

# List Manipulation

## Chapter - 7

# Introduction

- In Python, a list is a kind of container that contains collection of any kind of values.
- A List is a mutable data type which means any value from the list can be changed. For changed values , Python does not create a new list.
- List is a sequence like a string and a tuple except that list is mutable whereas string and tuple are immutable.
- In this chapter we will see the manipulation on lists. We will see creation of list and various operation on lists via built in functions.

# List Creation

- List is a standard data type of Python. It is a sequence which can store values of any kind.
- List is represented by square brackets “ [ ] “

For ex -

- [ ] Empty list
- [1, 2, 3] integers list
- [1, 2.5, 5.6, 9] numbers list (integer and float)
- ['a', 'b', 'c'] characters list
- ['a', 1, 'b', 3.5, 'zero'] mixed values list
- ['one', 'two', 'three'] string list
- In Python, only list and dictionary are mutable data types, rest of all the data types are immutable data types.

# Creation of List

- List can be created in following ways-

- Empty list -

```
L = [ ]
```

- list can also be created with the following statement-

```
L = list()
```

```
>>> Mylist = list('hello')
>>> Mylist
['h', 'e', 'l', 'l', 'o']
>>>
```

```
>>> L = ('p', 'a', 'n', 'k', 'j')
>>> newList = list(L)
>>> newList
['p', 'a', 'n', 'k', 'j']
>>>
```

- Long lists-

```
even = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

This is tuple

- Nested list -

```
L = [ 3, 4, [ 5, 6 ], 7]
```

Another method

```
>>> L1 = list(input("Enter List Elements"))
Enter List Elements12345
>>> L1
['1', '2', '3', '4', '5']
```

# Creation of List

-As we have seen in the example

That when we have supplied  
values as numbers to a list even then

They have automatically converted to string

- If we want to pass values to a list in numeric form then we have to write following function -

```
eval(input())
```

```
L=eval(input("Enter list to be added "))
```

**eval ( ) function identifies type of the passed string and then return it.**

```
>>> a="15"  
>>> b="25"  
>>> print(a+b)}  
1525  
>>> print(eval(a)+eval(b))  
40
```

String Values

Another example

```
>>> a="15"  
>>> b=eval(a)  
>>> type(a)  
<class 'str'>  
>>> type(b)  
<class 'int'>  
>>>
```

# Accessing a List

- First we will see the similarities between a List and a String.
- List is a sequence like a string.
- List also has index of each of its element.
- Like string, list also has 2 index, one for forward indexing (from 0, 1, 2, 3, ....to n-1) and one for backward indexing (from -n to -1).
- In a list, values can be accessed like string.

Forward index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
List	R	E	S	P	O	N	S	I	B	I	L	I	T	Y
Backward index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> vowels=['a','e','i','o','u']
>>> vowels[4]
'u'
>>> vowels[-5]
'a'
>>> vowels[-1]
'u'
```

# Accessing a List

- `len( )` function is used to get the length of a list.

```
>>> name=list("Pankaj")
>>> name
['P', 'a', 'n', 'k', 'a', 'j']
>>> len(name)
6
>>>
```

**Important 1:** membership operator (*in, not in*) works in list similarly as they work in other sequence.

- `L[ i ]` will return the values exists at `i` index.
- `L [ i : j ]` will return a new list with the values from `i` index to `j` index excluding `j` index.

```
>>> name=list("Pankaj")
>>> name[3]
'k'
>>> nm=name[2:4]
>>> nm
['n', 'k']
```

**Important 2:** `+` operator adds a list at the end of other list whereas `*` operator repeats a list.

# Difference between a List and a String

- Main difference between a List and a string is that string is immutable whereas list is mutable.
- Individual values in string can't be change whereas it is possible with list.

```
>>> string="aeiou"
>>> string[2]
'i'
>>> string[2]='I'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    string[2]='I'
TypeError: 'str' object does not support item assignment
```

Value didn't  
change in string.  
Error shown.

Value got changed  
in list specifying  
list is mutable

```
>>> st=list("aeiou")
>>> st
['a', 'e', 'i', 'o', 'u']
>>> st[2]='I'
>>> st
['a', 'e', 'I', 'o', 'u']
>>>
```

# Traversal of a list

- Traversal of a list means to access and process each and every element of that list.
- Traversal of a list is very simple with for loop –

```
for <item> in <list>:
```

```
L=['P','Y','T','H','O','N']  
for a in L:  
    print(a)
```

```
L=['P','Y','T','H','O','N']  
length=len(L)  
for a in range(length):  
    print("Index ",a,"and the element at index ",(a-length), "is", L[a])  
'''  
== RESTART: C:/Users/Neha/AppData/Local/Programs/  
Index 0 and the element at index -6 is P  
Index 1 and the element at index -5 is Y  
Index 2 and the element at index -4 is T  
Index 3 and the element at index -3 is H  
Index 4 and the element at index -2 is O  
Index 5 and the element at index -1 is N  
>>> |
```

```
RESTART: C:  
P  
Y  
T  
H  
O  
N
```

\*Python supports UNICODE therefore  
output in Hindi is also possible

# Comparison of Lists

- Relational operators are used to compare two different lists.
- Python compares lists or tuples in lexicographical order, means comparing sequences should be of same type and their elements should also be of similar type.

```
>>> L1, L2=[1, 2, 3], [1, 2, 3]
>>> L3=[1, [2, 3]]
>>> L1==L2
True
>>> L1==L3
False
```

Some examples

```
>>> [1, 2, 8, 9]<[9, 1]
True
>>> [1, 2, 8, 9]<[1, 2, 9, 1]
True
>>> [1, 2, 18, 9]<[1, 2, 9, 10]
False
```

```
>>> L1, L2=[1, 2, 3], [1, 2, 3]
>>> L3=[1, [2, 3]]
>>> L1<L2 ←
False
>>> L1<L3 ←
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    L1<L3
TypeError: '<' not supported between instances of 'int' and 'list'
```

- In first example, python did not raise the error because both the lists are same.
- In second comparison, both the lists are not similar hence, python raised the error.

# List Operations (+, \*)

- Main operations that can be performed on lists are joining list, replicating list and list slicing.
- To join Lists, + **operator**, is used which joins a list at the end of other list. With + operator, both the operands should be of list type otherwise error will be generated.

```
>>> L1=[1,2,3]
>>> L2=[4,5,6,7]
>>> L3=L1+L2
>>> L3
[1, 2, 3, 4, 5, 6, 7]
```

- To replicate a list, \* **operator**, is used.

```
>>> L1=[1,2,3]
>>> L2=L1*3
>>> L2
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

# List Slicing

- To slice a List, syntax is

```
seq = list [ start : stop ]
```

```
>>> LST=[10,12,14,20,22,24,30,32,34]  
>>> SEQ=LST[3:-3]  
>>> SEQ  
[20, 22, 24]  
>>> SEQ=LST[2:4]  
>>> SEQ  
[14, 20]
```

- Another syntax for List slicing is –

```
seq=list[start:stop:step]
```

```
>>> LST=[10,12,14,20,22,24,30,32,34]  
>>> SEQ=LST[0:10:2]  
>>> SEQ  
[10, 14, 22, 30, 34]  
>>> LST[2:10:3]  
[14, 24, 34]  
>>> LST[::3]  
[10, 20, 30]
```

```
>>> LST[::-1]  
[34, 32, 30, 24, 22, 20, 14, 12, 10]
```

# Use of slicing for list Modification

- Look carefully at following examples-

```
>>> L=["one", "two", "three"]
```

```
>>> L
```

```
['one', 'two', 'three']
```

```
>>> L[0:2]=[0,1] ←
```

New value is being assigned here.

```
>>> L
```

```
[0, 1, 'three']
```

```
>>> L=["one", "two", "three"]
```

```
>>> L[0:2]="a" ←
```

Here also, new value is being assigned.

```
>>> L
```

```
['a', 'three'] ←
```

See the difference between both the results.

```
>>> l=[1,2,3]
```

```
>>> l[2:]="604"
```

```
>>> l
```

```
[1, 2, '6', '0', '4']
```

```
>>> l[2:]=144 ←
```

144 is a value and not a sequence.

```
Traceback (most recent call last):
```

```
  File "<pyshell#12>", line 1, in <module>
```

```
    l[2:]=144
```

```
TypeError: can only assign an iterable
```

# List Manipulation

- Element Appending in List

```
list.append(item)
```

```
>>> L=[10,12,14]
>>> L.append(16)
>>> L
[10, 12, 14, 16]
```

- Updating List elements

```
list[index]=<new value>
```

```
>>> L=[10,12,14,30]
>>> L[2]=24
>>> L
[10, 12, 24, 30]
```

- Deletion of List elements

```
del list[index]
```

**Important:** `del` command can be used to delete an element of the list or a complete slice or a complete list.

```
>>> L=[10,12,14,30]
>>> del L[2]
>>> L
[10, 12, 30]
```

**Important:** if we write `del list` complete list will be deleted.

# List Manipulation

- Only one element will be deleted on `pop()` from list.
- `pop( )` function can not delete a slice.
- `pop( )` function also returns the value being deleted.

`list.pop(<index>)`

```
>>> L=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
```

```
>>> L.pop()
```

17

Last item

```
>>> L.pop(5)
```

6

6<sup>th</sup> item

```
>>> L.pop(0)
```

1

1<sup>st</sup> item

# List Functions and Methods

- Python provides some built-in functions for list manipulation
- Syntax is like

`<list-object>.<method-name>`

Function	Details
<code>List.index(&lt;item&gt;)</code>	Returns the index of passed items.
<code>List.append(&lt;item&gt;)</code>	Adds the passed item at the end of list.
<code>List.extend(&lt;list&gt;)</code>	Append the list (passed in the form of argument) at the end of list with which function is called.
<code>List.insert(&lt;pos&gt;,&lt;item&gt;)</code>	Insert the passed element at the passed position.
<code>List.pop(&lt;index&gt;)</code>	Delete and return the element of passed index. Index passing is optional, if not passed, element from last will be deleted.
<code>List.remove(&lt;value&gt;)</code>	It will delete the first occurrence of passed value but does not return the deleted value.

# List Functions and Methods

Function	Details
<b>List.clear ( )</b>	It will delete all values of list and gives an empty list.
<b>List.count (&lt;item&gt;)</b>	It will count and return number of occurrences of the passed element.
<b>List.reverse ( )</b>	It will reverse the list and it does not create a new list.
<b>List.sort ( )</b>	It will sort the list in ascending order. To sort the list in descending order, we need to write---- list.sort(reverse =True).

# List Functions and Methods

- List.index( ) function:

```
>>> lst=[13,18,11,16,18,14]  
>>> lst.index(18)  
1
```

- List.append( ) function:

```
>>> lst=[13,18,11,16,18,14]  
>>> lst.append(27)  
>>> lst  
[13, 18, 11, 16, 18, 14, 27]
```

- List.extend( ) function:

```
>>> lst=[13,18,11,16,18,14]  
>>> lst1=[67,78,89]  
>>> lst.extend(lst1)  
>>> lst  
[13, 18, 11, 16, 18, 14, 67, 78, 89]
```

- List.insert( ) function:

```
>>> t1=['a','e','u']  
>>> t1.insert(2,'i')  
>>> t1  
['a', 'e', 'i', 'u']
```

# List Functions and Methods

- List.pop( ) function:

```
>>> lst=[13,18,11,16,18,14]
>>> lst.pop()
14
>>> lst.pop(2)
11
>>> lst
[13, 18, 16, 18]
```

- List.remove( ) function:

```
>>> lst=[13,18,11,16,18,14]
>>> lst.remove(18)
>>> lst
[13, 11, 16, 18, 14]
```

- List.count( ) function:

```
>>> lst=[2,3,4,5]
>>> lst
[2, 3, 4, 5]
>>> lst.clear()
>>> lst
[]
```

```
>>> lst=["one","two","three","three","four"]
>>> lst.count("three")
2
```

# List Functions and Methods

- List.reverse( ) function:

```
>>> lst=["one", "two", "three", 4, 5]
>>> lst.reverse()
>>> lst
[5, 4, 'three', 'two', 'one']
```

- List.sort( ) function:

```
>>> t1=['e', 'i', 'q', 'p', 'a', 'u', 'o', 'r']
>>> t1.sort()
>>> t1
['a', 'e', 'i', 'o', 'p', 'q', 'r', 'u']
```

## Important

- To sort a list in reverse order, write in following manner–  
**List.sort(reverse=True)**
- If a list contains a complex number, sort will not work.

# Thank you