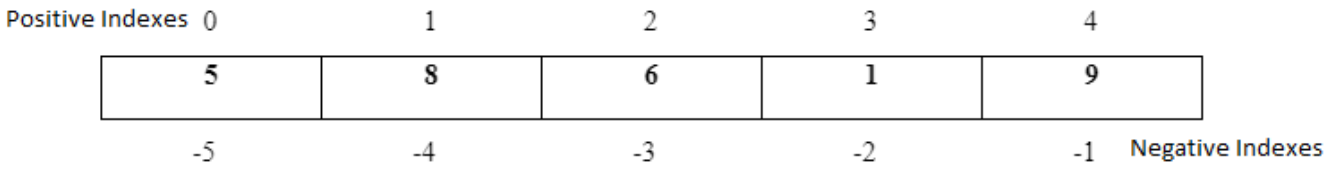
LIST IN PYTHON

- List is a **sequence** i.e. sequenced data type.
- List can store multiple values in a single object.
- List is an **ordered sequence** of values/elements.
- Elements of a list are enclosed in **square brackets** [ ], separated by commas.
- List are **mutable** i.e. values in the list can be modified.
- The elements in a list can be of **any type** such as integer, float, string, objects etc.
- List is **heterogeneous** type i.e. collection of different types.

L = [5, 8, 6, 1, 9]

Length = 5 (Number of elements)



What is a sequence?

Sequence is a positional ordered collection of items, which can be accessed using index. Examples of sequence are List, Tuple, String etc.

List is Mutable

In Python, lists are mutable. It means that the contents of the list can be changed after it has been created.

```
>>> L = [1, 5, 7, 2]

>>> L

[1, 5, 7, 2]
```

Change the second element of list from 5 to 8

```
>>> L[1]=8

>>> L

[1, 8, 7, 2]
```

How to create a list in Python:

Using integer values

```
>>> L = [1, 5, 7, 2]

>>> L

[1, 5, 7, 2]
```

Using heterogeneous type:

```
>>> L = [1.5, 2, "Hello", 'S']

>>> L

 [1.5, 2, "Hello", 'S']
```

Using list( ) function:

list ( ) function is useful to create or convert any iterable into list. For example, create a list from string.

```
>>> L=list("Hello")

>>> L

['H', 'e', 'l', 'l', 'o']
```

Creating Empty List:

```
>>> L=[ ]

>>> L

[ ]
```

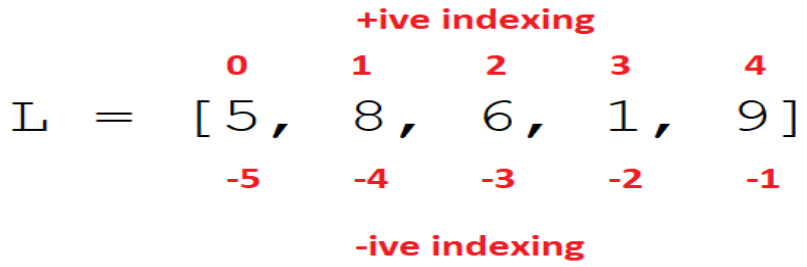
Creating empty list using list( )

```
>>> L=list( )

>>> L

[]
```

Indexing in List:

- Index means the position of an item in an iterable type.
- Indexing can be positive or negative.
- Positive Indexes start from 0 and are positioned like 0, 1, 2, ….
- The index of the first element is always 0 and the last element is n-1 where n is the total number of elements in the list.
- Negative index starts from the last element and is labeled as -1, -2, -3, ……

For example:



Accessing elements using index:

Syntax

Object_name[Index]

For Ex:

```
>>> L = [5, 8, 6, 1, 9]
>>> L[0]
5
 >>> L[3]
1
 >>> L[-1]
9
 >>> L[-4]
8
 >>> L[5]
IndexError: list index out of range
```

```
>>> L[-6]

IndexError: list index out of range
```

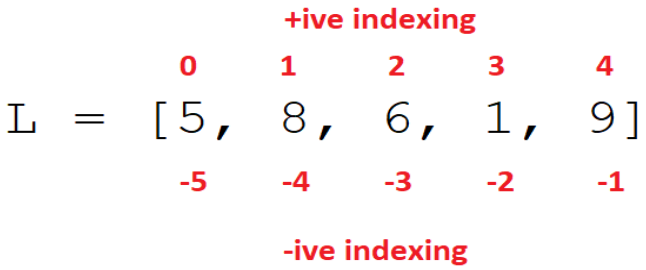
If we try to access the element from outside the index range then the interpreter raises an IndexError.

LIST OPERATIONS

Slicing in List:

Note: Slicing never gives IndexError
Syntax:
Object_name[start : stop : interval]

- Here start, stop and interval all have default values.
- Default value of start is 0, stop is n (if interval is +ive) and -1 if interval is –ive
- Default value interval is 1
- In slicing, start is included and stop is excluded.
- If interval is +ive then slicing is performed from left to right. For Ex:




```
>>> L[1:4:1]

[8, 6, 1]
```

If we give an index out of range then it is considered as the default value.

```
>>> L[1:5:2]

[8, 1]
```

Start index is not given then it consider as default value

```
>>> L[:5:2]

[5, 6, 9]
```

Start, stop and interval is not given then they consider as default value

>>> L[: :]

[5, 8, 6, 1, 9]

If start>=stop and interval is positive the slicing will return empty list

>>> L[4:2]

[ ]

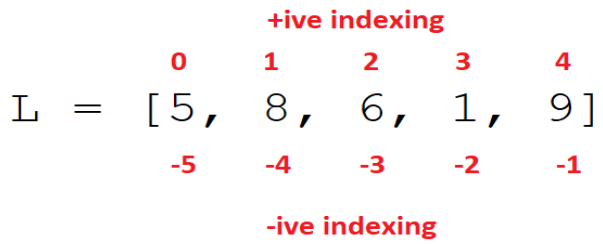
>>> L[4:4]

[ ]

>>> L[4:2:1]

[ ]

If the interval is negative then slicing is performed from right to left. For Ex:



Access the list element where start index is 4 and stop index is 1 in reverse order

>>>L[4:1:-1]
[9, 1, 6]

Access the list element where start index is 5 and stop index is 0 with interval 2 in reverse order
>>>L[5:0:-2]
[9, 6]

>>>L[-1:-4:-1]
[9, 1, 6]

>>>L[-1:-4:-1]
[9, 1, 6]

Access the list in reverse order
>>>L[: : -1]
[9, 1, 6, 8, 5]

```
>>> L[-1:2:-1]
[9, 1]
```

Concatenation of two lists:

Concatenation is the process of joining the elements of a particular list at the end of another list

```
>>> L1=[1,2,3]
>>> L2=['a','b','c']

Using concatenation operator (+)
>>> L1+L2
[1, 2, 3, 'a', 'b', 'c']
```

Repetition Operator

- The repetition operator makes multiple copies of a list and joins them all together. The general format is: list * n,
  where 'list' is a list and n is the number of copies to make.

```
>>> L=[0]*3
>>> L
[0, 0, 0]
```

In this example, [0] is a list with one element 0, The repetition operator makes 3 copies of this list and joins them all together into a single list.

```
>>> L = [1, 2, 3]*3
>>> L
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

In this example, [1, 2, 3] is a list with three elements 1, 2 and 3. The repetition operator makes 3 copies of this list and joins them all together into a single list.

Membership operator

Membership operator is used to check or validate the membership of an element in the list or sequence.

There are two types of membership operators
1. in operator: It return True if a element with the specified value is present in the List
2. not in operator: It return True if a element with the specified value is not present in the List

Examples:

```
L = [5,"India", 8, 6, 1]
>>> 5 in L
  True

>>> "India" in L
  True

>>> 9 in L
  False

>>> 9 not in L
  True
```

Traversing a list using loops

Traversing a list means to visit every element of the list one by one. One of the most common methods to traverse a Python list is to use a for loop.

Ex: Write a program in Python to traverse a list using for loop.

```
L = [5, 8, 9, 6, 1]
for i in L:
     print(i)
```

Output:

5
8
9
6
1

Using range() method:

We can traverse a list using range() method

```
L = [5, 8, 9, 6, 1]
n = len(L)
for i in range(n):
     print(L[i])
```

Output:

5
8
9

6
1

Built-in functions/methods of list:

1: len():
It returns the total number of elements in the list i.e. length of the list.
For ex:

```
>>> L = [8, 6, 3, 9, 1]
>>> len(L)
5
```

It returns the total numbers of elements in the list i.e. 5

2: list()
list() is a function used to convert an iterable into a list.
For Ex:

```
>>> S="Python"
>>> L=list(S)
>>> L
['P', 'y', 't', 'h', 'o', 'n']
```

In the above example, a string S is converted into the List.

3: append()
appends an element at the end of the list.
list.append(item)
append() adds a single element to the end of a list.
The length of the list increases by one.
The method doesn't return any value

```
>>> L = [4,5,9,2,6]
>>> L.append(1)
>>> L
[4, 5, 9, 2, 6, 1]
```

A list is an object. If we append another list onto a list, the parameter list will be a single object at the end of the list.

```
>>> L1=[1,2,3]
>>> L2=[4,5,6]
>>> L1.append(L2)
>>> L1
[1, 2, 3, [4, 5, 6]]
```

4: extend()

extend() is a method of list, which is used to merge two lists.

extend() iterates over its argument and adds each element at the end of the list.

list.extend(iterable)

Argument of extend() method is any iterable (list, string, set, tuple etc.)

The length of the list increases by a number of elements in its argument.

The method doesn't return any value

```
>>> L1=[1,2,3]
>>> L2=[4,5,6]
>>> L1.extend(L2)
>>> L1
[1, 2, 3, 4, 5, 6]
```

A string is iterable, so if you extend a list with a string, you'll append each character as you iterate over the string.

```
>>> L1=[1,2,3]
>>> S="Python"
>>> L1.extend(S)
>>> L1
[1, 2, 3, 'P', 'y', 't', 'h', 'o', 'n']
```

5: insert()

append() and extend() method is used to add an element and elements of an iterable object respectively at the end of the list. If we want to add an element at any index then we can use insert() method of list.

Syntax: **list_name.insert(index, element)**

```
>>> L=[1,2,3]
Insert element 8 at index 1 i.e. at 2nd position
>>> L.insert(1,8)
>>> L
[1, 8, 2, 3]
```

6: count()

count() method returns how many times a given element occurs in a List i.e. it returns the frequency of an element in the list.

Syntax: **list_name.count(element)**

```
>>> L = [1, 2, 8, 9, 2, 6, 2]
>>> L
[1, 2, 8, 9, 2, 6, 2]
>>> L.count(2)
3
```

7: index()

index() is a method of the list, which returns the index of the first occurrence of the element.

Syntax: **list_name.index(element, start, end)**

element – The element whose index for first occurrence is to be returned.

start (Optional) – The index from where the search begins. Start is included.

end (Optional) – The index from where the search ends. End is excluded.

```
>>> L = [5, 8, 9, 6, 1, 8]
>>> L.index(8)
1
>>> L.index(8,2)
5
>>> L.index(8,2,6)
5
>>> L.index(8,2,5)
ValueError: 8 is not in list
>>> L.index(8,2,7)
5
```

8: remove()

remove() method of list is used to delete the first occurrence of the given element. If the given element is not exist in the list then it will give the ValueError

```
Syntax: list_name.remove(element)
>>> L
[5, 8, 9, 6, 1, 8]
>>> L.remove(8)
>>> L
[5, 9, 6, 1, 8]
>>> L.remove(4)
ValueError: list.remove(x): x not in list
```

9: pop()

Syntax: **list_name.pop(index)**

Index is optional. If index is not given, pop() method of list is used to remove and return the last element of the list, otherwise remove and return the element of the given index. If Index is out of range then it gives IndexError.

```
>>> L = [5, 8, 9, 6, 1]
>>> L
[5, 8, 9, 6, 1]
#remove the last element from the list
>>> L.pop()
```

1
>>> L
[5, 8, 9, 6]
#remove the element from the list whose index is 1
>>> L.pop(1)
8
>>> L
[5, 9, 6]
>>> L.pop(4)
IndexError: pop index out of range

10: reverse()
reverse() method of the list used to reverse the order of the elements of the list

Syntax: **list_name.reverse()**
>>> L = [5, 8, 9, 6, 1]
>>> L
[5, 8, 9, 6, 1]
>>> L.reverse()
>>> L
[1, 6, 9, 8, 5]

11: sort()
sort() method can be used to sort List in ascending, descending, or user-defined order. It sort the existing list.

Syntax: **List_name.sort(reverse=True/False)**
>>> L = [5, 8, 9, 6, 1]
>>> L
[5, 8, 9, 6, 1]
#sort the elements of the list in ascending order
>>> L.sort()
>>> L
[1, 5, 6, 8, 9]
#sort the elements of the list in descending order
>>> L.sort(reverse=True)
>>> L
[9, 8, 6, 5, 1]

12: sorted()
sorted() is a function of list. sorted() function returns a sorted list in ascending order by default. It does not sort or change the existing list.

```
>>> L = [5, 8, 9, 6, 1]
>>> L
[5, 8, 9, 6, 1]
>>> sorted(L)
[1, 5, 6, 8, 9]
>>> L
[5, 8, 9, 6, 1]
#Create a list which contains the element of another list in descending order.
>>> L1=sorted(L, reverse=True)
>>> L1
[9, 8, 6, 5, 1]
```

13: min()

min() function of the list is used to find the minimum element from the list.

```
>>> L = [5, 8, 1, 6, 9]
>>> min(L)
1
# If the elements of the list are string then min() returns the minimum element based on the ASCII values.
>>> L=['a','D','c']
>>> min(L)
'D'
```

14: max()

max() function of the list is used to find the maximum element from the list.

```
>>> L = [5, 8, 1, 6, 9]
>>> max(L)
9
```

If the elements of the list are string then max() returns the maximum element based on the ASCII values.

```
>>> L=['a','D','c']
>>> max(L)
'c'
```

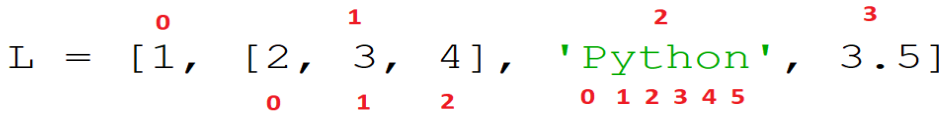
15: sum()

sum() is a function of list, which returns the sum of all elements of the list.

```
>>> L = [5, 8, 9, 6, 1]
>>> sum(L)
29
```

Nested List:

A list within another list is called the nested list i.e. list of list.




```
>>> L = [1, [2, 3, 4], 'Python', 3.5]
>>> L[1]
[2, 3, 4]
>>> L[1][0]
2
>>> L[2]
'Python'
>>> L[2][2]
't'
```

Suggested Programs:

Program-1 : Write a program in Python to find the maximum and minimum number from the given List.

```
L=[]
c='y'
while c=='y' or c=='Y':
     a=int(input("Enter an integer number to append in the list: "))
     L.append(a)
     c=input("Do you want to add more elements in the list (Y/N): ")
print("List is: \n", L)
print("Minimum(Smallest) number of the list = ",min(L))
print("Maximum(Largest) number of the list = ",max(L))
```

Output:

Enter an integer number to append in the list: 5
Do you want to add more elements in the list (Y/N): Y
Enter an integer number to append in the list: 8
Do you want to add more elements in the list (Y/N): Y

Enter an integer number to append in the list: 1
Do you want to add more elements in the list (Y/N): Y
Enter an integer number to append in the list: 9
Do you want to add more elements in the list (Y/N): Y
Enter an integer number to append in the list: 6
Do you want to add more elements in the list (Y/N): N
List is:
[5, 8, 1, 9, 6]
Minimum (Smallest) number of the list = 1
Maximum (Largest) number of the list = 9

Program-2 : Write a program in Python to find the mean of numeric values stored in the given List.

```
L=eval(input("Enter a list: "))
print("List is: \n", L)
NL=[ ]
for i in L:
     if isinstance(i, (int, float)) ==True:
                NL.append(i)
print("Numeric List is: \n", NL)
mean=sum(NL)/len(NL)
print("Mean of the numeric values of the list = ",mean)
```

Output:
Enter a list: [5, 8, 'hello', (1,2), [7,8], 9.5, 10]
List is:
[5, 8, 'hello', (1, 2), [7, 8], 9.5, 10]
Numeric List is:
[5, 8, 9.5, 10]
Mean of the numeric values of the list = 8.125

Program-3 : Write a program in Python to find the index of the given number from a List using linear search.

```
L= [10, 20, 30, 40, 50]
print("List is : ",L)
c='Y'
while c=='Y' or c=='y':
     n=int(input("Enter the number to be search in the list: "))
     found=False
     for i in L:
```

```
    if i==n:
            print("Element found at index: ",L.index(i))
            found=True
    if found==False:
    print("Number is not found in the list")
    c=input("Do you want to search more item (Y/N): ")
```

Output:

List is : [10, 20, 30, 40, 50]
Enter the number to be search in the list: 20
Element found at index: 1
Do you want to search more item (Y/N): y
Enter the number to be search in the list: 50
Element found at index: 4
Do you want to search more item (Y/N): y
Enter the number to be search in the list: 15
Number is not found in the list
Do you want to search more item (Y/N): n

Program-4 : Write a program in Python to count the frequency of all elements of a list.

```
L= [10, 20, 30, 20, 40, 30, 20, 50]
print("List is: ",L)
Uni= []
Freq= []
for i in L:
    if i in Uni:
    index=Uni.index(i)
    Freq[index]+= 1
    else:
    Uni.append(i)
    Freq.append(1)
for i in range(len(Uni)):
    print(Uni[i],":",Freq[i])
```

Output:

List is: [10, 20, 30, 20, 40, 30, 20, 50]
10 : 1
20 : 3
30 : 2
40 : 1
50 : 1