Python List Manipulation

List

A collection of comma separated values which are surrounded by square brackets [ ] are known as List in Python. The sequence of values may include any type of values like int, float, string, list itself and other sequences. List in Python is ordered collection of heterogeneous values.

Note: Lists are Mutable (Changeable)

L=['hello' , 333001 , 'abc.kvs@gov.in']

You can display a List literal with the print() function:

```
print(L)
Output:  ['hello' , 333001 , 'abc.kvs@gov.in']
```

Creating List

A List can be create as similar to other variable creation in Python. For creation of List we just assign a List value (which enclosed between square brackets) to a variable name followed by an equal sign.

```
L=[20]                          Lst=[10,20,30,40,50]

L=[10,20.5,30.8,40.4,50,60]
      Lst=["Apple","Mango","Orange"]

L=['A','E','I','O','U']
      Lst=["Apple",100,"Banana",30]

L=[10,[2,4,6],"KVS",5.8,'A']
```

Creating an Empty List

1. An empty List can be created by assigning [] to a variable.

   Lst=[ ]

   Here Lst is a List type Variable.

   Print(Lst) Output: [ ]

2. An empty List can be created by list() constructor also.

   Lst=list()

   Print(Lst) Output: [ ]

Creating a List from Existing Sequence
Creating a List using list() Constructor

A list can be created from other sequence or by using list() constructor.

Example:

Lst=list("KVS")

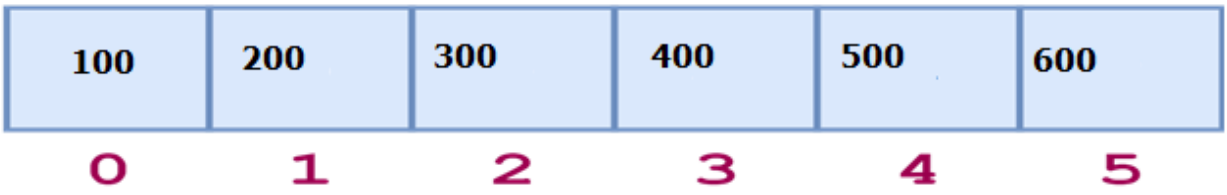
Print(Lst) Output: ['K', 'V', 'S']

Indexing a List

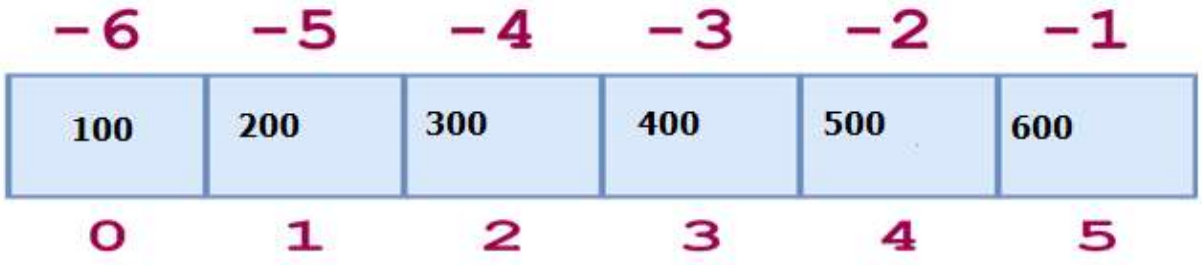
In Python, Lists are ordered sequences of values, and thus can be indexed in this way. One value in a List can be accessed by specifying the List name followed by an index number in square brackets ([ ]).

List indexing in Python is zero-based: the first value in the List has index 0, the next has index 1, and so on. The index of the last value will be the length of the List minus one.

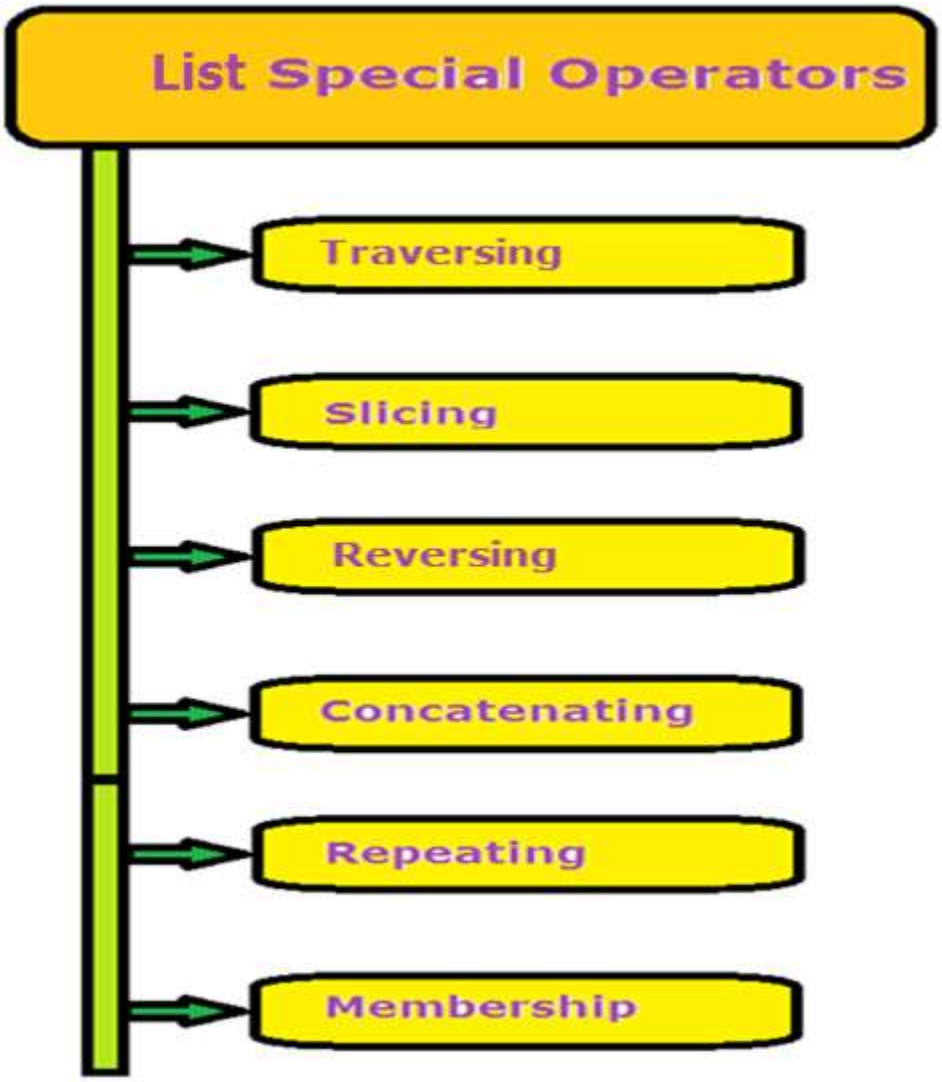
For example, a schematic diagram of the indices of the List [100,200,300,400,500,600] would look like this:



List indices can also be specified with negative numbers, in which case indexing occurs from the end of the List backward: -1 refers to the last value, -2 the second-to-last value, and so on. Here is the same diagram showing both the positive and negative indices into the List [100,200,300,400,500,600]:



Special List Operators



Traversing a List

Square brackets can be used to access elements of the List.

Get the element at position 1 (remember that the first element has the position 0):

```
Lst=[100,200,300,400,500,600]
Print( Lst[3] )                    # Output:  400
Print( Lst[-2] )                   # Output: 500
```

Display All List elements using loop

```
Lst=[100,200,300,400,500,600]
for val in Lst:
        print(val, end=",")
# Output:   100,200,300,400,500,600
```

Display All List elements using range()

```
Size=len(Lst)
for index in range(Size):
      print(Lst[index], end=" ")
# Output:   100,200,300,400,500,600
```

List Slicing

Python also allows a form of indexing syntax that extracts sublist from a List, known as List slicing.

Format of List slicing:

List_Variable [Start Index: End Index: Step Value]

Where start, end index and step value separated with colon (:)

Start index is by default Zero (0)

End Index is by default end of List.

Step Value is by default 1

Note:

1. Start index always include in result
2. End index (if given) always exclude in result
3. End index (if not given) then last index of List include

Get the elements from position 2 to position 5 (not included):

Lst=[100,200,300,400,500,600] print(Lst[2:5])	Output: [300, 400, 500]

Use negative indexes to start the slice from the end of the List:

Get the elemetns from position 5 to position 1, starting the count from the end of the List:

Lst=[100,200,300,400,500,600] print(Lst[-5:-2])	[200, 300, 400]
print(Lst[2:8:2])	[300, 500]
print(Lst[2:8:-1])	[]
print(Lst[::2])	[100, 300, 500]
print(Lst[5:0:-1])	[600, 500, 400, 300, 200]
print(Lst[10:0:-2])	[600, 400, 200]
print(Lst[::-1])	[600, 500, 400, 300, 200, 100]
print(Lst[::-2])	[600, 400, 200]

Reversing List

A List in Python can be reversed by using following ways.

1. By using Traversing (Negative index Traversing)
2. By using loop

```
Lst=[100,”KVS”,20.5,’A’,500]
print(Lst[-1::-1])
```

Output: [500, 'A', 20.5, 'KVS', 100]

By Using Loop:

```
Lst=[100,”KVS”,20.5,’A’,500]
size=len(Lst)
size=Size-1
While (size>=0):
    print(Lst[size], end=””)
    size=size-1
```

Output: [500, 'A', 20.5, 'KVS', 100]

Concatenating List

Concatenating List mean joining two or more Lists with the help of + operator and make a new List.

Example:

```
L1=[100,200]
L2=["Jaipur","Ajmer"]
L3=L1+L2      print(L3)
```

Output: [100, 200, 'Jaipur', 'Ajmer']

```
L3=L1+[300]
Print(L3)                    # Output: [100, 200,300]
L3=["Ajmer","Jaipur","Bikaner"]+[100,200,300,400]
Print(L3)
```

Output: ['Ajmer', 'Jaipur', 'Bikaner', 100, 200, 300, 400]

```
L3=L1+300       Print(L3)
```

Output: It will give error as

TypeError: can only concatenate list (not "int") to list

Replicating List

The replicating operator * is used to repeating multiple times of a List to create multiple copies of given List elements.

Example:

```
L1=[100,200]
Print(L1*3)
```

Output: [100, 200, 100, 200, 100, 200]

```
Lst = 2 * ["Jaipur","Ajmer"]
Print(Lst)
```

Output: ['Jaipur', 'Ajmer', 'Jaipur', 'Ajmer']

Membership Operator in List

Membership operator is Boolean type operator. It used to check whether a value is present in main List or not. There are two membership operators used in List.

1. in operator: If value is present in main List then it will return True otherwise it will return False result.
2. not in operator: If value is not present in main List then it will return True otherwise it will return False result.

Example:

```
L1=[100, 200, 100, 200, 100, 200]
B=200 in L1
Print(B)                    # Output: True
Print( 300 in L1)           # Output: False
Print([100,200] in L1)      # Output: False
Print(300 not in L1)        # Output: True
Print(100 not in L1)        # Output: False
```

Relational / Comparison Operator

The relational operators of Python can also apply on Lists also. The relational operators are also the Boolean operators and produce either True or False result.

Python compares the Lists using their values of first value of each List, if they are equal then next elements of List compare and so on, until it finds differ Values.

If the list have string type values then they will compare using the ASCII values of their characters.

Example:

L1=[100,200,300]

L2=[100,200,300]

Print(L1 ==L2) Output: True

Print(L1 < L2) Output: False

Print(L1 >=L2) Output: True

Print(L1 > L2) Output: False

Print(L1 != L2) Output: False

Lists are Mutable

Lists are Mutable in nature in Python. Its mean the content of List can be changed once it is created. In other words, the assignment operator works to change the content of List.

Example:

Lst=[100,200,300]

Lst[1]=”Jaipur”

Print(Lst)

Output: [100,’Jaipur’,300]

Lst[3]=”Ajmer”

Print(Lst)

Output: IndexError: list assignment index out of range

(New index with element can be added by using various list functions)

Lists Built in Functions

Note: All List methods change the original List because Lists are Mutable.

1.append()

The append() function adds as a single value at the end of the list. It does not create a new list rather it modifies the original list.

Syntax:

List_Variable.append(Element)

Example:

```
Lst=["Jaipur",14,"Jhhunjhunu",18]
Lst.append("Alwar")
Print(Lst)
```

Output: ['Jaipur', 14, 'Jhhunjhunu', 18, 'Alwar']

```
Lst.append(["Ajmer",28])
Print(Lst)
```

Output: ['Jaipur', 14, 'Jhhunjhunu', 18, 'Alwar', ['Ajmer', 28]]

Problem: Add 5 elements in list at runtime.

```
Lst=list()
For n in range(5):
    Elem=input("Enter Element: ")
    Lst.append(Elem)
Print(Lst)
```

2. extend()

The extend() function adds one list at the end of list. That mean all the elements of sequence (like String/ List/ tuple/ dictionary)are added at the end of already created list.

In other words, extend() not accept the single value like int, float etc.

Syntax:

List_variable.extend(Sequence)

Example:

```
Lst=[100,"KVS",10.5]
Lst.extend("JJN")      # here “JJN” is String Sequence
Print(Lst)
        Output: [100, 'KVS', 10.5, 'J', 'J', 'N']
Lst.extend([2,4])  # here [2,4] is List Sequence
Print(Lst)
        Output: [100, 'KVS', 10.5, 'J', 'J', 'N', 2, 4]
Lst.extend((5,7))  # here (5,7) is tuple Sequence
print(Lst)
Output: [100, 'KVS', 10.5, 'J', 'J', 'N', 2, 4, 5, 7]
Lst.extend({1:”Apple”,2:”Orange”})
# here {1:”Apple”,2:”Orange”} is dictionary Sequence
print(Lst)
Output: [100, 'KVS', 10.5, 'J', 'J', 'N', 2, 4, 5, 7, 1, 2]
```

Note: List will add only Keys of dictionary in list.

```
L1=[10,20]
L2=[30,40,50]
L1.extend(L2)
Print(L1)
```

Output: [10, 20, 30, 40, 50]

```
Print(L2)
```

Output: [30, 40, 50]

```
Lst=[10,20]
Lst.extend(30)              # Here 30 is not a sequence
```

TypeError: 'int' object is not iterable

```
Lst.extend(10.5)       # Here 10.5 is not a sequence
```

TypeError: 'float' object is not iterable

3.insert()

The insert() function of list used to add new element at specified index. This function requires two parameters,

First parameter:

Index number where an element is to be inserted.

Second Parameter:

Element/Value that will insert at given index.

Syntax:

List_variable.insert(index_number, element)

Example:

```
Lst=[10,30]
Lst.insert(1,20)
print(Lst)
    Output: [10, 20, 30]
Lst.insert(2,"Ajmer")
print(Lst)
    Output: [10, 20, 'Ajmer', 30]
Lst.insert(1,[100,200])
print(Lst)
    Output: [10, [100, 200], 20, 'Ajmer', 30]
```

4.index()

The index() used to return the index number of given element. The provided element should be exist in the list otherwise error will be generated.

Syntax:

```
List_Variable.index(Element)
```

Example:

```
Lst=[10, 20, ”Ajmer”, [100, 200], 30.6]
Lst.index(“Ajmer”)
    Output: 2
Lst.index([100, 200])
    Output: 3
Lst.index(1000)
```

Output: ValueError: 1000 is not in list

Lst.index(Lst[3][1])

Output: ValueError: 200 is not in list

5.reverse()

This function reverse the order of elements in a list. It replaces the existing values of list and put new values in place of existing value.

Syntax:

List_variable.reverse()

Example:

```
Lst=[10,"KVS",20.5,30]
Lst.reverse()
Print(Lst)
```

Output: [30, 20.5, 'KVS', 10]

6.len()

len() used to find the number of elements in list.

Syntax:

len(List_variable)

Example:

```
Lst=[10,20.5,"KVS",[100,200,300],(1,2)]
Print( len(Lst) )
```

Output: 5

7.sort()

The sort () sort the list elements in either ascending or descending order. The by default order is ascending order. This function requires one optional parameter reverse=True or reverse=False.

reverse=True : Descending order

reverse=False: Ascending order

Syntax:

List_variable.sort(reverse=True/ False)

Example:

Lst= [10,40,20,5,13,20.5]

Lst.sort() OR Lst.sort(reverse=True)

Print(Lst)

Output: [5, 10, 13, 20, 20.5, 40]

Lst.sort(reverse=True)

Print(Lst)

Output: [40, 20.5, 20, 13, 10, 5]

8.count()

The count() used to return the count of element repeated in list. That mean it finds that how many times an element occurs in list. This function requires single parameter as element that to be counted in list.

Syntax:

List_variable.count(Element)

Example:

Lst= [20,40,20,20,10,20]

Lst.count(20)

Output: 4

Lst.count(200)

Output: 0

Deletion Operation on List

The deletion operation performed by using following ways.

- If index is known- pop(index)
- Remove last element of list- pop()
- If element is known- remove(element)
- Remove All elements of List- clear()
- Remove more than one but not all element- del keyword
- Remove list variable- del keyword

1. clear()

clear() remove all elements of list and make the list an empty list. This function not take any list of parameter.

Syntax:

List_variable.clear()

Example:

Lst=[1,3,5,7,8]

Lst.clear()

Print(Lst)

Output: []

2. pop()

The pop() function used to remove an element from list whose index is provided as parameter.

If index is not provided than pop() will remove last element of list.

Note: pop() will show the deleted element.

Syntax:

```
List_variable.pop()  OR
List_variable.pop(index)
```

Example:

```
Lst=[10,30,50,20,80]
Lst.pop()
Print(Lst)          # Output: [10,30,50,20]
Lst.pop(1)
Print(Lst)          # Output: [10,50,20]
Lst.pop(5)          # Error
IndexError: pop index out of range
```

3. remove()

The remove() function used to remove an element from list whose value is provided as parameter.

Note: remove() will not show the deleted element.

Syntax:

```
List_variable.remove(element)
```

Example:

```
Lst=[10,30,10,20,10]
Lst.remove(10)
```

```
Print(Lst)          # Output: [30,10,20,10]
Lst.remove(100)
# ValueError: list.remove(x): x not in list
```

4. del keyword

The del keyword is used to delete not only list nut also any variable in Python from memory.

If we provide a range of elements of list then it will delete the particular range from list and the list variable remains in memory.

When we provide a list variable without range then del keyword will delete the complete list variable from memory and no further allowed to use it.

Syntax:

```
del List_variable[index / range]      OR
del Lsit_variable
```

Example:

```
Lst=[10,30,20,50,40]
del Lst[2]
# It will remove the element from index-2.
Print(Lst)      # Output: [10,30,50,40]
del Lst[1:3]
Print(Lst)      # Output: [10,40]
del Lst
# It will remove List variable from memory
print(Lst)
NameError: name 'Lst' is not defined
```

5. max()

This function will return maximum value from the list.

Syntax:

max(List_Variable) OR max(lsit[range])

Example:

```
Lst=[10,40,20,15,26,38]
Print(max(Lst))                 # Output:40
Print(max(Lst[2:5]) )           # Output:26
L=["KVS","JJN","JPR"]
Print(max(Lst))                 # Output:'KVS'
```

6. min()

This function will return minimum value from the list.

Syntax:

min(List_Variable) OR min(lsit[range])

Example:

```
Lst=[10,40,20,15,26,38]
Print(min(Lst))                 # Output:10
Print(min(Lst[2:5]) )           # Output:15
L=["KVS","JJN","JPR"]
Print(max(Lst))                 # Output:'JJN'
```

Assignments:

1. Write a program to find reverse of List.
2. WAP to check that List is palindrome or not.
3. WAP to find Odd number in List.
4. WAP to find Even number in List.
5. WAP to find Sum of odd index elements in List.
6. WAP to find Sum of even index elements in List.
7. WAP to find maximum number in List.
8. WAP to find minimum number in List.
9. WAP to print elements in ascending order.
10. WAP to print elements in descending order.