

## 8.5 COPYING LISTS

Given a list, the simplest way to make a copy of the list is to assign it to another list.

```
>>> list1 = [1,2,3]
>>> list2 = list1
>>> list1
[1, 2, 3]
>>> list2
[1, 2, 3]
```

The statement `list2 = list1` does not create a new list. Rather, it just makes `list1` and `list2` refer to the same list object. Here, `list2` actually becomes an alias of `list1`. Therefore, any changes made to either of them will be reflected in the other list.

```
>>> list1.append(10)
>>> list1
[1, 2, 3, 10]
>>> list2
[1, 2, 3, 10]
```

We can also create a copy or clone of the list as a distinct object by three methods. The first method uses slicing, the second method uses built-in function `list()` and the third method uses `copy()` function of Python library.

**Method 1:** We can slice our original list and store it into a new list variable as follows:

```
newList = oldList[:]
```

```
>>> list1 = [1,2,3,4,5]
>>> list2 = list1[:]
>>> list2
[1, 2, 3, 4, 5]
```

**Method 2:** We can use the list constructor function `list()` as follows:

```
newList = list(oldList)
```

```
>>> list1 = [10,20,30,40]
>>> list2 = list(list1)
>>> list2
[10, 20, 30, 40]
```

Calling it without any arguments will create an empty list.

**Method 3:** We can use the `copy()` function as follows:

```
newList = list(oldList).copy()
```

```
>>> list1=[1,2,3,4,5]
>>> list2=list1.copy()
>>> list2
[1, 2, 3, 4, 5]
>>> list1[0]=100
>>> list1
[100, 2, 3, 4, 5]
>>> list2
[1, 2, 3, 4, 5]
>>> list2[2]=300
>>> list2
[1, 2, 300, 4, 5]
>>> list1
[100, 2, 3, 4, 5]
```

The `copy()` function returns a new list stored in a different memory location. It doesn't modify the original list or the modifications made in the new list will not be reflected to the base list.

## 8.6 BUILT-IN FUNCTIONS (MANIPULATING LISTS)

Python offers several built-in 'in-place' functions that can alter the elements of the list rather than creating a new list, such as adding new elements, changing the elements in a list, etc. These functions perform various operations on lists which are described in Fig. 8.11.

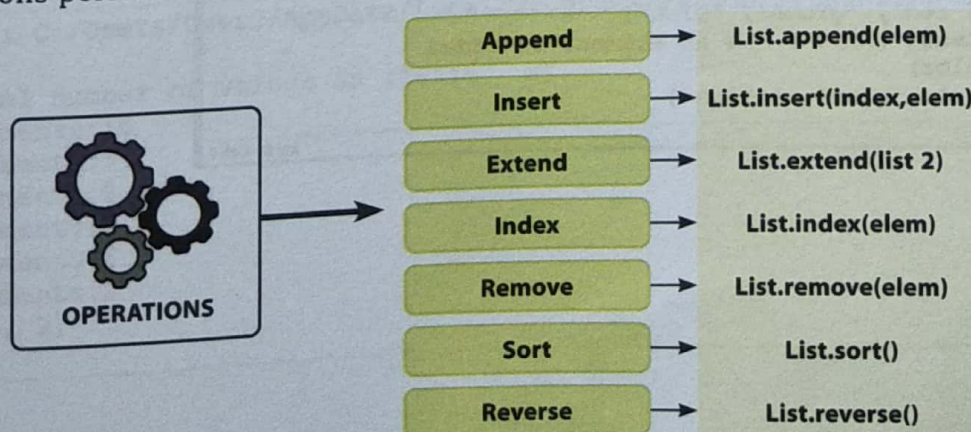


Fig. 8.11: Built-in List operations



### 8.6.1 append()

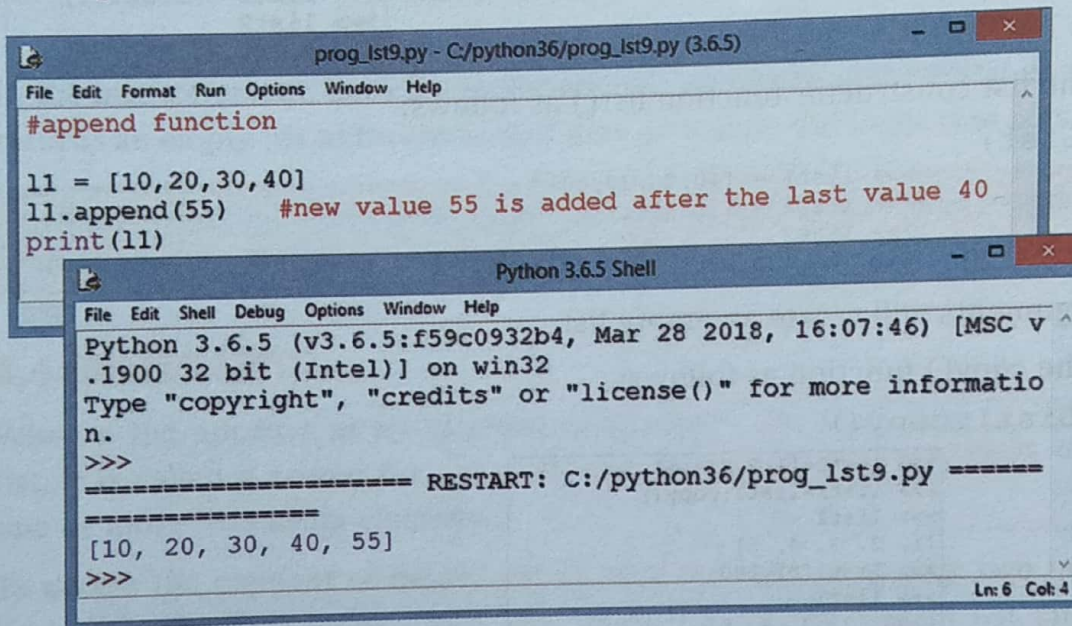
The `append()` method adds a single item to the end of the list. It doesn't create a new list; rather it modifies the original list. The single element can also be a list. The syntax of `append()` method is:

#### Syntax:

```
list.append(item)
```

The item can be numbers, strings, another list, dictionary, etc.

#### Example 3:



The screenshot shows two windows from a Python IDE. The top window, titled 'prog\_lst9.py - C:/python36/prog\_lst9.py (3.6.5)', contains the following code:

```
#append function  
l1 = [10,20,30,40]  
l1.append(55)    #new value 55 is added after the last value 40  
print(l1)
```

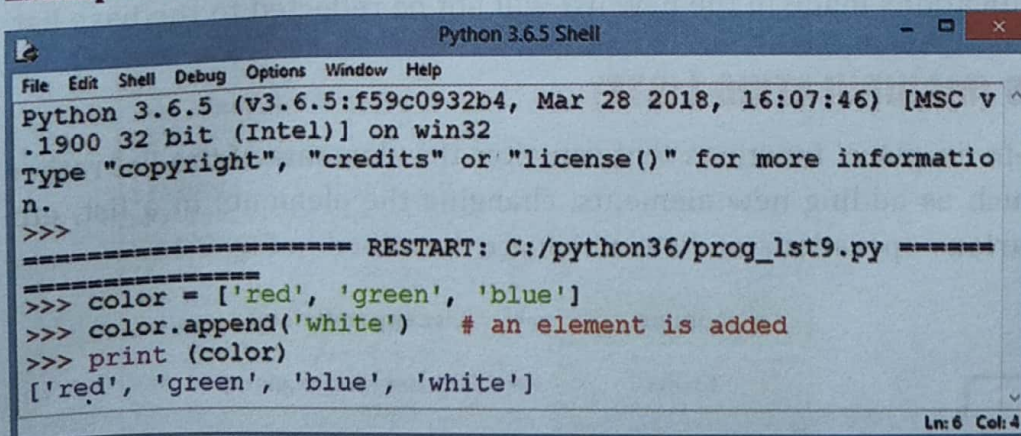
The bottom window, titled 'Python 3.6.5 Shell', shows the output of running the script:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v  
.1900 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more informatio  
n.  
>>>  
===== RESTART: C:/python36/prog_lst9.py =====  
[10, 20, 30, 40, 55]  
>>>
```

A list can also be added to the above existing list. For example,

```
>>> l1 = [10,20,30,40]  
>>> l1.append([50,60])  
>>> l1  
[10, 20, 30, 40, [50, 60]]
```

#### Example 4:



The screenshot shows a 'Python 3.6.5 Shell' window with the following code and output:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v  
.1900 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more informatio  
n.  
>>>  
===== RESTART: C:/python36/prog_lst9.py =====  
>>> color = ['red', 'green', 'blue']  
>>> color.append('white')    # an element is added  
>>> print (color)  
['red', 'green', 'blue', 'white']
```



## Practical Implementation-1

Write a program to find the second largest element in a list 'Num'.

```
#Program to find the second largest element in a list 'Num'
print("Enter the number of elements in a list: ")
N = int(input())
i = 0
num = []
while i < N:
    print("Enter list values:")
    num1 = int(input())
    num.append(num1)
    i += 1
print("The Original List is:", end=' ')
for i in range(N):
    print(num[i], end=' ')
if (num[0] > num[1]):
    m, m2 = num[0], num[1]
else:
    m, m2 = num[1], num[0]
for x in num[2:]:
    if x > m2:
        if x > m:
            m2, m = m, x
        else:
            m2 = x
print()
print("The second largest element in the list is:", m2)
```

```
>>>
RESTART: C:/Users/preeti/AppData/Local/Programs
list.py
Enter the number of elements in a list:
5
Enter list values:
7
Enter list values:
33
Enter list values:
66
Enter list values:
8
Enter list values:
99
The Original List is: 7 33 66 8 99
The second largest element in the list is: 66
```

## Practical Implementation-2

Give Python code to accept values from the user up to a certain limit entered by the user; if the number is even, then add it to a list.

```
list_even_ch-7_CS.py - C:/Users/User2/AppData/Local/...
File Edit Format Run Options Window Help
L=[]
n=int(input("Enter total number of values in list:"))
i=1
while i<=n:
    a=int(input("Enter elements:"))
    if a%2==0:
        L.append(a)
        i=i+1
print(L)
Ln: 10 Col: 0
```

\*\*\*\*\*Output of the program\*\*\*\*\*

```
= RESTART: C:/Users/User2/AppData/Local/Programs/Python/Python39/list_even_ch-7_
CS.py
Enter total number of values in list:4
Enter elements:10
Enter elements:5
Enter elements:8
Enter elements:3
Enter elements:6
Enter elements:2
[10, 8, 6, 2]
>>>
```



### 8.6.2 extend()

The extend() method adds one list at the end of another list. In other words, all the items of a list are added at the end of an already created list.

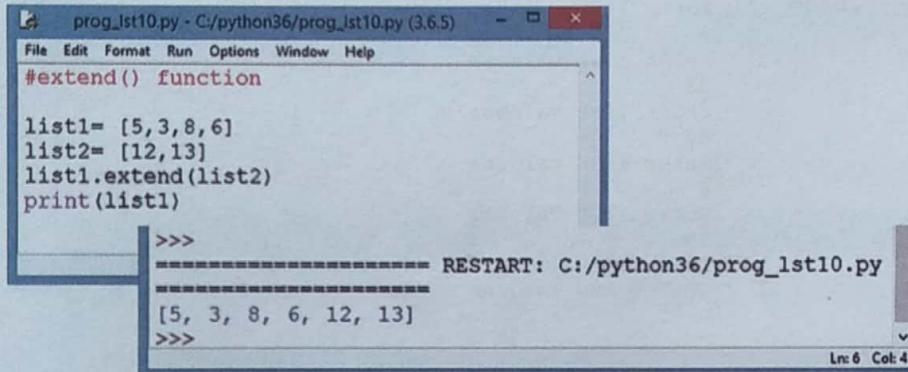
**Syntax:**

```
list1.extend(list2)
```

→ The list which will be added to list1.

→ list1 is the primary list which will be extended.

**Example 5:**



```
prog_lst10.py - C:/python36/prog_lst10.py (3.6.5)
File Edit Format Run Options Window Help
#extend() function

list1= [5,3,8,6]
list2= [12,13]
list1.extend(list2)
print(list1)

>>>
===== RESTART: C:/python36/prog_lst10.py
=====
[5, 3, 8, 6, 12, 13]
>>>
```

**Example 6:**

```
>>> L1=[100,200,300,400] → list1
>>> L2=[10,20,30] → list2
>>> L1.extend(L2)
>>> print(L1)
```

→ items of list2 added to list1

```
[100, 200, 300, 400, 10, 20, 30]
```

### 8.6.3 insert()

The insert() function can be used to insert an element/object at a specified index. This function takes two arguments: the index where an item/object is to be inserted and the item/element itself.

**Syntax:**

```
list_name.insert(index_number, value)
```

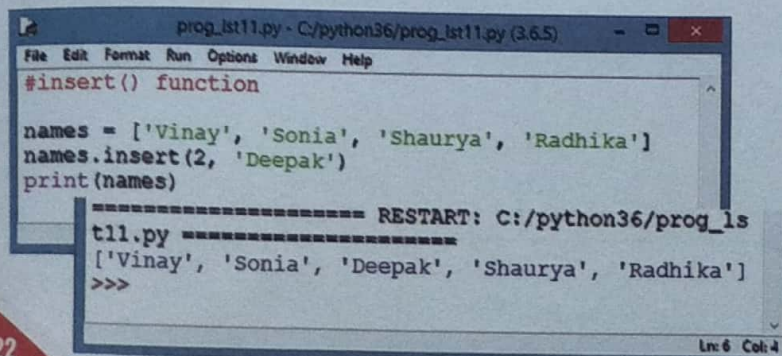
where,

list\_name is the name of the list

index\_number is the index where the new value is to be inserted

value is the new value to be inserted in the list

**Example 7:**



```
prog_lst11.py - C:/python36/prog_lst11.py (3.6.5)
File Edit Format Run Options Window Help
#insert() function

names = ['Vinay', 'Sonia', 'Shaurya', 'Radhika']
names.insert(2, 'Deepak')
print(names)

===== RESTART: C:/python36/prog_ls
=====
t11.py
['Vinay', 'Sonia', 'Deepak', 'Shaurya', 'Radhika']
>>>
```



### Example 8:

```
>>> l1 = [10,20,30,40]
>>> l1.append([50,60])
>>> l1
[10, 20, 30, 40, [50, 60]]
>>> list1 = [10,20,30,40,50]
>>> list1.insert(2,25)
>>> list1
[10, 20, 25, 30, 40, 50]
>>> list1.insert(0,5)
>>> list1
[5, 10, 20, 25, 30, 40, 50]
```

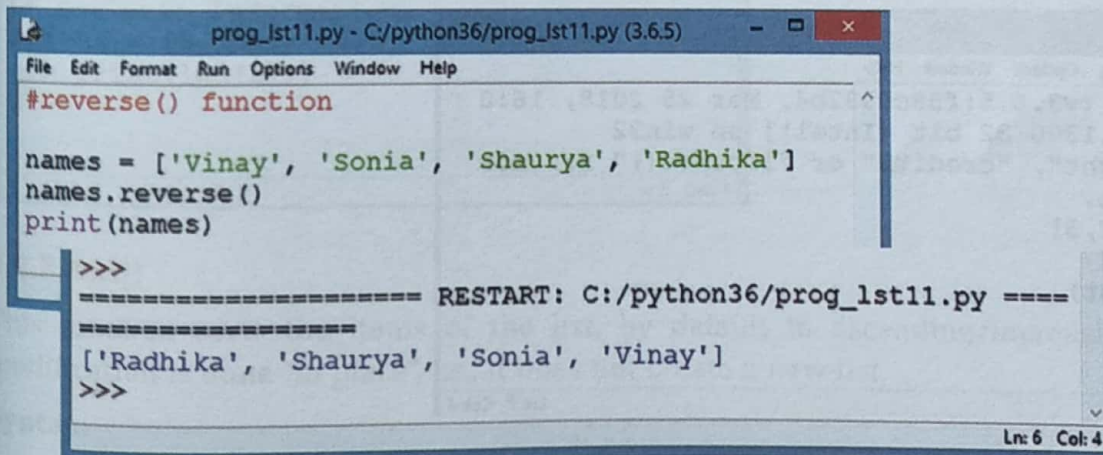
### 8.6.4 reverse()

The `reverse()` function in Python reverses the order of the elements in a list. This function reverses the item of the list. It replaces a new value "in place" of an item that already exists in the list, *i.e.*, it does not create a new list.

### Example 9:

```
>>> list1 = [24,56,10,99,28,90]
>>> list1.reverse()
>>> list1
[90, 28, 99, 10, 56, 24]
```

### Example 10:



```
prog_lst11.py - C:/python36/prog_lst11.py (3.6.5)
File Edit Format Run Options Window Help
#reverse() function
names = ['Vinay', 'Sonia', 'Shaurya', 'Radhika']
names.reverse()
print(names)

>>>
===== RESTART: C:/python36/prog_lst11.py =====
['Radhika', 'Shaurya', 'Sonia', 'Vinay']
>>>

Ln: 6 Col: 4
```

### 8.6.5 index()

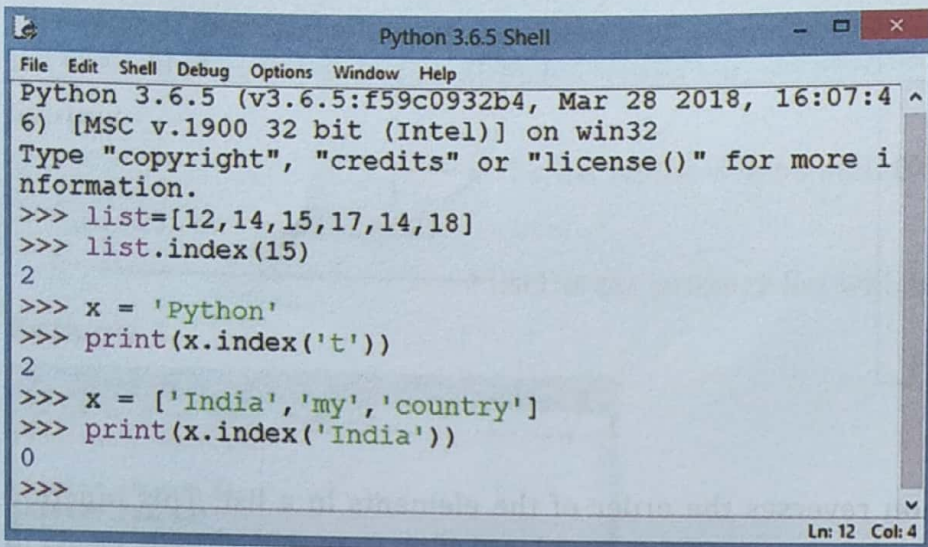
The `index()` function in Python returns the index of first matched item from the list. It returns the first occurrence of an item for which the index position is to be searched for in the list. If the element is not present, `ValueError` is generated.

#### Syntax:

```
List.index(<item>)
```



### Example 11:



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> list=[12,14,15,17,14,18]
>>> list.index(15)
2
>>> x = 'Python'
>>> print(x.index('t'))
2
>>> x = ['India','my','country']
>>> print(x.index('India'))
0
>>>
```

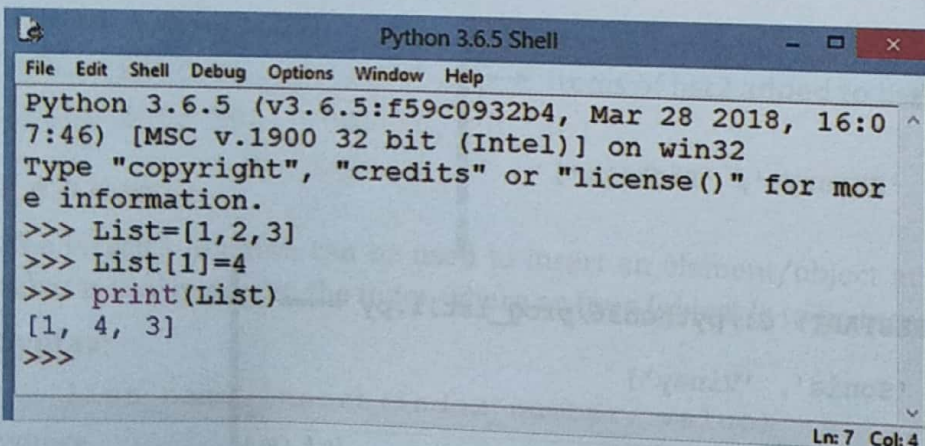
### 8.6.6 Updating List

Lists are mutable; you can assign new value to existing value. We can use assignment operator (=) to change an item or a range of items.

#### Syntax:

List[index]=<new value>

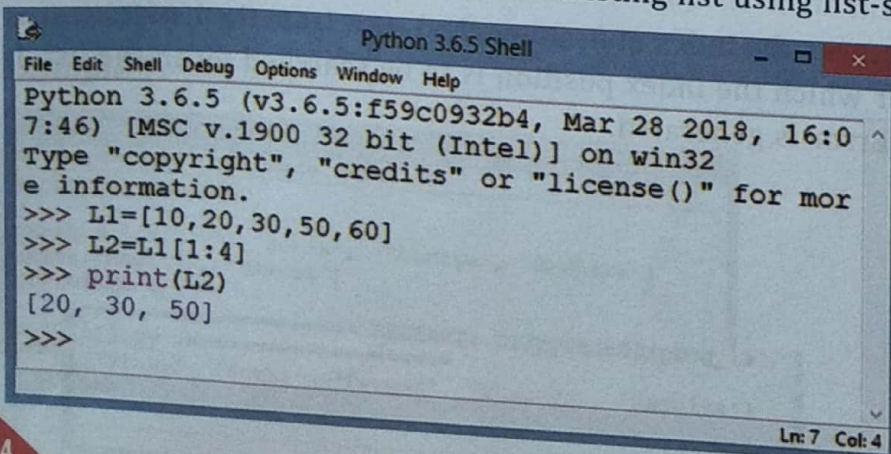
### Example 12:



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> List=[1,2,3]
>>> List[1]=4
>>> print(List)
[1, 4, 3]
>>>
```

### Example 13:

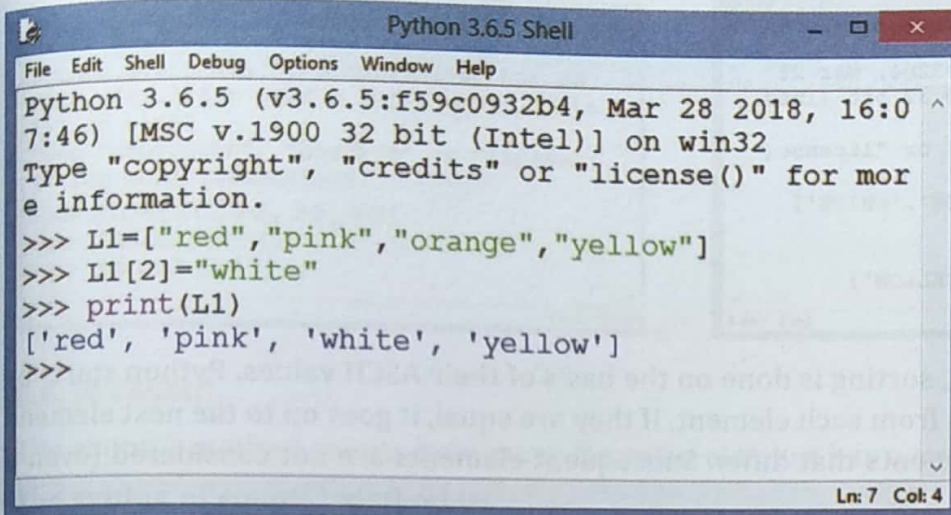
We can also create a new list from an existing list using list-slicing.



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> L1=[10,20,30,50,60]
>>> L2=L1[1:4]
>>> print(L2)
[20, 30, 50]
>>>
```



### Example 14:



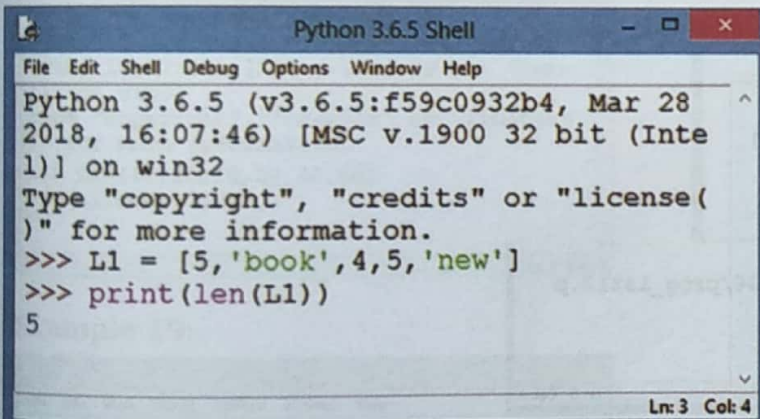
```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> L1=["red","pink","orange","yellow"]
>>> L1[2]="white"
>>> print(L1)
['red', 'pink', 'white', 'yellow']
>>>
```

Ln: 7 Col: 4

### 8.6.7 len()

The len() function returns the length of the list, i.e., the number of elements in a list.



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> L1 = [5, 'book', 4, 5, 'new']
>>> print(len(L1))
5
```

Ln: 3 Col: 4

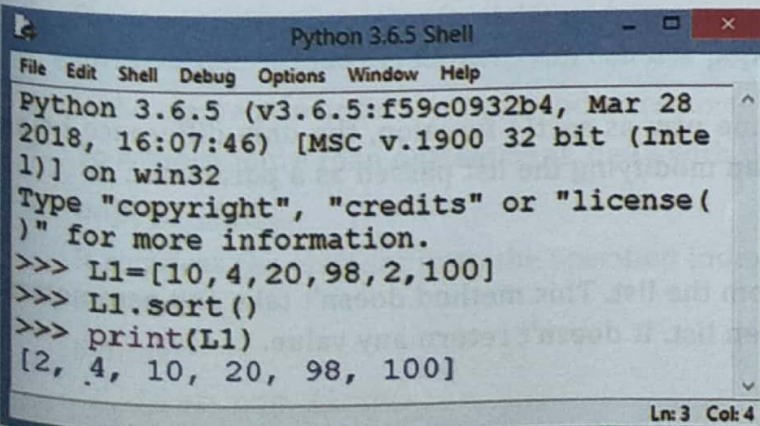
### 8.6.8 sort()

This function sorts the items of the list, by default in ascending/increasing order. This modification is done "in place", i.e., it does not create a new list.

Syntax:

```
List.sort()
```

### Example 15:



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> L1=[10,4,20,98,2,100]
>>> L1.sort()
>>> print(L1)
[2, 4, 10, 20, 98, 100]
```

Ln: 3 Col: 4



### Example 16:

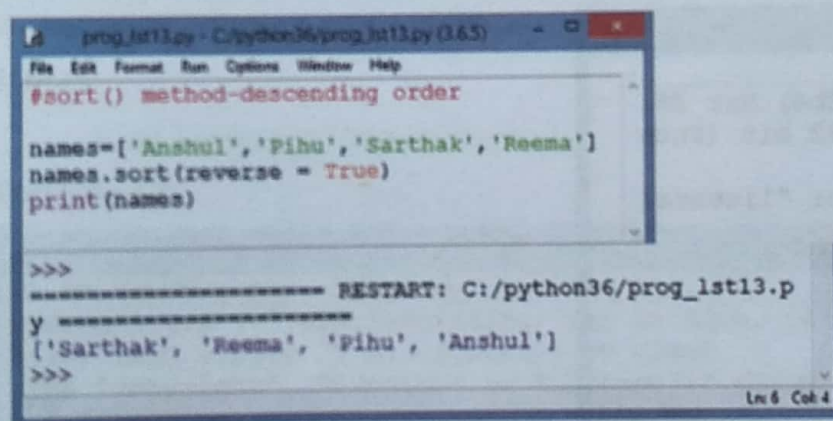


```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28
2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license(
)" for more information.
>>> L1=['RED', 'YELLOW', 'BLUE', 'WHITE']
>>> L1.sort()
>>> print(L1)
['BLUE', 'RED', 'WHITE', 'YELLOW']
```

For string elements in a list, sorting is done on the basis of their ASCII values. Python starts by comparing the first element from each element. If they are equal, it goes on to the next element, and so on, until it finds elements that differ. Subsequent elements are not considered (even if they are really big).

It sorts the string in a lexicographic manner. If we want to sort the list in decreasing order, we need to write:

```
>>> list1.sort(reverse=True)
```



```
prog_lst13.py - C:/python36/prog_lst13.py (3.6.5)
File Edit Format Run Options Window Help
#sort() method-descending order
names=['Anshul', 'Pihu', 'Sarthak', 'Reema']
names.sort(reverse = True)
print(names)

>>>
===== RESTART: C:/python36/prog_lst13.p
y =====
['Sarthak', 'Reema', 'Pihu', 'Anshul']
>>>
```

**sorted()**—This function also sorts the items of the list like **sort()** function.

**Syntax:** **sorted(list, reverse = True/False)**

```
>>> list1 = [4,2,1,9,3,4,6,8,5]
>>> sorted(list1)
[1,2,3,4,4,5,6,8,9]
>>> sorted(list1, reverse=True)
[9,8,6,5,4,4,3,2,1]
>>> sorted(list1, reverse=False)
[1,2,3,4,4,5,6,8,9]
```

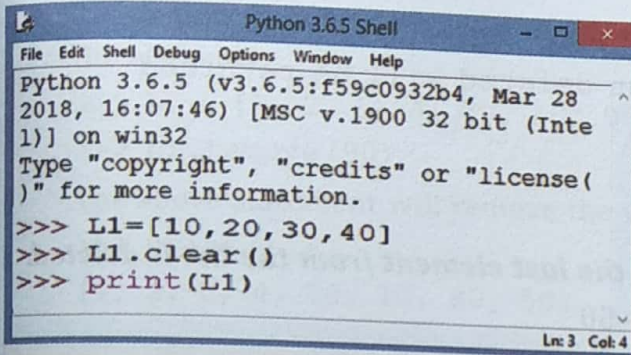
**Note:** **sorted()** function works in the same way as **sort()** function, the only difference being that it returns a new list rather than modifying the list passed as a parameter.

#### 8.6.9 clear()

The **clear()** method removes all items from the list. This method doesn't take any parameters. The **clear()** method only empties the given list. It doesn't return any value.



### Example 17:



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> L1=[10,20,30,40]
>>> L1.clear()
>>> print(L1)
```

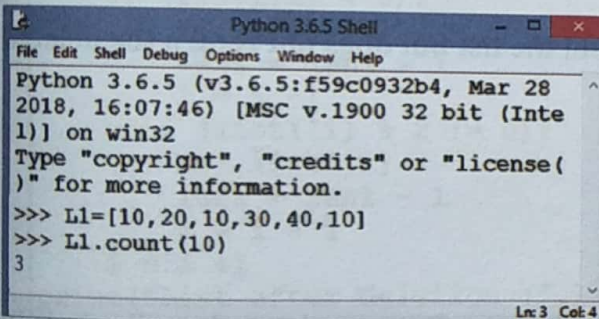
### 8.6.10 count()

The count() method counts how many times an element has occurred in a list and returns it.

The syntax of count() method is:

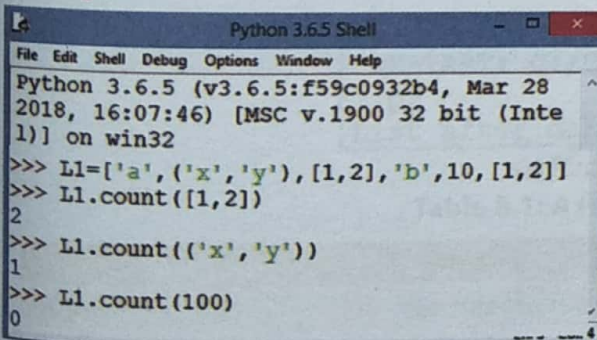
```
list.count(element)
```

### Example 18:



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> L1=[10,20,10,30,40,10]
>>> L1.count(10)
3
```

### Example 19:



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
>>> L1=['a',('x','y'),[1,2],'b',10,[1,2]]
>>> L1.count([1,2])
2
>>> L1.count(('x','y'))
1
>>> L1.count(100)
0
```

## 8.7 DELETION OPERATION

1. Python provides operator for deleting/removing an item from a list. There are many methods for deletion: If index is known, you can use pop() or del statement.
2. If the element is known, not the index, remove() can be used.
3. To remove more than one element, del() with list slice can be used.

#### ➤ pop():

It removes the element from the specified index and also returns the element which was removed.

#### Syntax:

```
List.pop (index)
```



### Example 20:

```
>>> L1 = [1, 2, 5, 4, 70, 10, 90, 80, 50]
>>> L1.pop(1)      # here the element deleted will be returned
>>> 2              # 2 gets deleted
>>> print(L1)
[1, 5, 4, 70, 10, 90, 80, 50]
```

**If no index value is provided in pop(), then the last element from the list is deleted.**

```
>>> L1.pop() -----50
```

**CTM:** If no index value is provided in pop(), then the last element is deleted.

pop() can be used with negative index value also.

```
>>> L1=[100, 200, 300, 500]
>>> L1.pop(-1)
500
```

### ➤ del statement

del statement removes the specified element from the list but does not return the deleted value.

```
>>> L1=[100, 200, 300, 400, 500]
>>> del L1[3]
>>> print(L1)
[100, 200, 300, 500]
```

**Note:** If an out of range index is provided with del() and pop(), the code will result in runtime error.

```
>>> L1=[100,200,300,400,500]
>>> del L1[5]
```

Traceback (most recent call last):

```
File "<pyshell#31>", line 1, in <module>
    del L1[5]
```

IndexError: list assignment index out of range

del can be used with negative index value also.

```
>>> L1=[100,200,300,400,500]
>>> del L1[-2]
>>> print(L1)
[100, 200, 300, 500]
```

### ➤ del() with slicing

Consider the following example:

```
>>> L1=[10,20,30,40,50]
>>> del L1[2:4]
>>> print(L1)
[10, 20, 50]
```

The above del statement shall remove 2<sup>nd</sup> and 3<sup>rd</sup> elements from the list. As we know that slice selects all the elements up to 4<sup>th</sup> index, so 4<sup>th</sup> element will remain in the list.



### ➤ remove()

The remove() function is used when we know the element to be deleted, not the index of the element.

```
>>> L1 = [1, 2, 5, 4, 70, 10, 90, 80, 50]
>>> L1.remove(90)
```

The above statement will remove the value 90 from the list.

```
>>> print(L1)
[1, 2, 5, 4, 70, 10, 80, 50]
```

### Practical Implementation-3

Write a Python code to delete all odd numbers and negative numbers from a given numeric list.

```
#Program to delete all odd numbers and negative numbers in a numeric list
list1 = [11,-1,22,-3,33,55,44,-50,46,101,77,-100,42]
len1 = len(list1)
i = 0
while i < len1:
    if (list1[i] < 0):
        del list1[i]
        len1 = len1 - 1
        i = i - 1
    elif (list1[i] % 2 != 0):
        del list1[i]
        len1 = len1 - 1
        i = i - 1
    i = i + 1
print("List after deletion :",list1)
```

\*\*\*\*\*Output of the program\*\*\*\*\*

```
>>>
RESTART: C:/Users/preeti/AppData/Local
1.PY
List after deletion : [22, 44, 46, 42]
```

Table 8.1: A few List Methods in a Nutshell

Method	Result
append(item)	Add item to the end of the list.
index(item)	Returns the index of the first element whose value is equal to item. A ValueError exception is raised if item is not found in the list.
insert(index, item)	Inserts item into the list at the specified index. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at a specified index and all the items after it are shifted by one position towards the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
sort()	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
remove(item)	Removes the first occurrence of item from the list. A ValueError exception is raised if item is not found in the list.
reverse()	Reverses the order of the items in the list.



## ➤ Searching List

Lists can easily be searched for values using the `index()` method that expects a value that is to be searched. The output is the index at which the value is kept.

### Syntax:

```
list_name.index(element)
```

### Example 21:

```
>>> list1=[100, 200, 50, 400, 500, 'Raman', 100, 'Ashwin']
>>> list1.index(400)
3
```

### Explanation:

Here we are searching list1 for value 400. The answer is 3 as the element 400 is present at the index value 3.

If the value, we are searching in a list is not found, then an error is displayed.

### Example 22:

```
>>> list1=[100, 200, 50, 400, 500, 'Raman', 100, 'Ashwin']
>>> list1.index(100)
0
```

This method only returns the first occurrence of the matching element. Here, the element 100 is present at 0<sup>th</sup> and 6<sup>th</sup> index, but it returns 0<sup>th</sup> index.

### Example 23:

```
>>> list1=[100, 200, 50, 400, 500, 'Raman', 100, 'Ashwin']
>>> list1.index(700)
```

```
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    list1.index(700)
```

```
ValueError: 700 is not in list
```

### Example 24:

To find out whether an element is present in a list or not, we can do the following sequence of operations:

```
>>> list1 = [100, 200, 50, 400, 500, 'Raman', 100, 'Ashwin']
>>> 500 in list1
True
```

### Explanation:

So, from the above example it is evident that 'True' was returned as '500' was present in list1.

## ➤ max()

Returns the element with the maximum value from the list.

**Note:** To use the `max()` function, all values in the list must be of the same type.

### Example 25:

```
>>> list3=[10,20,30,40]
>>> max(list3)
40
```



**Example 26:**

```
>>> list4=['A', 'a', 'B','C']
>>> max(list4)
'a'
```

Here, it will return the alphabet with the maximum ASCII value.

**Example 27:**

```
>>> list5=['ashwin','bharat','shelly']
>>> max(list5)
'shelly'
```

Here, it will return the string which starts with character having the highest ASCII value.

**Example 28:**

```
>>> list5=['ashwin','bharat','shelly', 'surpreet']
>>> max(list5)
'surpreet'
```

If there are two or more strings which start with the same character, then the second character is compared and so on.

➤ **min()**

Returns the element with the minimum value from the list.

**Note:** To use the min() function, all values in the list must be of the same type.

**Example 29:**

```
>>> list3=[10,20,30,40]
>>> min(list3)
10
```

**Example 30:**

```
>>> list4=['A', 'a', 'B','C']
>>> min(list4)
'A'
```

Here, it will return the alphabet with the minimum ASCII value.

**Example 31:**

```
>>> list5=['ashwin','bharat','shelly']
>>> min(list5)
'ashwin'
```

Here, it will return the string which starts with character having the smallest ASCII value.

**Example 32:**

```
>>> list5=['ashwin','bharat','shelly', 'surpreet']
>>> min(list5)
'ashwin'
```

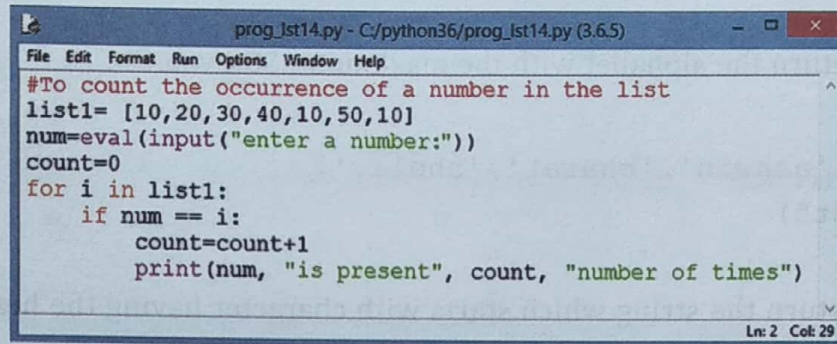
If there are two or more strings which start with the same character, then the second character is compared and so on.



### Practical Implementation-4

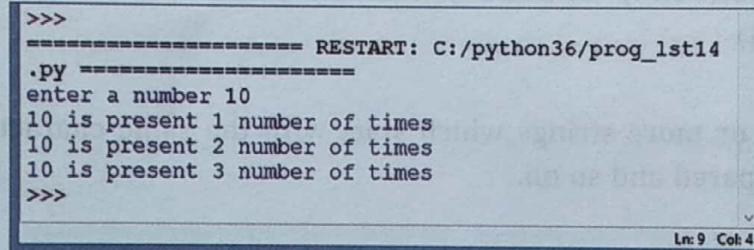
Write a Python script to input a number and count the occurrences of that number in a given list.

```
list1=[10, 20, 30, 40, 10, 50, 10]
```



```
prog_lst14.py - C:/python36/prog_lst14.py (3.6.5)
File Edit Format Run Options Window Help
#To count the occurrence of a number in the list
list1= [10,20,30,40,10,50,10]
num=eval(input("enter a number:"))
count=0
for i in list1:
    if num == i:
        count=count+1
        print(num, "is present", count, "number of times")
Ln: 2 Col: 29
```

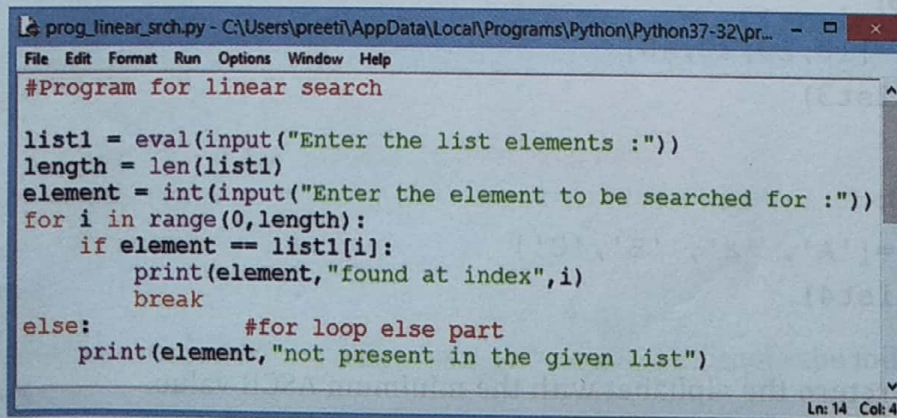
\*\*\*\*\*Output of the program\*\*\*\*\*



```
>>>
===== RESTART: C:/python36/prog_lst14
.py =====
enter a number 10
10 is present 1 number of times
10 is present 2 number of times
10 is present 3 number of times
>>>
Ln: 9 Col: 4
```

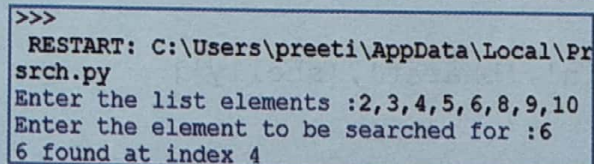
### Practical Implementation-5

Write a Python script to input a number and perform linear search.



```
prog_linear_srch.py - C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\pr...
File Edit Format Run Options Window Help
#Program for linear search

list1 = eval(input("Enter the list elements :"))
length = len(list1)
element = int(input("Enter the element to be searched for :"))
for i in range(0,length):
    if element == list1[i]:
        print(element,"found at index",i)
        break
else:
    #for loop else part
    print(element,"not present in the given list")
Ln: 14 Col: 4
```



```
>>>
RESTART: C:\Users\preeti\AppData\Local\Pr
srch.py
Enter the list elements :2,3,4,5,6,8,9,10
Enter the element to be searched for :6
6 found at index 4
```

### Practical Implementation-6

The temperature of a week is given in WTemp list in the form of day and its temperature values are as follows:

```
WTemp = ['Mon', 45, 'Tue', 43, 'Wed', 42, 'Thu', 40, 'Fri', 38, 'Sat', 40,
         'Sun', 38]
```

Using the above list, write a program to create and print two separate lists as given below:

```
Days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
```

```
Degrees = [45, 43, 42, 40, 38, 40, 38]
```



```

#Program to split Original list into two separate lists
WTemp = ['Mon', 45, 'Tue', 43, 'Wed', 42, 'Thu', 40, 'Fri', 38, 'Sat', 40, 'Sun', 38]
ln = len(WTemp)
Days = []
Degrees = []
for i in range(ln):
    if (i % 2) == 0:
        Days.append(WTemp[i])
    else:
        Degrees.append(WTemp[i])
print("Original list is: ", WTemp)
print("The days are: ", Days)
print("The temperatures are: ", Degrees)

```

```

>>>
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_split_1
ist1.py
Original list is: ['Mon', 45, 'Tue', 43, 'Wed', 42, 'Thu', 40, 'Fri', 38, 'Sat',
, 40, 'Sun', 38]
The days are: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
The temperatures are: [45, 43, 42, 40, 38, 40, 38]

```