Class Notes

Prepared by **VIJITH P**

Class: **XI - B Subjects**: **COMPUTER SCIENCE**

Chapter/Unit: **Unit 7: LISTS IN PYTHON**

Page/s From **2** to **16**

SILVER HILLS PUBLIC SCHOOL

Affiliated to CBSE New Delhi, No. 930433, Code No. 75439

Paroppady, Marikunnu PO, Kozhikode - 673012 Kerala, India

Phone: 0495 2370075, 2375144 Fax: 0495 2375155

E-mail: shpsoffice@gmail.com Web: www.silverhillspublicschool.org

a cmi institution • nurturing a youth sublime

LIST IN PYTHON

A list is defined as the ordered sequence of data (may be same or different types) separated by commas and enclosed with in opening and closing square brackets. [ ]

One of the basic advantages of list over other collection of data (such as string, tuple) is that the list is mutable collection of data. It means that any change or alternation in data is maintained in the same address for its storage. Strings and tuples are immutable in nature. Means, the updated collection of data will occupy separate location in memory for its storage.

Creating and displaying list

```
>>>[]
```

[] – It is an empty or null list.

```
>>>[2,4,6,8,10]
```

[2,4,6,8,10] – list contains sequence of even integers up to 10.

```
>>> ['a','b','c','d']
```

['a', 'b', 'c', 'd']- list contains the letters ('a' to 'd')

```
>>> ["The",'cat',"Is","Black"]
```

['The', 'cat', 'Is', 'Black'] – list containing a set of four strings.

```
>>> [24,"x",11.6,'python']
```

[24, 'x', 11.6, 'python'] – list includes a set of elements with different data types.

Assigning a list

Assigning a list means naming a sequence of data with the help of a variable.

- A list can be assigned to a variable by using the following syntax:
  Variable=[value1,value2,value3,…….]
  Eg: list1=[1,3,5,7,9]

- An empty list can be assigned to a variable by using list() function as shown below:
  List1=list()
- In case, a list is too large to fit on a single line, it can be split up into multiple lines.
  ```
  List1=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,
            17,18,19,20]
  ```
- Python allows assigning multiple lists to different variables in a single statement as shown below:
  Syntax: variable1,variable2=list1,list2
  Eg: L1,L2=[1,2,3], ['a', 'b', 'c']
  Here, L1 and L2 are assigned the lists [1,2,3] and ['a', 'b', 'c'] respectively.

Accessing List Elements

A list consists of any collection of items which are stored according to its index.

Creating a List from an existing List

Consider a list

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

list5=list1[:]

list5 will create a copy of list1.

list6=list1[1:4]

list6 shall contain elements at index 1,2,3 from list1.

list6=[20,30,40]

list7=list1

list7 have all elements present in list1

list7=[10,20,30,40,50,]

Accessing List Elements. (Indexing)

A list consists of any collection of items which are stored according to its index. Like strings, any item in a list can be accessed by using the slice operator [] by specifying its index value. List indices start with 0 and go to length-1.

Eg: >>>list=[3,4,1,2]

>>>list[2] will access the element 1

Forward Indexing	0	1	2	3	
	3	4	1	2	
	-4	-3	-2	-1	Backward Indexing

An IndexError appears if you try and access an element that does not exist in the list.

Traversing a List

Traversing a list means accessing each element of a list. It can be done in the following three ways:

- Using *in* operator inside *for* loop

  In this, a *for* loop is used to access the list elements one by one from the beginning to the end with the help of *in* operator.

  Syntax:

  ```
  List variable=[elements]
  for<control variable> in <List variable>:
        Print(control variable)
  ```

  Eg:

  ```
  Lst=['a', 'b', 'c']

  for x in Lst:

  print(x)

  o/p: a

        b

        c
  ```

- Using range() function

  In this system, a *range* of indices are used in the *for* loop.

  Syntax:

  ```
  List variable=[elements]
  for index in range(initial,(length of the list)):
  print(List variable[index])
  ```

  Eg:

  ```
  Lst=[2,4,6,8]
  for i in range(len(Lst)):
          print(Lst[i])
  ```

  o/p:

  ```
  2
  4
  6
  8
  ```

- Using while loop

  In while loop, initially the len() function is used to determine the length of the list, then while loop starts from index 0 and iterated through the list items by referring to their indexes. Unlike a for loop, it is must tom increase the index by 1 after each iteration.

  Eg:

  ```
  Mylist=['welcome', 'to python','language']
  index=0
  while index<len(Mylist):
          print(Mylist[index])
          index+=1
  ```

  o/p:

  ```
  welcome
  to python
  language
  ```

ALIASING

In python, variables refer to objects. The list we create in python are also objects. Since list are mutable, if we assign the elements of one list to another list, both shall refer to the same object.

```
>>>subjects=[‘Hindi’,’English’,’Maths’,’History’]
>>>temporary=subjects
>>>temporary[0]=’sanskrit’
>>>print(temporary)
[‘Sanskrit’,’English’,’Maths’,’History’]
>>>print(subjects)
[‘Sanskrit’,’English’,’Maths’,’History’]
```

The above list can be modified with two values at one index position.

```
>>>subjectcodes=[[‘Sanskrit’,43],[‘English’,85],[‘Maths’,65
],[‘History’,36]]
>>>subjectcodes[1]
[‘English’,85]
>>>print(subjectcodes[1][0],subjectcodes[1][1])
[‘English’,85]
```

COMPARING LISTS

Python allows us to compare two lists. Each element is individually compared in lexicographical order.

```
>>>L1,L2=[10,20,30],[10,20,30]

>>>L3=[10,[20,30]]

>>>L1==L2

>>>True

>>>L1==L3

False
```

Nested Lists

When a list appears as an element of another list, it is called a nested list. A list can have one or more lists as an element.

To access the element of nested list, we have to specify two indices.

Eg: list1[i][j]

The first index i will take us to the desired nested list and the second index j will take us to the desired element in that nested list.

Eg:

list1=[1,2,'a','c',[6,7,8],4,9]

list1[4]

[6, 7, 8]

list1[4][1]

7

list2=[1,2,3,'a',['apple','green']]

list2[4][0][1]

'p'

Copying Lists

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

Eg: list1=[1,2,3,4]

    list2=list1

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

We can also create a copy or clone of the list as distinct object by three methods.

- The first method uses the slicing
- Second method uses built in function list()
- 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]
>>>list2=list1[:]
>>>list2 –> [1,2,3]
```

Method 2:

We can use list constructor function list() as follows:

newlist=list(oldlist)

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

>>>list2=list(list1)

>>>list2   -  [10,20,30,40]
```

Method 3:

We can use the copy() function as follows:

newlist=list(oldlist).copy()

```
>>>list1=[1,2,3,4]

>>>list2=list1.copy()

>>>list2 - [1,2,3,4]
```

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.

BUILT IN FUNCTIONS (MANIPULATING LISTS)

Python offers several built in 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.

append()

The append() method adds 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.

Syntax:

List.append(item)

```
Eg:1>>>l1=[10,20,30]
    >>>l1.append(55)
    >>>l1
    >>>[10,20,30,55]
    Eg:2>>>l1=[10,20,30,40]
        >>>l1.append([50,60])
        >>>l1
        >>>[10,20,30,40,[50,60]]
    Eg3>>>colour=['red','green','blue']
        >>>colour.append('white')
        >>>colour
        >>>=['red','green','blue','white']
```

extend()

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

Syntax:

list1.extend(list2)

- list1 is the primary list which will be extended.
- list2 is the list will be added to list1.

Example:

```
>>>l1=[100,200,300,400]

>>>l2=[10,20,30]

>>>l1.extend(l2)

>>>l1 - [100,200,300,400,10,20,30]
```

insert()

The insert() function can be used to insert an element/object at 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 a list

index_number is the index where the new value is to be inserted

values the new value to be inserted in the list.

```
Eg:  >>>l1=[10,20,30]

     >>>l1.insert(0,5)

     >>>l1

     >>>[5,10,20,30]
```

reverse() function

The reverse function in python reverses the order of the elements in a list. This function reverses the item of a list. It replaces a new value “in place” of an item that already exists in the list. It doesn’t create a new list.

Syntax:

List.reverse()

```
Eg:1   >>>list1=[10,20,30,40,50,60]
       >>>list1.reverse()
       >>>list1
       >>>[60,50,40,30,20,10]
Eg2:>>>names=[‘Vibay’,’Sonia’,’Soorya’,’Radhika’]
     >>>names.reverse()
     >>>[’Radhika’,’Soorya’,’Sonia’,‘Vibay’]
```

index() function

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 in the list. If the element is not present, ValueError is generated.

Syntax:

```
      list.index(<item>)
     Eg1)  list1=[10,20,30,40,50, 40]
           list1.index(40)
           3
   Eg2)  x='python'
         x.index(‘t’)
         2
```

```
Eg3) x=['India','my','country']
     x.index('India')
     0
```

sort() function

This function sorts the items of the list by default in ascending / increasing order.

Syntax:

List.sort()

```
Eg1 L1=[10,4,20,98,2,100]
     L1.sort()
      print(L1)
      [2,4,19,20,98,100]
Eg2 L2=['RED','YELLOW','BLUE','WHITE']
     L2.sort()
     print(L2)
     ['BLUE','RED','WHITE','YELLOW']
```

For string elements in a list, sorting is done on the basis of their ASCII values. It sorts string in a lexicographic manner.

If we want to sort the list in descending order, we need to write:

List1.sort(reverse=True)

```
Eg: L1=[10,4,20,98,2,100]
     L1.sort(reverse=True)
     print(L1)
    [100,98,20,10,4,2]
```

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(list)
[1,2,3,4,5,6,7,8,9]
>>>sorted(list1,reverse=True)
[9,8,7,6,5,4,3,2,1]
```

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.

clear()

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

```
>>>l1=[10,20,30,40]
>>>l1.clear()
>>>print(l1)
>>>
```

count()

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

Syntax: list.count(element)

Eg:

```
>>>L1=[1,2,3,2,4,2]
>>>L1.count(2)
>>>3
```

DELETION OPERATION

Python provides operator for deleting/removing an item from a list. There are many methods for deletion:

- pop()- if the index is known you can use pop() method. It removes the element from the specified index and also returns the element which was removed.

  Syntax:
  List.pop(index)
  Eg:
  ```
  >>>L1=[1,5,10,50,60]
  >>>L1.pop(1)
  >>>5
  ```

  If no index is provided in pop() function, then last element from the list is deleted.
  Eg:
  ```
  >>>L1=[1,2,3,4,5]
  >>>L1.pop()
  >>>5
  ```
  pop() can be used with negative index value also.
  ```
  Eg: >>>l1=[1,2,3]
      >>>l1.pop(-1)
       >>>3
  ```
- del statement
  It removes the specified element from the list based on its index value, but does not return the deleted value.
  Eg:
  ```
  >>>L1=[100,200,300,400,500]
  >>>del L1[3]
  >>>print(L1)
  >>>[100,200,300,500] – 400 deleted
  ```

  del can be used with negative index value also.

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

To remove more than one element, del() with list slice can be used.
Eg:

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

Note: if an out of range index is provided with del and pop(), the code will result an IndexError.

- remove () – the remove() function is used when we know the element to be deleted rather than its index.

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

  The above statement will remove 30 from the list.

  ```
  >>>print(l1)
  [10,20]
  ```

  Note: if the given element is not available in the list, it retunes ValueError:

<u>Important functions related to list search</u>

- max()
  It returns the element with maximum value from the list. To use the max() function, all values in the list must be of same type.
  Eg:
  1.

  ```
  >>>l1=[10,54,68,8]
  >>>max(l1)
  >>>68
  ```

2.

```
>>> list=['A','a','B','b']
>>> max(list)
'b'
```

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

3.

```
>>> list1=['ashwin','bharat','shelly']
>>> max(list1)
>>>'shelly'
```

Here, it will return the string which starts with character having the highest ASCII value. If there are two or more strings which start with the same character , then second character compared and so on.

- min()

  Returns the element with the minimum value from the list.

  Eg:

  ```
  >>> list=['A','a','B','b']
  >>> min(list)
  'A'
  ```

****