List Manipulation in Python

Python lists are container that are used to store a list of values of any type. It is mutable i.e. we can change the element of a list in place; Python will not create a fresh list when you make any changes to an element of a list.

A list is a standard data type of Python that can store a sequence of values belonging to any type. Lists are depicted through square bracket.

[]	empty list
[1, 2, 3]	list of integers
[1, 2.5, 3.7, 9]	list of numbers
[‘a’, ‘b’, ‘c’]	list of characters
[‘a’, 1, ‘b’, 3.5, ‘zero’]	list of mixed values
[‘one’, ‘two’, ‘three’]	list of strings

Creating list

To create list put a number of expressions in square brackets, That is use square brackets to indicate the start and end, and separate the item by commas.

```
x = [2, 4, 5]
y = [‘abc’, ‘def’]
```

The Empty list

We can also create an empty list as:

```
L = list()
```

it will generate an empty list it is equivalent to 0 or ‘ ’.

Long list

If a list contains many elements, then to enter such long list, you can split it across several lines,

```
sqrs = [0, 1, 4, 9, 16, 25, 36,
    49, 64, 81, 100, 121, 144, 169]
```

Creating list from Existing sequence

You can also use the built in type object to create list from sequence as per the syntax

```
    L = list(<sequence>)
>>> l1 = list (‘hello’)
>>>l1
    [‘h’, ‘e’, ‘l’, ‘l’, ‘o’]
```

we can use this method of creating list of single character or single digits via keyboard input

```
l1 = list(input(“Enter list element”))
Enter list elements : 234567
>>>l1
```

```
[2,  '3', '4', '5',  '6', '7']
```

Notice, this way the data type of all characters entered is string event though we entered digits. To enter a list of integers through keyboard, you can use the method given below:

```
list = eval(input("Enter list to be added "))
print("list you entered ", list)
```

Accessing lists

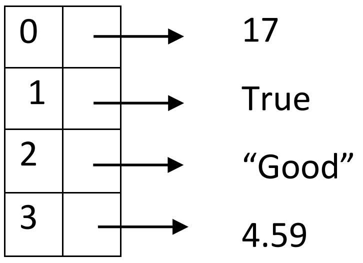
In list the individual element also have their index, it just like string do.

```
list = ['a', 'e', 'I', 'o', 'u']
```

0	1	2	3	4
a	e	i	o	u
-5	-4	-3	-2	-1

Lists are stored in the memory exactly like string except that because some of their objects are larger than other, they store reference of each index instead of single character as in string.

Each of the individual items of list are stored somewhere else in the memory



List[i] will give you element at ith index. List[a:b] will give you element between a to b-1 and so on.

We can print whole list, or we can access individual element from list

```
>>> vowel = ['a', 'e', 'I', 'o', 'u']
>>> vowel
['a', 'e', 'I', 'o', 'u']
>>>vowel[o]
'a'
```

Lists are similar to string in following way

- Length : len(L) return the number of items in the list L.
- Indexing & Slicing : L[i:j] return a new list, containing the object between I & j.
- Member ship operator : Both in and not in operator work on list just like they work for string.
- Concatenation and Replication Operator : The '+' adds one list to the end of another; The * operator repeat a list.

Difference between lists and string

Strings are not mutable, while lists are. You cannot change individual elements of string in place, but lists allow you to do so.

```
    L[i] = <element>      # valid for Python list
```

Traversing a list

Traverse means accessing and processing each element of it. The for loop make it easy to traverse of loop over the item in a list.

```
         for <item> in <List>
             process each element
         L = [‘P’, ‘y’, ‘t’, ‘h’, ‘o’, ‘n’]
         for a in L:
             print(a)
         P
         y
         t
         h
         o
         n
```

List Operations

The most common operations that you perform with list include joining lists, replicating lists and slicing list.

Joining Lists

The concatenation operator +, when used with two lists, joins two lists.

```
>>>lst1 = [1, 3, 5]
>>>lst2 = [6, 7, 8]
>>> lst1 + lst2
[1, 3, 5, 6, 7, 8]
```

The ‘+’ operator concatenates two lists and creates a new list

i.e.

```
>>> x = lst1 + lst2
>>>x
[1, 3, 5, 6, 7, 8]
```

The + operator when used with lists requires that both the operands must be of **list type.** You cannot add a number or any other value to a list. For example, following expression will result into error:

list + number

list + complex number

list + string

Replicating or Repeating lists

Like strings, you use * operator to replicate a list specified number of time.

```
>>> lst1 * 3
>>>[1, 3, 5, 1, 3, 5, 1, 3, 5]
```

Slicing the Lists

List Functions and Methods

Python also offers many built-in functions and methods for list manipulation. Every list object that you create in Python is actually an instance of List class. The list manipulation methods that are being discussed below can be applied to list as per following syntax:

<list-object>.<method-name>

1. The append method

   The append() method adds an item to the end of the list. It works as per following syntax

   ```
   <list-object>.append(<item>)
   >>>colours = ['red', 'green', 'blue']
   >>>colours.append('yellow')
   ```

The append does not return a new list just modifies the original.

```
>>> lst1 = [1, 2, 3]
>>> lst2 = lst.append(12)
>>> lst2
>>> lst1
```

2. The extend method

   The extend() method is also used for adding multiple elements to a list. But it is different from append().

   ```
   <list-object>.extend(<list>)
   ```

   it takes exactly one element (a list type) and return no value.

   That is extend() takes a list as an argument and appends all of the elements of the argument list to the list object on which extend() is applied.

   ```
   >>>t1 = ['a', 'b', 'c']
   >>>t2 = ['d', 'e']
   >>>t1.extend(t2)
   >>>t1
   ['a', 'b', 'c', 'd', 'e']
   ```

   but t2 remains unchanged

   ```
   >>>t2
   ['d', 'e']
   ```

3. The insert() method

   The insert() method is also an insertion method for lists, like append and extend methods. If you want to insert an element somewhere in between or any position of your choice, both append(), and extend() are of no use.

   ```
   l1.insert(<pos>, <item>)
   ```