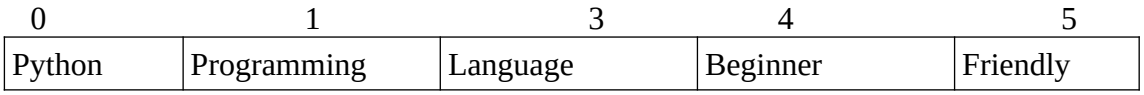
List Data Type:

- If we want to represent a group of individual objects as a single entity where insertion order preserved and duplicates are allowed, then we should go for List.
- insertion order preserved.
- duplicate objects are allowed.
- heterogeneous objects are allowed.
- List is dynamic because based on our requirement we can increase the size and decrease
- the size.
- In List the elements will be placed within square brackets and with comma seperator.
- Python supports both positive and negative indexes. +ve index means from left to right where as negative index means right to left.




```
myList=["Python","Programming","Language","Beginner","Friendly"]
print(myList[1])    #Programming
print(myList[-3])   #Language
```

- List objects are mutable i.e., we can change the content

Creation of List Objects:

1. We can create empty list object as follows...

```
list=[]
print(list)
print(type(list))
```

Output:

```
[]
<class 'list'>
```

2. If we know elements already then we can create list as follows

```
list=[10,20,30,40]
```

3. With dynamic input:(Taking input in the form of list)

```
list=eval(input("Enter List:"))
print(list)
print(type(list))
```

Output:

```
Enter List:[10,20,30,40]
[10, 20, 30, 40]
<class 'list'>
```

4. With list() function:

```
l=list(range(0,10,2))
print(l)
print(type(l))
```

Output:

```
[0, 2, 4, 6, 8]
<class 'list'>
```

```
#Type Casting String to List Data Type
s="piyush"
l=list(s)
print(l)
```

Output:

```
['d', 'u', 'r', 'g', 'a']
```

5. with split() function:

```
s="Learning Python is very very easy !!!"
l=s.split()
print(l)
print(type(l))
```

Output:

```
['Learning', 'Python', 'is', 'very', 'very', 'easy', '!!!']
<class 'list'>
```

Note:

Sometimes we can take list inside another list,such type of lists are called nested lists.

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

Accessing elements of List:

We can access elements of the list either by using index or by using slice operator(:)

1. By using index:
   - List follows zero based index. ie index of first element is zero.
   - List supports both +ve and -ve indexes.
   - +ve index meant for Left to Right
   - -ve index meant for Right to Left

list=[10,20,30,40]

0	1	2	3
10	20	30	40
-4	-3	-2	-1

print(list[0]) ==>10
print(list[-1]) ==>40
print(list[10]) ==>Index Error: list index out of range

2. By using slice operator:

Syntax:

list2= list1[start:stop:step]

start ==>it indicates the index where slice has to start

default value is 0

stop ===>It indicates the index where slice has to end

default value is max allowed index of list ie length of the list

step ==>increment value

default value is 1

Eg:

```
n=[1,2,3,4,5,6,7,8,9,10]
print(n[2:7:2])
print(n[4::2])
print(n[3:7])
print(n[8:2:-2])
print(n[4:100])
```

Output:

```
[3, 5, 7]
[5, 7, 9]
[4, 5, 6, 7]
[9, 7, 5]
[5, 6, 7, 8, 9, 10]
```

List vs mutability:

Once we creates a List object,we can modify its content. Hence List objects are mutable.

Eg:

```
n=[10,20,30,40]
print(n)
n[1]=777
print(n)
```

Output:

```
[10, 20, 30, 40]
[10, 777, 30, 40]
```

Traversing the elements of List:

The sequential access of each element in the list is called traversal.

1. By using while loop:

```
n=[0,1,2,3,4,5,6,7,8,9,10]
i=0
while i<len(n):
        print(n[i])
        i=i+1
```

Output:

```
0
1
2
3
4
5
6
7
8
9
10
```

2. By using for loop:

```
n=[0,1,2,3,4,5,6,7,8,9,10]
for n1 in n:
   print(n1)
```

Output:

```
0
1
2
3
4
5
6
7
8
9
10
```

3. To display only even numbers:

```
n=[0,1,2,3,4,5,6,7,8,9,10]
for n1 in n:
        if n1%2==0:
                print(n1)
```

Output:

```
 0
 2
 4
 6
 8
10
```

4. To display elements by index wise:

```
l=["A","B","C"]
x=len(l)
for i in range(x):
        print(l[i],"is available at positive index: ",i,"and at negative index: ",i-x)
```

Output

```
A is available at positive index: 0 and at negative index: -3
B is available at positive index: 1 and at negative index: -2
C is available at positive index: 2 and at negative index: -1
```

<u>Important functions of List:</u>

2). Manipulating elements of List:

1. append() function:

We can use append() function to add item at the end of the list.

Eg:

```
list=[]
list.append("A")
list.append("B")
list.append("C")
print(list)
```

output

```
['A', 'B', 'C']
```

Eg: To add all elements to list upto 100 which are divisible by 10

```
list=[]
for i in range(101):
        if i%10==0:
                list.append(i)
print(list)
```

output:

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

2. insert() function:

To insert item at specified index position

```
n=[1,2,3,4,5]
n.insert(1,888)
print(n)
```

Output:

```
[1, 888, 2, 3, 4, 5]
```

Eg:

```
n=[1,2,3,4,5]
n.insert(10,777)
n.insert(-10,999)
print(n)
```

Output:

```
[999, 1, 2, 3, 4, 5, 777]
```

Note: If the specified index is greater than max index then element will be inserted at last position. If the specified index is smaller than min index then element will be inserted at first position.

Differences between append() and insert()

append()	insert()
In List when we add any element it will come in last i.e. it will be last element.	In List we can insert any element in particular index number.

3. extend() function:
To add all items of one list to another list
l1.extend(l2)
all items present in l2 will be added to l1
Eg:

```
order1=["Chicken","Mutton","Fish"]
order2=["RC","KF","FO"]
order1.extend(order2)
print(order1)
```

```
D:\Python_classes>py test.py
['Chicken', 'Mutton', 'Fish', 'RC', 'KF', 'FO']
```

Eg:

```
1) order=["Chicken","Mutton","Fish"]
order.extend("Mushroom")
print(order)
```

```
D:\Python_classes>py test.py
['Chicken', 'Mutton', 'Fish', 'M', 'u', 's', 'h', 'r', 'o', 'o', 'm']
```

4. remove() function:
We can use this function to remove specified item from the list.If the item present multiple times then **only first occurrence** will be removed.

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

```
D:\Python_classes>py test.py
[20, 10, 30]
```

If the specified item not present in list then we will get ValueError

```
n=[10,20,10,30]
n.remove(40)
print(n)
```

ValueError: list.remove(x): x not in list
Note: Hence before using remove() method first we have to check specified element
present in the list or not by using in operator.

5. pop() function:
It removes and returns the last element of the list.
This is only function which manipulates list and returns some element.
Eg:

```
n=[10,20,30,40]
print(n.pop())
print(n.pop())
print(n)
```

```
D:\Python_classes>py test.py
40
30
[10, 20]
```

If the list is empty then pop() function raises IndexError
Note:
n.pop(index)===>To remove and return element present at specified index.
n.pop()==>To remove and return last element of the list

```
n=[10,20,30,40,50,60]
print(n.pop()) #60
print(n.pop(1)) #20
print(n.pop(10)) ==>IndexError: pop index out of range
```

Difference between remove and pop list method?

remove()	**pop()**
1) We can use to **remove special element** from the List.	1) We can use to **remove last element** from the List /or from **any specified index**
2) It can't return any value.	2) It return the popped value.
3) If special element not available then we get **VALUE ERROR.**	3) If List is empty then we get **Index Error.**

Note:
List objects are **dynamic**. i.e based on our requirement we can increase and decrease the
size.
append(),insert() ,extend() ===>for increasing the size/growable nature
remove(), pop() ======>for decreasing the size /shrinking nature

III. Ordering elements of List:

1. reverse():

We can use to reverse() order of elements of list.

```
n=[10,20,30,40]
n.reverse()
print(n)

D:\Python_classes>py test.py
[40, 30, 20, 10]
```

2. sort() function:

In list by default insertion order is preserved. If want to sort the elements of list according
to default natural sorting order then we should go for sort() method.

For numbers ==>default natural sorting order is Ascending Order

For Strings ==> default natural sorting order is Alphabetical Order/Chronological order.

```
n=[20,5,15,10,0]
n.sort()
print(n) #[0,5,10,15,20]

s=["Dog","Banana","Cat","Apple"]
s.sort()
print(s) #['Apple','Banana','Cat','Dog']
```

Note: To use sort() function, compulsory list should contain only homogeneous elements.

otherwise we will get TypeError

Eg:

```
n=[20,10,"A","B"]
n.sort()
print(n)
```

TypeError: '<' not supported between instances of 'str' and 'int'

To sort in reverse of default natural sorting order:

We can sort according to reverse of default natural sorting order by using **reverse=True** argument.

Eg:

```
n=[40,10,30,20]
n.sort()
print(n) ==>[10,20,30,40]
n.sort(reverse=True)
print(n) ===>[40,30,20,10]
```

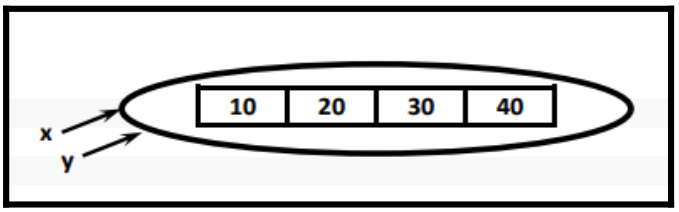
n.sort(reverse=False)
print(n) ==>[10,20,30,40]

Aliasing and Cloning of List objects

The process of giving another reference variable to the existing list is called aliasing.

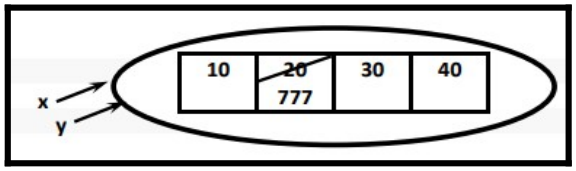
Eg:

1) x=[10,20,30,40]
2) y=x
3) print(id(x))
4) print(id(y))



The problem in this approach is by using one reference variable if we are changing content,then those changes will be reflected to the other reference variable.

1) x=[10,20,30,40]
2) y=x
3) y[1]=777
4) print(x) ==>[10,777,30,40]

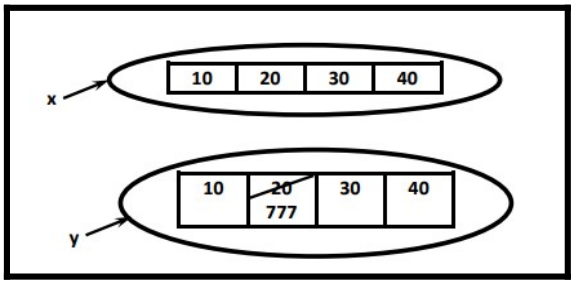


To overcome this problem we should go for cloning.

The process of creating exactly duplicate independent object is called cloning.
We can implement cloning by using slice operator or by using copy() function

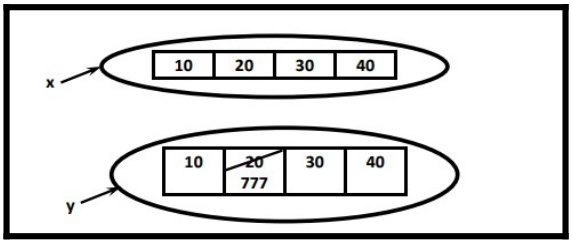
1. By using slice operator:

1) x=[10,20,30,40]
2) y=x[:]
3) y[1]=777
4) print(x) ==>[10,20,30,40]
5) print(y) ==>[10,777,30,40]



2. By using copy() function:

1) x=[10,20,30,40]
2) y=x.copy()
3) y[1]=777
4) print(x) ==>[10,20,30,40]
5) print(y) ==>[10,777,30,40]



Q. Difference between = operator and copy() function

= operator meant for aliasing
copy() function meant for cloning

Using Mathematical operators for List Objects:

We can use + and * operators for List objects.

1. Concatenation operator(+):
We can use + to concatenate 2 lists into a single list
1) a=[10,20,30]
2) b=[40,50,60]
3) c=a+b
4) print(c) ==>[10,20,30,40,50,60]
Note: To use + operator compulsory both arguments should be list objects,otherwise we
will get TypeError.
Eg:
c=a+40 ==>TypeError: can only concatenate list (not "int") to list
c=a+[40] ==>valid

2. Repetition Operator(*):
We can use repetition operator * to repeat elements of list specified number of times
Eg:
1) x=[10,20,30]
2) y=x*3
3) print(y)==>[10,20,30,10,20,30,10,20,30]

Comparing List objects
We can use comparison operators for List objects.
Eg:
1. x=["Dog","Cat","Rat"]
2. y=["Dog","Cat","Rat"]
3. z=["DOG","CAT","RAT"]
4. print(x==y) True
5. print(x==z) False
6. print(x != z) True

Note:
Whenever we are using comparison operators(==,!=) for List objects then the following
should be considered
1. The number of elements
2. The order of elements
3. The content of elements (case sensitive)

Note: When ever we are using relatational operators(<,<=,>,>=) between List objects,only
first element comparison will be performed.
Eg:

```
1. x=[50,20,30]
2. y=[40,50,60,100,200]
3. print(x>y) True
4. print(x>=y) True
5. print(x<y) False
6. print(x<=y) False
```

Eg:

```
1. x=["Dog","Cat","Rat"]
2. y=["Rat","Cat","Dog"]
3. print(x>y) False
4. print(x>=y) False
5. print(x<y) True
6. print(x<=y) True
```

Membership operators:
We can check whether element is a member of the list or not by using memebership operators.
in operator
not in operator
Eg:

```
n=[10,20,30,40]
print (10 in n)
print (10 not in n)
print (50 in n)
print (50 not in n)
```

Output

```
True
False
False
True
```

clear() function:
We can use clear() function to remove all elements of List.
Eg:

```
n=[10,20,30,40]
print(n)
```

```
n.clear()
print(n)
```

Output
```
[10, 20, 30, 40]
[]
```

Nested Lists:
Sometimes we can take one list inside another list. Such type of lists are called nested lists.
Eg:
```
n=[10,20,[30,40]]
print(n)
print(n[0])
print(n[2])
print(n[2][0])
print(n[2][1])
```

Output

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

List Comprehensions:
It is very easy and compact way of creating list objects from any iterable objects(like list,tuple,dictionary,range etc) based on some condition.
Syntax:
```
list=[expression for item in list if condition]
```
Eg:
```
s=[ x*x for x in range(1,11)]
print(s)
v=[2**x for x in range(1,6)]
print(v)
m=[x for x in s if x%2==0]
print(m)
```

Output
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[2, 4, 8, 16, 32]
[4, 16, 36, 64, 100]
```
Eg:

```
words=["Balaiah","Nag","Venkatesh","Chiranjeevi"]
l=[w[0] for w in words]
print(l)
```