List in python

The list is written as a list of comma-separated values (items) between square brackets. Items in a list need not be of the same type. List is a mutable data type means elements can be changed.

The first element of the list is stored at index 0 and the last element is stored at the location n-1 where n is the size of list. Also when a list is printed then no need to print each element separately we just need to pass name of list to print() function.

An Example:

```
l1 = [1,2,3,4,5];
print(l1)
print("The first element is",l1[0])
print("The last element is",l1[4])
print("The last element is",l1[-1])
print("The last three elements are",l1[2:])
print("The first three elements are",l1[:3])

l2 = ["python",1990,3.6];
print('\n',l2)
```

The above code will produce following output

```
[1, 2, 3, 4, 5]
The first element is 1
The last element is 5
The last element is 5
The last three elements are [3, 4, 5]
The first three elements are [1, 2, 3]

['python', 1990, 3.6]
```

To print a range of elements from a list we can use slicing operator same as we have used with string.

An Example

```
ls = ["PMD",2018,"Python","MySql","Django"];
print("Elements at index 0,1 are",ls[0:2])
print("Elements at index 2,3,4 are",ls[2:5])
```

The above code will produce following output

```
Elements at index 0,1 are ['PMD', 2018]
Elements at index 2,3,4 are ['Python', 'MySql', 'Django']
```

Let us take a look at some operations that can be performed on list like

1. We can delete an element from list using del statement.
2. We can concatenate two or more list using + operator
3. We can use * operator with list to repeat the list
4. We can use in operator to find an element in list
5. We can iterate through the elements of list

An Example

```
ls1 = [1,2,3,4]
print("The list ls1 is",ls1)
del ls1[0];
print("The list ls1 after removing first element is",ls1,"\n")

ls2 = [5,6,7]
print("The list ls2 is",ls2)
ls1 = ls1 + ls2
print("The list ls1 after concatenating ls2 is",ls1,"\n")

ls3 = [8,9]
print("The list ls3 is",ls3)
```

```
ls3 = ls3*2
print("The list ls3 after repeatation is",ls3,"\n")

ls4 = ["Java","MySql","Networking"]
print("\nThe elements of ls4 ")
for i in ls4:
    print(i)
```

The above code will produce following output

```
The list ls1 is [1, 2, 3, 4]
The list ls1 after removing first element is [2, 3, 4]

The list ls2 is [5, 6, 7]
The list ls1 after concatenating ls2 is [2, 3, 4, 5, 6, 7]

The list ls3 is [8, 9]
The list ls3 after repeatation is [8, 9, 8, 9]


The elements of ls4
Java
MySql
Networking
```

It is possible to use nested lists (i.e. list inside another list),

An Example

```
my_list = ["Java",["1.7","1.8","1.9"],"Python",["3.7","3.8"],"Android
Studio",["2.7","3.0","3.1"]]

print(my_list[0])

print(my_list[1][0])
print(my_list[1][2])

print(my_list[2])

print(my_list[3][0])
print(my_list[3][1])

print(my_list[4])

print(my_list[5][0])
print(my_list[5][1])
```

The code given above will produce following output

```
Java
1.7
1.9
Python
3.7
3.8
Android Studio
2.7
3.0
```

Now let us take a look at some inbuilt functions of the list

Function	Description
append(item)	The append() method takes a single item and adds it to the end of the list. The item can be numbers, strings, another list, dictionary etc. It doesn't return any value.

clear()	clear() method only empties the given list.	
count(element)	The count() method returns the number of occurrences of an element in a list.	
extend(list2)	the extend() method takes a single argument (a list) and adds it to the end. This method can also be used with tuple and set. It doesn't return any value.	
index(element)	The index() method returns the index of the element in the list. If not found, it raises a ValueError exception indicating the element is not in the list.	
insert(index, element)	insert() method inserts the element to the list. It doesn't return any value.	
	Index	position where element needs to be inserted
	Element	this is the element to be inserted in the list
iter(object[, sentinel])	The iter() method returns an iterator for the given object.	
	object	object whose iterator has to be created (can be sets, tuples, etc.)
	sentinel	special value that is used to represent the end of a sequence
len(list)	The len() function returns the number of items of an object. Failing to pass an argument or passing an invalid argument will raise a TypeError exception.	

An Example

```
l1 = [1,2,6,5];
print("Total elements in l1 is",len(l1))
print("The sum of l1 is",sum(l1))
print("The mean of l1 is",sum(l1)/len(l1))

l2 = ["Java","MySQL","Python"]
print("\nThe minimum of l2 is",min(l2))
print("The maximum of l2 is",max(l2))
```

The above code will produce following output

```
Total elements in l1 is 4
The sum of l1 is 14
The mean of l1 is 3.5

The minimum of l2 is Java
The maximum of l2 is Python
```

We can append elements in a list using append() method. This method can be used to take input

An Example:

```
l1 = [] #An empty list
while True:
    x = int(input("Please enter element to be added to list, 0 to stop "))
    if x == 0:
        break
    l1.append(x);
print("list is ",l1)
```

The above code will produce following output

```
Please enter element to be added to list, 0 to stop 1
Please enter element to be added to list, 0 to stop 2
Please enter element to be added to list, 0 to stop 3
Please enter element to be added to list, 0 to stop 4
Please enter element to be added to list, 0 to stop 5
Please enter element to be added to list, 0 to stop 0
list is  [1, 2, 3, 4, 5]
```

Conversion from String to list and vice versa

Join method is used to convert the list into string. It accept a list as an argument and a string is used to call that method like

```
print(",".join(["Java", "Python", "Android"]))
```

Will produce following output

```
Java,Python,Android
```

Apart from that to convert the String into list we have split method

```
print("Metier TechTutors".split())
```

Will produce following output

```
['Metier','TechTutors']
```

Relation between loop and Iterator

The for loop was able to iterate automatically through the list. In fact the for loop can iterate over any iterable. Let's take a closer look at how the for loop is actually implemented in Python.

```
for element in iterable:
    # do something with element
```

is actually implemented as.

```
iter_obj = iter(iterable)

# infinite loop
while True:
    try:
        # get the next item
        element = next(iter_obj)
        # do something with element
    except StopIteration:
        # if StopIteration is raised, break from loop
        break
```

So internally, the for loop creates an iterator object, iter_obj by calling iter() on the iterable. Ironically, this for loop is actually an infinite while loop. We will look at exception handling later.

Dictionary in python

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

An Example

```
dic = {"Company":"HP", "Model":"AT2216TU","Price":28700}
print(dic)
print("dic[Company] = ",dic["Company"])
print("dic[Model] = ",dic["Model"])
print("dic[Price] = ",dic["Price"])

#print("dic[Name] = ",dic["Name"]) this statement will produce KeyError

del dic["Price"]
print("After deletion dic is",dic)

#adding new elements
dic[4] = 15.6
dic["Price"] = 28600
#updating element
dic["Price"] = 29600
```

```
dic[True] = "hello";
print("After updation dic is",dic)
```

The above code will produce following output

```
{'Company': 'HP', 'Model': 'AT2216TU', 'Price': 28700}
dic[Company] =  HP
dic[Model] =  AT2216TU
dic[Price] =  28700
After deletion dic is {'Company': 'HP', 'Model': 'AT2216TU'}
After updation dic is {'Company': 'HP', 'Model': 'AT2216TU', 4: 15.6, 'Price':
29600, True: 'hello'}
```

Some important points to be remembered about list are as follow

1. More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.
2. Keys must be immutable. Which means you can use strings, numbers or tuples (discussed later) as dictionary keys.

clear(), iter() are len() are exactly same.

Function	**Description**	
enumerate()	This method adds counter to an iterable and returns it. The returned object is a enumerate object.	
	Iterable	a sequence, an iterator, or objects that supports iteration
	Start	enumerate() starts counting from this number. If start is omitted, 0 is taken as start.
get(key[, value])	get() method returns the value for the specified key if key is in dictionary.	
	Key	key to be searched in the dictionary
	Value	Value to be returned if the key is not found. The default value is None.
items()	items() method returns a view object that displays a list of dictionary's (key, value) tuple pairs.	
keys()	keys() method returns a view object that displays a list of all the keys in the dictionary	
pop(key[, default])	pop() method removes and returns an element from a dictionary having the given key. If key is not found and default argument is not specified - KeyError exception is raised	
values()	values() method returns a view object that displays a list of all the values in the dictionary.	
update([other])	The update() method takes either a dictionary or an iterable object of key/value pairs. The update() method adds element(s) to the dictionary if the key is not in the dictionary. If the key is in the dictionary, it updates the key with the new value.	

Iterating on the dictionary

An Example:

```
dic = {"Company":"HP", "Model":"AT2216TU","Price":28700}
for k,v in dic.items():
    print(k,":",v)
```

The above code will produce following output

```
Company : HP
Model : AT2216TU
Price : 28700
```

Tuples in Python

A tuple is a sequence of Python objects. It is similar to the list with the difference that the tuples cannot be changed means it is immutable but list is mutable. tuples use parentheses, whereas lists use square brackets. Actually it is not compulsory to use parentheses with tuples because it can also be written without that.

An Example

```
tup1 = () #An empty tuple
tupi = (50) #it's an integer
tup2 = (50,) #A tuple with single element, don't forgot to use comma
tup3 = (1, 2, 3, 4, 5) # is same as tup3 = 1, 2, 3, 4, 5
print(tup3)
print(tup3[0])
print(tup3[-1])
print(tup3[1:4])
```

The above code will produce following output

```
(1, 2, 3, 4, 5)
1
5
(2, 3, 4)
```

As we have already discussed that updating tuple is not possible because it is immutable. So the code given below will produce error

```
tup3[1] = 0 #Error because tuple is immutable
```

Similarly deleting individual element from tuple is not possible, although we can delete whole tuple using `del` keyword.

```
#del tup3[0] Wrong, can't remove individual element

del tup3 #okay, can remove entire tuple
```

Note: methods for tuple are same as for dictionary.