Comprehensive notes List manipulation class 11

In this article, you will read about comprehensive notes on list manipulation class 11. The topics are accessing or traversing lists, comparing lists, joining lists, repeating or replicating lists, slicing lists, making a true copies of lists.

List manipulation class 11

Now you learned how to create a lists, now its time to learn how to use and manipulate these lists and its elements. Basically lists are mutable i.e. values can be changed in place.

At first sight, lists are similar to strings. So the list manipulation is also similar to string manipulation. The following are ways to access lists:

1. Index
2. Slicing
3. Membership Operator
4. Concatenation
5. Replication
6. Comparing Lists
7. Making a true copy of lists

Now we will discuss each topic of List manipulation class 11 in detail.

Accessing/Traversing Lists by Index

List elements are stored with indexes. Each element of lists is having a specific index like a string. There are two ways to access lists:

1. Positive indexing
2. Negative Indexing
3. Using loop

Positive Indexing

The positive indexing allows to access the value starting with 0,1,2,3 and so on.

```
l = [11,22,33,44,55]
print("L[0]=>",l[0])
print("L[1]=>",l[1])
print("L[2]=>",l[2])
print("L[3]=>",l[3])
print("L[4]=>",l[4])
```

Negative Indexing

Negative indexing refers to access or traverse the list from reverse order in means when the negative index is used, it will display the last element first.

```
l = [11,22,33,44,55]
print("L[0]=>",l[-1])
print("L[1]=>",l[-2])
print("L[2]=>",l[-3])
print("L[3]=>",l[-4])
print("L[4]=>",l[-5])
```

Using Loop

You can use for loop to access list elements. Observe the following code:

```
l = [11,22,33,44,55]
for i in range(0,5):
    print("L[",i,"]=>",l[i])
```

I have used range to display the value of i in output, You can avoid range function as well in following manner:

```
l = [11,22,33,44,55]
for i in l:
    print(i)
```

This section of List manipulation class 11 will cover some important operations with lists.

Slicing Lists - List manipulation class 11

You can extract specified elements by slicing lists. Consider the following syntax for slicing lists:

slice1 = l [start : stop : step]

The start refer to the starting index value of list slice and stop refer to the end index value.

Let's have a look at the following example:

```
l = [11,22,33,44,55]
slice1 = l[0:4]
print(slice1)
```

The negative index value extract the list in reverse order.

```
l = [11,22,33,44,55]

slice1=l[2:-1]
print(slice1)
```

It will slice the list from index 2 i.e. 33 to index -1 i.e. 44. Hence the output will be [33,44].

Now just take a look at the following code:

```
l = [11,22,33,44,55]

slice1=l[2:33]
print(slice1)
```

In the above code, list slicing is starting with index 1 i.e. 33 to index 33 i.e not available in the list, hence it will print the rest of all the values. Python executes the statement without errors.

When both lists are given out of bounds it will return empty lists.

Now let's have a look at following examples:

```
l=[11,22,33,44,55,66,77,88,99]
print(l[0:10:2])
```

The above code slice the lists elements and display alternative index values. Similarly you can change the step value and slice the list accordingly. Consider these examples:

```
l=[11,22,33,44,55,66,77,88,99]
print(l[0:10:3])
print(l[:5:3])
print(l[::3])
print(l[4::2])
```

Type above examples and observe the output, if you have any doubt ask in the comment.

Now in next section of List manipulation class 11, we will discuss operators with lists.

Membership Operator - List manipulation class 11

There are two membership operators as we have covered in string manipulations:

1. in
2. not in

in operator

It returns true if the specified number is present in the list.

```
l=[11,22,33,44,55,66,77,88,99]
```

```
print(44 in l)
```

This code returns **True** as 44 is present in the list.

not in Operator

The not in Operator results exactly opposite compared to in operator.

```
l=[11,22,33,44,55,66,77,88,99]
print(44 not in l)
```

This code returns False as 44 is present in the list.

Concatenation

It is also known as joining lists. To join or concatenate the lists, '+' operator is used. For example,

```
l1 = [4,5,6]
l2 = [7,8,9]
l3 = l1 + l2
print(l3)
```

The above code joins the elements of l1 and l3 into l3. So the output will be [4,5,6,7,8,9].

Replication

It is use to repeat the list number of times.

```
l= [1,2,3]
l = l * 4
```

The output of above code is [1,2,3,1,2,3,1,2,3,1,2,3]

Comparing Lists

To compare lists relational operator is used.

```
l1 = [4,5,6]
l2 = [7,8,9]
if l1>l2:
 print("L1 is greater than L2")
else:
  print("L2 is greater than L1")
```

It will compare the list values and display the results accordingly.

Making true copy of a list

To make a true copy of a list copy function is used.

```
l1 = [1,2,3]
l2 =l1.copy()
print(l2)
```

That's all from the topic List manipulation class 11. If you have any doubts or queries feel free to ask in comment section.

Thank you very much for reading this article.

You can read all the contents of Class 11 from this link.

Class 11