Lists in Python

A sequence is a data type that represents a group of elements. The purpose of any sequence is to store and process a group of elements. In Python, strings, lists,tuples and dictionaries are very important sequence data types. All sequences allow some common operations like indexing and slicing.

List

A list is similar to an array that consists of a group of elements or items. Just like an array, a list can store elements. But, there is one major difference between an array and a list. An array can store only one type of element whereas a list can store different types of elements. Hence lists are more versatile and useful than an array. Perhaps lists are the most used data type in Python programs.

In our daily life, we do not have elements of the same type. For example, we take marks of a student in 5 subjects:

66, 75, 82, 94, 96

These are all belonging to the same type, i.e. integer type. Hence, we can represent such elements as an array. But, we need other information about the student, like his roll number, name, gender along with his marks. So, the information looks like this:

08, Ram, M, 66, 75, 82, 94, 96

Here, we have different types of data. Roll number (08) is an integer. Name (Ram) is a string Gender (M) is a character and the marks (66, 75, 82, 94, 96) are again integers. In daily life, generally we have this type of information that is to be stored and processed. This type of information cannot be stored in an array because an array can store only one type of element. In this case, we need to go for a list datatype. A list can store different types of elements. To store the student's information discussed so far, we can create a list as:

student=[08, “Ram”, 'M’, 66, 75, 82, 94, 96]

Please observe that the elements of the student' list are stored in square braces []. We can create an empty list without any elements by simply writing empty square braces as e_lst=[ ]

Thus we can create a list by embedding the elements inside a pair of square braces [ ]. The elements in the list should be separated by a comma (,). To view the elements of a list as a

whole, we can simply pass the list name to the print() function as:

```
print(student)
```

The list appears as given below:

```
[08, “Ram”, 'M', 66, 75, 82, 94, 96]
```

Indexing and slicing operations are commonly done on lists. Indexing represents accessing elements by their position numbers in the list. The position numbers start from 0 onwards and are written inside square braces as: student[0], student[1], etc... It means, student[0] represents 0th element, student[1] represents 1st element and so forth. For example, to print the student's name, we can write:

```
print (student[1])
```

The name of student appears as given below:

Ram

Slicing represents extracting a piece of the list by mentioning starting and ending position numbers. The general format of slicing is: [start: stop: stepsize]. By default, ‘start’ will be 0, ‘stop’ will be the last element and ‘stepsize' will be 1. For example, student[0:3:11] represents a piece of the list containing 0 to 2nd elements.

```
print(student [0:3:11])
```

The elements are given below:

```
[08, ‘Ram’, "M"]
```

We can also write the above statement as:

```
print (student[:3:])
```

The same elements appears as shown following

[08, "Ram", "M"]

Here, since we did not mention the starting element position, it will start at 0 and stepize will be taken as 1. Suppose, we do not mention anything in slicing, then the total list will be extracted as:

```
print (student[::])
```

It displays the output as

[08, “Ram”, 'M', 66, 75, 82, 94, 96]

Apart from indexing and slicing, the 5 basic operations: finding length, concatenation. repetition, membership and iteration operations can be performed on lists and other sequences like strings, tuples or dictionaries.

In the following program, we are creating lists with different types of elements and also displaying the list elements.

Ex:

```
num=[10, 20, 30, 40, 50]
print('Total list= ',num)
print('First= %d, Last= %d'%(num[0], num[3]))
names=["Ram","Sita","Laxman"]
print('Total list= ',names)
print('First=%s,Last=%s'% (names[0],names[2]))
x=[10, 20, 10.5, 2.55, "Ganesh", "vishnu"]
print('Total list=',x)
print('First=%d,Last=%s'%(x[0],x[5]))
```

Output:

```
Total list=  [10, 20, 30, 40, 50]
First= 10, Last= 40
Total list=  ['Ram', 'Sita', 'Laxman']
First=Ram,Last=Laxman
Total list= [10, 20, 10.5, 2.55, 'Ganesh', 'vishnu']
First=10,Last=vishnu
```

Creating Lists using range() Function

We can use the range() function to generate a sequence of integers which can be stored in a list. The format of the range() function is:

range (start, stop, stepsize)

If we do not mention the start, it is assumed to be 0 and the ‘stepsize' is taken as 1. The range of numbers stops one element prior to 'stop'. For example,

range(0, 10, 1)

This will generate numbers from 0 to 9, as: 10, 1, 2, 3, 4, 5, 6, 7, 8, 9). Consider another Example:
range (4,9,2)

The preceding statement will generate numbers from 4 to 8 in steps of 2, i.e. [4,6,8]
In fact, the range() function does not return a list of numbers. It returns only a range class object that stores the data about start, 'stop' and ‘stepsize’. For example, if we write

print(range(4,9,2)

The preceding statement will display:

range (4. 9. 2)

This range object should be used in for loop to get the range of numbers desired by the programmer. For example,

```
for i in range (4,9,2):
print(i)
```

The preceding statement will display 4, 6, 8. Hence, we say a range object is iterable, that is, suitable as a target for functions and loops that expect something from which they can obtain successive items. For example, range object can be used in for loops to display the numbers, or with list) function to create a list. In the following example, the range function is used in the list function to create a list:

```
lst=list(range(4, 9, 2))
print(lst)
```

The list is shown below.

[4,6,8]

If we are not using the list() function and using range() alone to create a list, then we will have only a range class object returned by the range() function. For example,

```
lst=range(4, 9, 2)
print (lst)
```

The preceding statements will give the following output:
range(4, 5, 2)

In this case, using a loop like for or while is necessary to view the elements of the list. For example,

```
for i in lst:
print (i)
```

will display 4, 6, 8.
Ex:

```
list1=range (10)
for i in list1:
    print(i,',',end=' ')
print()

list2=range(5,10)
for i in list2:
    print(i,',', end='')
print()

list3=range(5, 10, 2)
for i in list3:
    print(i,',', end='')
```

Output:

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

We can use while loop or for loop to access elements from a list. The len() function is useful to know the number of elements in the list. For example, lenlliati gives tonal number of elements in the list The following while loop retrieves starting from 0 to the last element of the list:

```
i=0
while i<len(list):
print(list[i])
i=i+1
```

Observe the len(list) function in the while condition as: while i<len(list).This will return the total number of elements in the list. If the total number of elements is 'n', then the condition will become while(i-n). It means I values are changing from 0 to n-1.Thus this loop will display all elements of the list from 0 to n-1.

Another way to display elements of a list is by using a for loop, as:

```
for i in list:
print(i)
```

Here, ‘i' will assume one element at a time from the list and hence if we display value, it will display the elements one by one.

Updating the Elements of a List

Lists are mutable. It means we can modify the contents of a list. We can append, update or delete the elements of a list depending upon our requirements. Appending an element means adding an element at the end of the list. To append a new element to the list, we should use the list.append() method. In the following example, we are creating a list with elements from 1 to 4 and then appending a new element 9.

```
lst=list (range (1,5))
print (lst)
```

The preceding statements will give the following output:

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

Now, consider the following statements:

```
lst.append(9)
print(lst)
```

The preceding statements will give the following output:

```
[1,2,3,4,9]
```

Updating an element means changing the value of the element in the list. This can be done by accessing the specific element using indexing or slicing and assigning a new value. Consider the following statements:

```
1st [1] = 8
print(1st)
```

The preceding statements will give:

```
[1. 8. 3, 4, 9]
```

Consider the following statements:

```
lst[1:3] = 10, 11
print(lst)
```

The preceding statements will give:
[1, 10, 11, 4, 9]
Deleting an element from the list can be done using the ‘del’ statement. The del statement takes

the position number of the element to be deleted
del lst[1]
print (lst)

Now, the list appears as:

[1, 11, 4, 9]

We can also delete an element using the remove() method. In this method, we should pass the element to be deleted.

lst.remove(11)
print(lst)

Now, the list appears as:
[1, 4, 9]

Let's write a program to retrieve the elements of a list in reverse order. This can be done easily by using the reverse() method, as:

list.reverse()

This will reverse the order of elements in the list and the reversed elements are available in the list. Suppose, we do not have the reverse() method, then how can we develop logic to display the elements in reverse order? This can be done using while loop and accessing the elements in reverse order. For example, let's assume a list with 5 elements. The positions of these elements can be specified using indexing, as: list[0] to list[4]. To display in reverse order, we should follow the order: list[4] to list[0]. The following while loop can be used to display the list in reverse order:

```
while i>=0:
    print(list[i])
    i-=1
```

In the previous code, 'i' starts with a value 'n-1' where 'n' elements of the list. Thus if the list has 5 elements,'i' would start from the 4th element onwards till the 0th element.

Another way to access elements in reverse order is by using negative indexing. When we represents the 2th element from the end.Hence, we should display from list[-1] to list[-5] so that the list will be displayed in reverse order.

```
i=-1
while i>=-5:
```

print(days[i])
i-=1
This logic is shown in the program below where we are displaying the elements of a list in reverse order using while loop in two different ways.

Method 1:

```
days=['sunday' , 'Monday', 'Tuesday', 'wednesday', 'Thursday']
print('\nIn reverse order: ')
i=len(days)-1
while i>=0:
    print(days[i])
    i-=1
```

Method 2:

```
print('\nIn reverse order: ')
i=-1
while i>=-len(days):
    print(days[i])
    i-=1
```

Output:

```
In reverse order:
Thursday
wednesday
Tuesday
Monday
sunday
```

Concatenation of Two Lists:

We can simply use the '+' operator on two lists to join them. For example, 'x' and 'y' are two lists.If we write x+y, the list 'y' is joined at the end of the list 'x':

```
x = [10,20,30,40,50]
y = [100,110,120]
print(x+y)
```

The concatenated list appears:

[10, 20, 30, 40, 50, 100, 110, 120]

Repetition of Lists

We can repeat the elements of a list ‘n’ number of times using the ‘*’ operator. For example, if we write x*n, the list ‘x’ will be repeated for n times as:
print (x*2)
Now, the list appears as:
[10, 20, 30, 40, 50, 10, 20, 30, 40, 50]

Membership in Lists

We can check if an element is a member of a list or not by using ‘in’ and ‘not in’ operators. If the element is a member of the list, then ‘in’ operator returns True else False. If the element is not in the list, then ‘not in’ operator returns True else False. See the examples below:

x= [10,20, 30,40, 50]
a=20
print(a in x)

The preceding statements will give:

True

If you write, print(a not in x) then, it will give

False

Aliasing and Cloning Lists

Giving a new name to an existing list is called ‘aliasing’. The new name is called ‘alias name’. For example, take a list ‘x’ with 5 elements as
x = [10,20,30,40,50]
To provide a new name to this list, we can simply use assignment operator as:
y=x
In this case, we are having only one list of elements but with two different names ‘x’ and y. Here, ‘x’ is the original name and ‘y’ is the alias name for the same list. Hence, any modifications done to ‘x’ will also modify ‘y’ and vice versa. Observe the following statements where an element x[1] is modified with a new value.

```
x = [10,20,30,40,50]
y=x
print(x)
print(y)
```

Output:

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

```
x = [10,20,30,40,50]
y=x
x[1]=99
print(x)
print(y)
```

Output:

```
[10, 99, 30, 40, 50]
[10, 99, 30, 40, 50]
```

Hence, if the programmer wants two independent lists, he should not go for aliasing.On the other hand, he should use cloning or copying.

Obtaining an exact copy of an existing object (or list) is called ‘cloning’. To clone a list, We can take help of the slicing operation as:

y=x[:]

When we clone a list like this, a separate copy of all the elements is stored into ‘y’. The lists ‘x’ and ‘y’ are independent lists. Hence, any modifications to ‘x’ will not affect ‘y’ and vice versa. Consider the following statements:

```
x=[10, 20, 30,40, 50]
y = x[:]
print(x)
print(y)
```

Output:

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

```
x=[10, 20, 30,40, 50]
y=x[:]
x[1]=99
print(x)
print(y)
```

Output:

```
[10, 99, 30, 40, 50]
[10, 20, 30, 40, 50]
```

We can observe that in cloning, modifications to a list are confined only to that list. The same can be achieved by copying the elements of one list to another using the copy() method. For example, consider the following statement:

y=x.copy()

When we copy a list like this, a separate copy of all the elements is stored into ‘y’.The lists ‘x’ and ‘y’ are independent.Hence,any modifications to ‘x’ will not affect ‘y’ and vice versa.

Methods to Process Lists

The function len() is useful to find the number of elements in a list. We can use this function as:

n= len(list)

Here ‘n' indicates the number of elements of the list. Similar to the len() function, we have a max() function that returns the biggest element in the list. Also, the min() function returns the smallest element in the list. Other than these functions, there are various other methods provided by Python to perform various operations on lists.

sum()	list.sum()	Returns sum of all elements in the list.
index()	list.index()	Returns the first occurrence of x in the list
list.append()	list.append()	Appends x at the end of the list
insert()	list.insert()	Inserts x in to the list in the position specified by i
copy()	list.copy()	Copies all the list elements into a new list and returns it
extend()	list.extend()	Appends list 1 to list
count()	list.count()	Returns number of occurrences of x in the list
remove()	list.remove()	Removes x from the list
pop()	list.pop()	Removes the ending element from the list
sort()	list.sort()	Sorts the elements of the list into ascending order

reverse()	list.reverse()	Reverses the sequence of elements in the list
clear()	list.clear()	Deletes all elements from the list

Ex:

Output:

Finding Biggest and Smallest Elements in a List

Python provides owe functions, max() and min(), which return the biggest and smallest elements from a list. For example, if the list is ‘x’, then:

n1 = max(x)
n2= min(x)
If we do not want to use these functions, but want to find the biggest and smallest elements in the list, how is it possible? For this purpose, we should develop our own logic.First of all, we will store the elements in a list. Let’s take the example list as:

x = [20, 10, 5, 20, 15]

The list elements here are represented as x[0], x[1], ... x[4]. That means the elements in the list are in general represented as x[i]. We will take the first element as the biggest and also as the smallest one as:
big=x[0]
small=x[0]
We will compare big' and 'small' elements with other elements in the list. If the other element is big, then it should be taken as big. That means,
if x[i]>big: big-=x[i]
Similarly, if the other element is <”small', we should take that other element as small as:
if x[i]<small: small=x[i]
Since the comparison starts from 1st element onwards, ‘i' value will change from 1 till the end of the list.
Ex:

Output:

Sorting the List Elements

Python provides the sort() method to sort the elements of a list. This method can be used as:
x.sort()
This will sort the list ‘x' into ascending order. If we want to sort the elements of the list into descending order, then we can mention 'reverse=True' in the sort() method as:
x.sort(reverse=True)
This will sort the list 'x' into descending order.

Suppose, we want to sort a list without using the sort() method, we have to develop logic like bubble sort technique. In this technique, all the 'n' elements from 0 to n are taken and the first element of the list x[j] is compared with the immediate element s If x[j] is bigger than x[j+1], then they are swapped (or interchanged) since in a order we expect the smaller elements to be in the first place. When two elements interchanged, the number of elements to be sorted becomes lesser by 1. When the no more swaps found, the flag' will become 'False' and we can abort sorting Suppose,we give the following elements for sorting:

Original list: 1, 5, 4, 3, 2

Compare 1 with other elements. 1 is smallest hence no swaps. we get:
1, 5, 4, 3, 2
Compare 5 with other elements. Swap 5 with 4, 5 with 3. 5 with 2. We get:
1, 4, 3, 2. 5
Compare 4 with other elements. Swap 4 with 3, 4 with 2. we get:
1,3, 2, 4, 5
Compare 3 with other elements. Swap 3 with 2. we get:
1, 2, 3,4,5
swapping means interchanging the values of two variables. If ‘a’ and ‘b’ are variables, we are supposed to store ‘a’ value into ‘b’ and vice versa. Swapping the values of ‘a’ and ‘b’ can be done using a temporary variable ‘t’ as:
store a value into t. i.e. t= a
store b value into a. i.e, a= b
store t value into b. j.e. b= t

Ex:

Output:

Number of Occurrences of an Element in the List

Python provides a count() method that returns the number of times a particular element is repeated in the list. For example,
n = x.count(y)
Will return the number of times ‘y’ is found in the list ‘x’. This method returns 0 if the element is not found in the list.

It is possible to develop our own logic for the count() method. Let’s take ‘y’ as the element to be found in the list ‘x’. We will use a counter ‘c’ that counts how many times the element ‘y’ is found in the list ‘x’. Initially ‘c’ value will be 0. When ‘y’ is found in the list, ‘c’ will increment by 1. We need a for loop that iterates over all the elements of the list as: for i in x:
This for loop stores each element from the list ‘x’ into ‘y’. So, if ‘y == i’ we found the element and hence we should increment i value by 1. This is shown by the following code

```
c=0
for i in x:
_ if(y==i): c+=1
print('{} is found {} times.'.format(y, c))
```

The above code represents our own logic to find the number of occurrences of ‘y’ in the list ‘x’.
Ex:

Output:

Finding Common Elements in Two Lists

Sometimes,it is useful to know which elements are repeated in two lists.For example,there is a scholarship for which a group of students enrolled.Now, we want to know the names of the students who enrolled for both the scholarships so that we can restrict them to take only one scholarship. That means, we are supposed to find out the common students (or elements) in both the lists.

Let's take the two groups of students as two lists. First of all, we should convert the lists into sets, using set() function, as: set(list). Then we should find the common elements in the two sets using intersection() method as:
set1. intersection(set2)
This method returns a set that contains common or repeated elements in the two sets. This gives us the names of the students who are found in both the sets.

Output:

Storing Different Types of Data in a List

The beauty of lists is that they can store different types of elements. For example, we can store an integer, a string, a float type number etc. in the same string. It is also possible to , retrieve the elements from the list. This has advantage for lists over arrays. We can create list 'emp' as an empty list as:

emp = []
Then we can store the employee data id number,name and salary details into the ‘emp’ list using the append() method.After that, it is possible to retrieve employee details depending on id number of the employee.
Ex:

Output:

Nested Lists

A list within another list is called a nested list. We know that a list contains several elements. When we take a list as an element in another list, then that list is called a nested list. For example, we have two lists ‘a’ and ‘b’ as:
a= [80, 90]
b = [10,.20, 30, a]

Observe that the list ‘a’ is inserted as an element in the list ‘b’ and hence ‘a’ is called a nested list. Let’s display the elements of ‘b’, by writing the following statement: print(b)

The elements of b appears:

[10, 20, 30, [80, 90]]

The last element [80, 90] represents a nested list. So, ‘b’ has 4 elements and they are:
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = [80, 90]

So, b[3] represents the nested list and if we want to display its elements separately, we can use a for loop as:

```
for x in b[3]:
print(x)
```
Ex:

Output:

List Comprehensions

List comprehensions represent creation of new lists from an iterable object (like a list, set, tuple, dictionary or range) that satisfy a given condition. List comprehensions contain very compact code, usually a single statement that performs the task.
Ex:

```
squares=[]
for x in range(1, 11):
squares.append(x**2)
```

The preceding code will create a 'squares' list with the elements as shown below.

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

The previous code can be rewritten in a compact way as:

squares = [x**2 for x in range (1, 11)])

This is called list comprehension. From this, we can understand that a list comprehension consists of square braces containing an expression (i.e. x**2). After the expression, a for loop and then zero or more if statements can be written. Consider the following syntax:

[expression for item1 in iterable1 if statement1
for in iterable2 if statement2
for item3 in iterable3 if statement3...]

Here, ‘iterable' represents a list, set, tuple, dictionary or range object. The result of a list comprehension is a new list that contains elements as a result of executing the expression according to the for loops and if statements. So, we can store the result of a list comprehension in a new list.

Ex:

Suppose, we want to get squares of integers from 1 to 10 and take only the even numbers from the result, we can write a list comprehension as:
even squares = [x**2 for x in range(1, 11) if x % 2==0]
If we display the list 'even_squares', it will display the following list:
[4, 16, 36, 64, 100]
The same list comprehension can be written without using the if statement as:
even squares = [x**2 for x in range (2, 11, 2)]