Learn Complete Python In Simple Way

LIST

DATA STRUCTURE STUDY MATERIAL

- 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.
- We can differentiate duplicate elements by using index and we can preserve insertion order by using index. Hence index will play very important role.
- Python supports both positive and negative indexes. +ve index means from left to right where as negative index means right to left.

[10,"A","B",20, 30, 10]

-6	-5	-4	-3	-2	-1
10	A	B	20	30	10
0	1	2	3	4	5

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

Creation of List Objects:

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

```
1) list=[]
2) print(list)
3) print(type(list))
4)
5) []
6) <class 'list'>
```

2) If we know elements already then we can create list as follows list = [10, 20, 30, 40]

3) With Dynamic Input:

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

```
D:\Python_classes>py test.py
Enter List:[10,20,30,40]
[10, 20, 30, 40]
<class 'list'>
```

4) With list() Function:

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

```
D:\Python_classes>py test.py
[0, 2, 4, 6, 8]
<class 'list'>
```

Eg:

```
1) s="durga"
2) l=list(s)
3) print(l)
```

```
D:\Python_classes>py test.py
['d', 'u', 'r', 'g', 'a']
```

5) With split() Function:

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

```
D:\Python_classes>py test.py
['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.

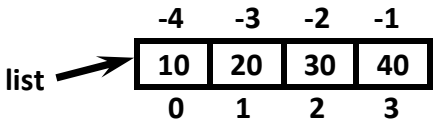
```
[10, 20, [30, 40]]
```

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]



- print(list[0]) → 10
- print(list[-1]) → 40
- print(list[10]) → IndexError: 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

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

Output

```
D:\Python_classes>py test.py
[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.

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

```
D:\Python_classes>py test.py
[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:

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

```
D:\Python_classes>py test.py
0
1
2
3
4
5
6
7
8
9
10
```

2) By using for Loop:

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

```
D:\Python_classes>py test.py
0
1
2
3
4
5
6
7
8
9
10
```

3) To display only Even Numbers:

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

```
D:\Python_classes>py test.py
0
2
4
6
8
10
```

4) To display Elements by Index wise:

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

```
D:\Python_classes>py test.py
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
```

Important Functions of List:

I. To get Information about List:

1) len():

Returns the number of elements present in the list

Eg: n = [10, 20, 30, 40]

print(len(n) → 4

2) count():

It returns the number of occurrences of specified item in the list

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

```
D:\Python_classes>py test.py
1
4
2
0
```

3) index():

Returns the index of first occurrence of the specified item.

```
1) n = [1, 2, 2, 2, 2, 3, 3]
2) print(n.index(1)) → 0
3) print(n.index(2)) → 1
4) print(n.index(3)) → 5
5) print(n.index(4)) → ValueError: 4 is not in list
```

Note: If the specified element not present in the list then we will get ValueError.Hence before index() method we have to check whether item present in the list or not by using in operator.

print(4 in n) → False

II. Manipulating Elements of List:

1) append() Function:

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

```
1) list=[]
2) list.append("A")
3) list.append("B")
4) list.append("C")
5) print(list)
```

```
D:\Python_classes>py test.py
['A', 'B', 'C']
```

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

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

```
D:\Python_classes>py test.py
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

2) insert() Function:

To insert item at specified index position

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

```
D:\Python_classes>py test.py
[1, 888, 2, 3, 4, 5]
```

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

```
D:\Python_classes>py test.py
[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

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

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

```
1) order = ["Chicken","Mutton","Fish"]
2) order.extend("Mushroom")
3) 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.

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

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

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

```
1) n=[10,20,10,30]
2) n.remove(40)
3) 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.

```
1) n=[10,20,30,40]
2) print(n.pop())
3) print(n.pop())
4) print(n)
```

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

If the list is empty then pop() function raises IndexError

```
1) n = []
2) print(n.pop()) → IndexError: pop from empty list
```

Note:

1) pop() is the only function which manipulates the list and returns some value
2) In general we can use append() and pop() functions to implement stack datastructure by using list,which follows LIFO(Last In First Out) order.

In general we can use pop() function to remove last element of the list. But we can use to remove elements based on index.

n.pop(index) → To remove and return element present at specified index.
n.pop() → To remove and return last element of the list

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

Differences between remove() and pop()

remove()	pop()
1) We can use to remove special element from the List.	1) We can use to remove last element from the List.
2) It can't return any value.	2) It returned removed element.
3) If special element not available then we get VALUE ERROR.	3) If List is empty then we get 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.

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

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

2) sort():

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

```
1) n = [20,5,15,10,0]
2) n.sort()
3) print(n) → [0,5,10,15,20]
4)
5) s = ["Dog","Banana","Cat","Apple"]
6) s.sort()
7) print(s) → ['Apple','Banana','Cat','Dog']
```

Note: To use sort() function, compulsory list should contain only homogeneous elements. Otherwise we will get TypeError

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

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

Note: In Python 2 if List contains both numbers and Strings then sort() function first sort numbers followed by strings

```
1) n=[20,"B",10,"A"]
2) n.sort()
3) print(n)# [10,20,'A','B']
```

But in Python 3 it is invalid.

To Sort in Reverse of Default Natural Sorting Order:

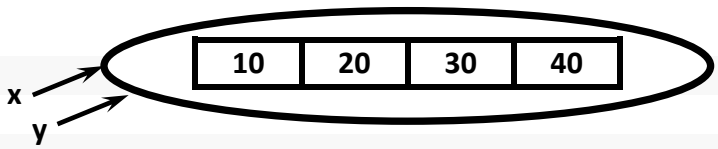
We can sort according to reverse of default natural sorting order by using reverse=True argument.

```
1) n = [40,10,30,20]
2) n.sort()
3) print(n) → [10,20,30,40]
4) n.sort(reverse = True)
5) print(n) → [40,30,20,10]
6) n.sort(reverse = False)
7) print(n) → [10,20,30,40]
```

Aliasing and Cloning of List Objects:

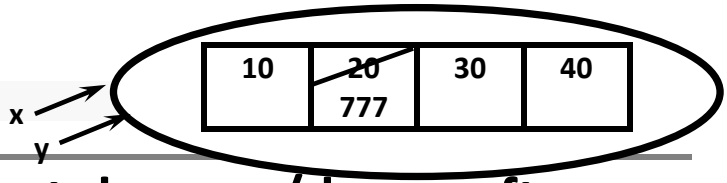
The process of giving another reference variable to the existing list is called aliasing.

```
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
```



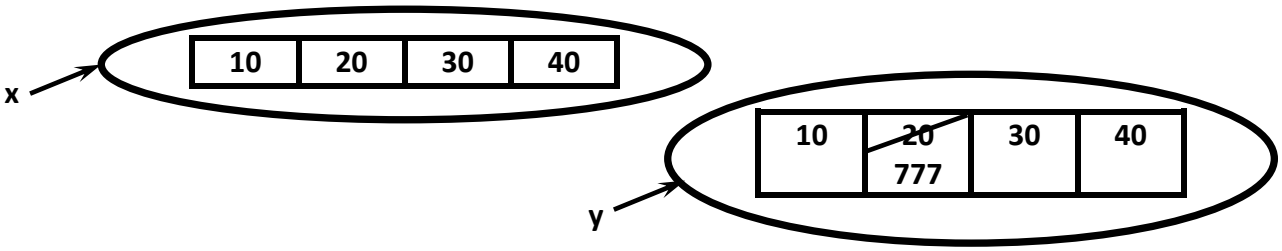
```
3) y[1] = 777
4) print(x) → [10,777,30,40]
```

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.

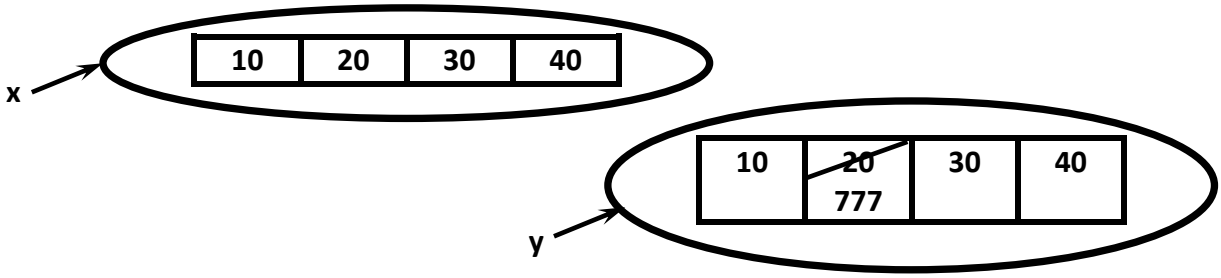
1) By using Slice Operator:

```
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.

```
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.

```
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 1ST Element comparison will be performed.

```
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.

1) in Operator
2) not in Operator

```
1) n=[10,20,30,40]
2) print (10 in n)
3) print (10 not in n)
4) print (50 in n)
5) print (50 not in n)
```

Output

```
True
False
False
True
```

clear() Function:

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

```
1) n=[10,20,30,40]
2) print(n)
3) n.clear()
4) print(n)
```

Output

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

Nested Lists:

Sometimes we can take one list inside another list. Such type of lists are called nested lists.

```
1)  n=[10,20,[30,40]]
2)  print(n)
3)  print(n[0])
4)  print(n[2])
5)  print(n[2][0])
6)  print(n[2][1])
```

Output

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

Note: We can access nested list elements by using index just like accessing multi dimensional array elements.

Nested List as Matrix:

In Python we can represent matrix by using nested lists.

```
1)  n=[[10,20,30],[40,50,60],[70,80,90]]
2)  print(n)
3)  print("Elements by Row wise:")
4)  for r in n:
5)      print(r)
6)  print("Elements by Matrix style:")
7)  for i in range(len(n)):
8)      for j in range(len(n[i])):
9)          print(n[i][j],end=' ')
10)     print()
```

Output

```
D:\Python_classes>py test.py
[[10, 20, 30], [40, 50, 60], [70, 80, 90]]

Elements by Row wise:
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]

Elements by Matrix style:
10 20 30
40 50 60
70 80 90
```

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]

```
1) s = [ x*x for x in range(1,11)]
2) print(s)
3) v = [2**x  for x in range(1,6)]
4) print(v)
5) m = [x for x in s if x%2==0]
6) print(m)
```

```
D:\Python_classes>py test.py
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[2, 4, 8, 16, 32]
[4, 16, 36, 64, 100]
```

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

Output: ['B', 'N', 'V', 'C']

```
1) num1=[10,20,30,40]
2) num2=[30,40,50,60]
3) num3=[ i for i in num1 if i not in num2]
4) print(num3)   [10,20]
5)
6) common elements present in num1 and num2
```

```
7)  num4=[i for i in num1 if i in num2]
8)  print(num4)   [30, 40]
```

Eg:

```
1)  words="the quick brown fox jumps over the lazy dog".split()
2)  print(words)
3)  l=[[w.upper(),len(w)]     for w in words]
4)  print(l)
```

Output

```
['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
[['THE', 3], ['QUICK', 5], ['BROWN', 5], ['FOX', 3], ['JUMPS', 5], ['OVER', 4],
['THE', 3], ['LAZY', 4], ['DOG', 3]]
```

Q) Write a Program to display Unique Vowels present in the given Word?

```
1)  vowels=['a','e','i','o','u']
2)  word=input("Enter the word to search for vowels: ")
3)  found=[]
4)  for letter in word:
5)      if letter in vowels:
6)          if letter not in found:
7)              found.append(letter)
8)  print(found)
9)  print("The number of different vowels present in",word,"is",len(found))
```

```
D:\Python_classes>py test.py
Enter the word to search for vowels: durgasoftwaresolutions
['u', 'a', 'o', 'e', 'i']
The number of different vowels present in durgasoftwaresolutions is 5
```

List out all Functions of List and write a Program to use these Functions