty the list.
```

```
[]
```

Loop Through a List

We can use for loop to iterate through list elements.

In [8]:
```
#for loop to iterate through items in a list
my_list = ["Hai","welcome","to","python","list"]
for i in my_list:
    print(i)
```

```
Hai
welcome
to
python
list
```

Loop Through the Index Numbers

We can use range() and len() methods to iterate through index numbers of a list.

In [14]:
```
#for loop to iterate through index numbers of a list
my_list = ["Hai","welcome","to","python","list"]
#l = len(my_list)
#print(l) #retun no.of items in the list(here,5)
for i in range(len(my_list)):
    print(i)
```

```
0
1
2
3
4
```

In [23]:
```
#range function
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Using a While Loop

We can iterate list using for loop

In [27]:
```
#while loop to iterate through items in a list
my_list = ["Hai","welcome","to","python","list"]
i=0
while i<len(my_list):
    print(my_list[i])
    i = i+1
```

```
Hai
welcome
to
python
list
```

In [28]:
```
#while loop to iterate through all the index numbers in a list
my_list = ["Hai","welcome","to","python","list"]
i=0
while i<len(my_list):
    print(i)
    i = i+1
```

```
0
1
2
3
4
```

Looping Using List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

In [29]:
```
#Short hand of for loop :
my_list = ["Hai","welcome","to","python","list"]
```

```
[print(i) for i in my_list]
```

```
Hai
welcome
to
python
list
[None, None, None, None, None]
```

Out[29]:

In [32]:
```
#Short hand of for loop :
my_list = ["Hai","welcome","to","python","list"]
[print(i) for i in my_list if 'o' in i] # print all the items in list that has 'o' in it.
```

```
welcome
to
python
[None, None, None]
```

Out[32]:

Sort Lists

sort() :1. Method to sort the list elements alphanumerically, ascending by default.

2. By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters

In [35]:
```
#sort list elements
my_list = [12,2,44,23,9,15]
my_list.sort()
print(my_list)
```

```
[2, 9, 12, 15, 23, 44]
```

In [39]:
```
#sort list elements
my_list = ["Welcome","TO","pyhton","TUTORIAL"] #uppercase sorted first since sort() method is case sensitive
my_list.sort()
print(my_list)
```

```
['TO', 'TUTORIAL', 'Welcome', 'pyhton']
```

Case Insensitive Sort

If you want to sort list elemets irrespective of case(upper case or lower case) , use the keyword argument ' key = str.lower '.

In [40]:
```
#sort list elements
my_list = ["Welcome","TO","pyhton","TUTORIAL"] #uppercase sorted first since sort() method is case sensitive
my_list.sort(key = str.lower) #case insensitive sorting
print(my_list)
```

```
['pyhton', 'TO', 'TUTORIAL', 'Welcome']
```

Sort Descending

To sort descending, use the keyword argument : reverse = True

In [38]:
```
#sort list elements descending
my_list = [12,2,44,23,9,15]
my_list.sort(reverse = True)
print(my_list)
```

```
[44, 23, 15, 12, 9, 2]
```

Reverse Order

reverse() : Method to print reverse order of a list.

In [41]:
```
#Print list items in reverse order
my_list = ["Welcome","TO","pyhton","TUTORIAL"]
my_list.reverse()
print(my_list)
```

```
['TUTORIAL', 'pyhton', 'TO', 'Welcome']
```

Copy Lists

In [ ]:
```
There are two methods to make a copy of a list:
    1. copy() : built-in List mehtod to make a copy of a list.
    2. list() : built-in constructor to make a copy of a list.
```

In [44]:
```
#make a copy of a list using copy()
my_list = ["Welcome","TO","pyhton","TUTORIAL"]
my_list_copy = my_list.copy()
print(my_list_copy)
```

```
['Welcome', 'TO', 'pyhton', 'TUTORIAL']
```

In [46]:
```
#make a copy of a list using list()
my_list = ["Welcome","TO","pyhton","TUTORIAL"]
my_list_copy =list(my_list)
print(my_list_copy)
```

```
['Welcome', 'TO', 'pyhton', 'TUTORIAL']
```

Join Lists

In [ ]:
```
There are several methods to join two lists:
    1. Using '+' operator
    2. Using append() method
    3. Using extend() method
```

join list using '+' operator

In [48]:
```
#join two listsusing '+' operator
list1 = [1,2,3,4]
list2 = [5,6,7,8]
list3 = list1 + list2 #concatenation
print(list3)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

join list using append() method

In [51]:
```
### join list using append()
list1 = [1,2,3,4]
list2 = [5,6,7,8]
for i in list2:
    list1.append(i)
print(list1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

join list using extend() method

In [53]:
```
### join list using extend()
list1 = [1,2,3,4]
```

```
list2 = [5,6,7,8]
list1.extend(list2)
print(list1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

End