Topics:

- **UNIT – II [14 Lectures]**
- **Data Structures:** Sequence, Lists, Tuple, Sets, Dictionaries
- **Strings and its operations:** Concatenating, Appending, Multiplying strings, Built-in String methods and functions, Slice Operation, Iterating String, String Module
- **Modules:** Importing module, The from..import statement, Name of Module, Making your own modules, The dir()function, The Python Module,Math module, OS Module, Sys Module, Random module
- **Introduction to Functions:** Declaration and Definition, Variable Scope and Lifetime, Return Statements, Types of Arguments, Lambda function, Recursion
- **Functional Programming:** filter() function, map()function, reduce()function

Topics:

- Data Structures
  - Sequence
  - Lists
  - Tuple
  - Sets
  - Dictionaries

- Dynamic way of organizing data in memory in python is done through the following Data Types:
  - Strings
  - Lists
  - Tuples
  - Dictionaries
  - Sets

Introduction

- The most basic data structure in Python is the sequence.
- Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.
- Python has five built-in types of sequences, but the most common ones are strings, lists and tuples
- There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership.
- In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements so on.

Mutable V/s Immutable Data Types

Mutable Data Type	**Immutable Data Type**
Sequences that can be modified after creation	Sequences that cannot be modified after creation
Ex: Lists, Dictionary, Sets	**Ex:** Strings, Tuple
Operations like add, delete and update can be performed	Operations like add, delete and update cannot be performed

List

- A list consist of items separated by commas and enclosed within **square brackets [ ]**
- The values stored in a list are accessed using **indexes**. The index of the **first** element being **0** and **n-1** as that of the **last** element, where **n** is the total number of elements in the list.
- Important thing about a list is that items in a list **need not** be of the **same type**
- An ordered group of sequences enclosed inside square brackets and separated by symbol comma **[,]**
- List are **mutable type**, Python will not create a new list if we modify an element in the list.

- **Syntax:**

```
list1=[]
list2=['Sequence1',]
list3=['Sequence1','Sequence2']
print(list1,list2,list3)
```

Ex:

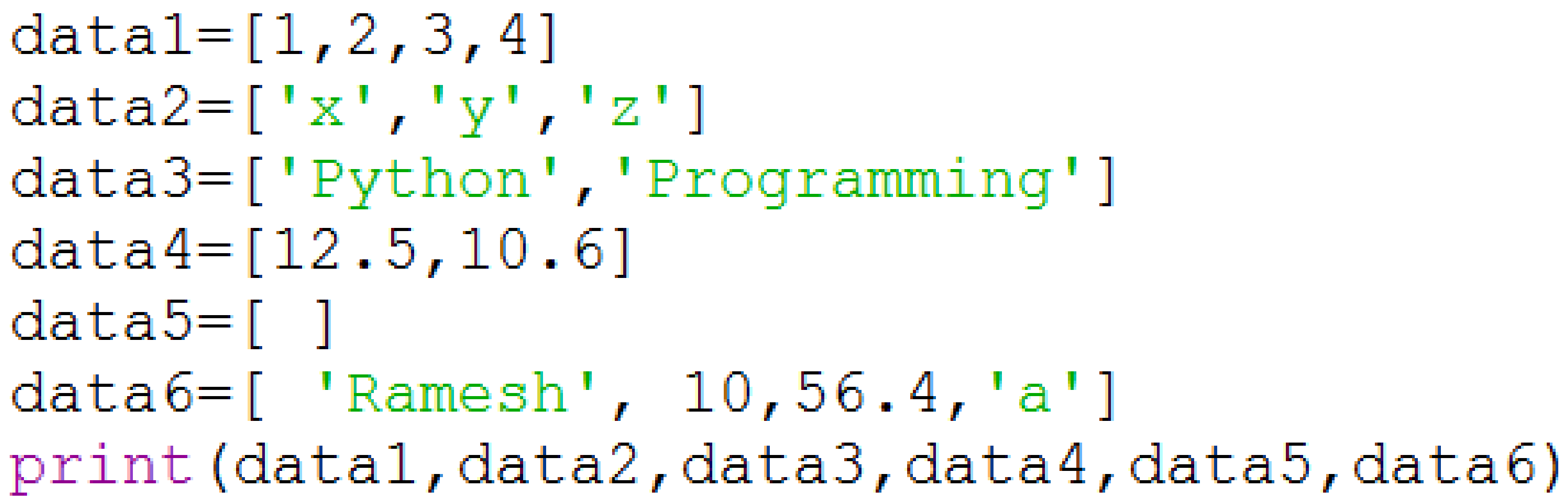
```
language=['Python']
languages =[ 'Python', 'C','C++','Java','Perl','R']
print( language)
print(languages,end="")
```

List Notations and Examples

List Examples	Description
[ ]	An Empty List
[1,3,4,7,9,9]	A list of integers
[1430,”RGUKT”, 168.65]	A list of mixed data types
[“Telangana”,”Andhra”,” Bangalore”,”Chennai”]	A list of Strings
[[1321,”Rajesh”, 72.34],[1456,” Anusha”, 68.89]]	A nested list
[“India”, [“Delhi”,[“Hyderabad”,[“RGUKT”,”Basara”]]]]	A deeply nested list

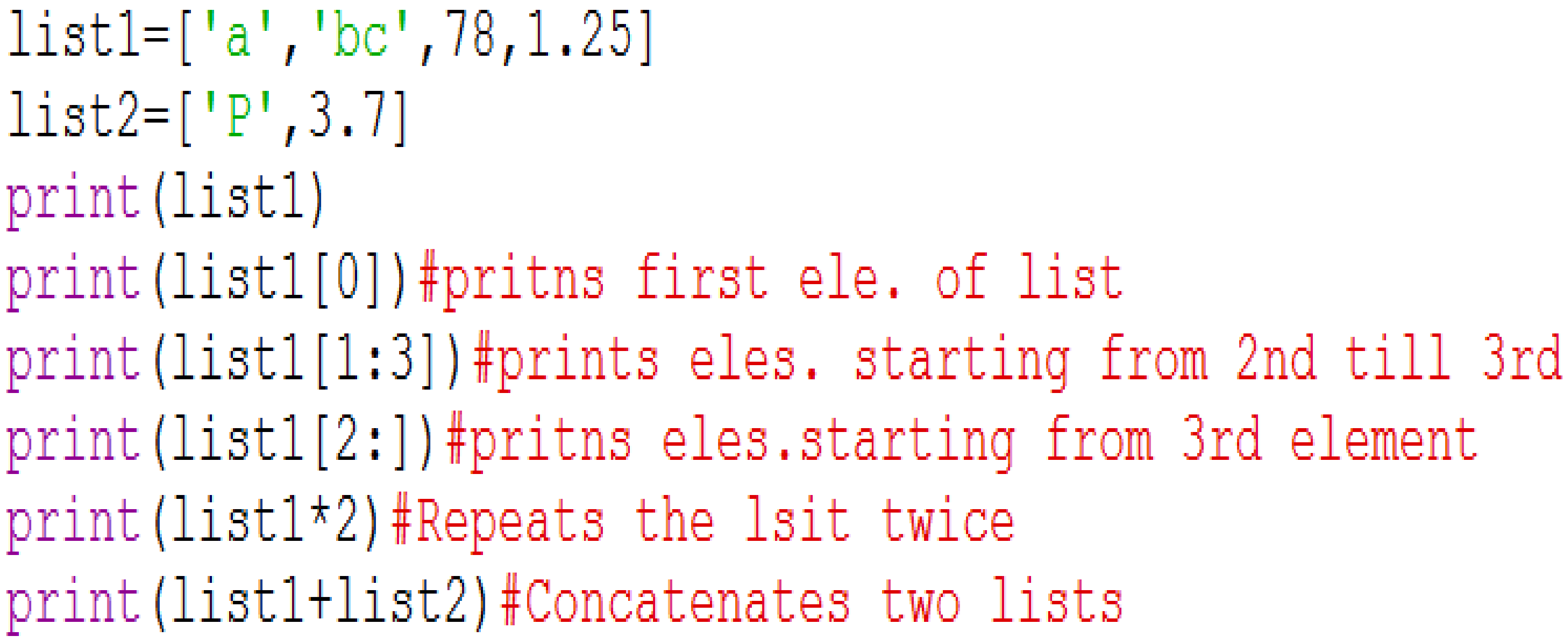
- A list can be composed by storing a sequence of different type of values separated by commas.
- The elements are stored in the index basis with starting index as 0.

Ex:



```
data1=[1,2,3,4]
data2=['x','y','z']
data3=['Python','Programming']
data4=[12.5,10.6]
data5=[ ]
data6=[ 'Ramesh', 10,56.4,'a']
print(data1,data2,data3,data4,data5,data6)
```

Ex:



```
list1=['a','bc',78,1.25]
list2=['P',3.7]
print(list1)
print(list1[0])#pritns first ele. of list
print(list1[1:3])#prints eles. starting from 2nd till 3rd
print(list1[2:])#pritns eles.starting from 3rd element
print(list1*2)#Repeats the lsit twice
print(list1+list2)#Concatenates two lists
```

Accessing Values in Lists

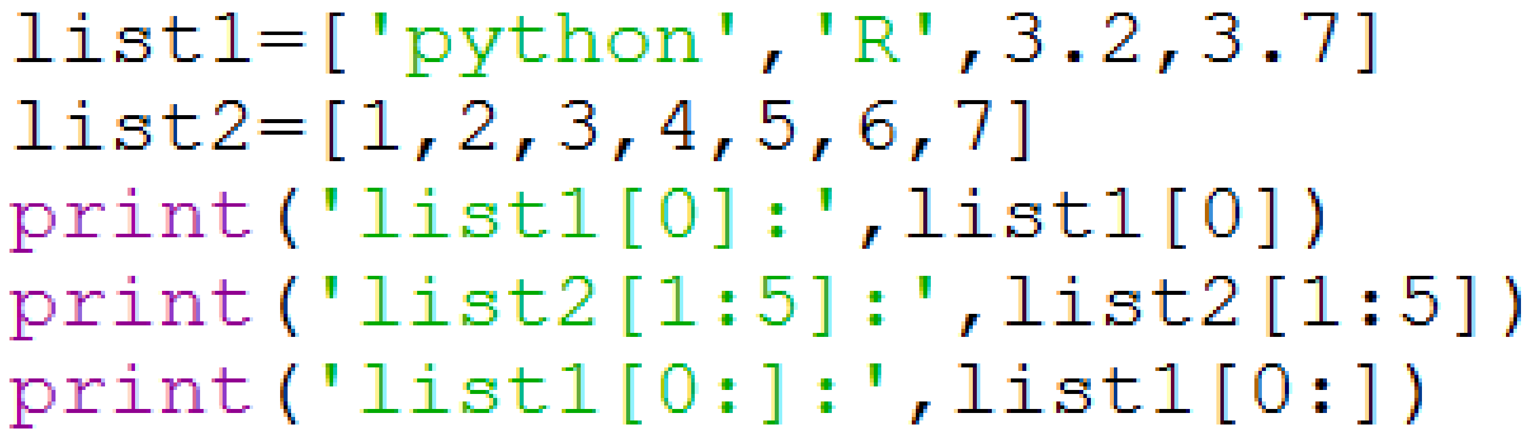
A list can be created by putting the value inside the square bracket and separated by comma.

Syntax:

<list_name>=[value1,value2,value3,...,valuen];

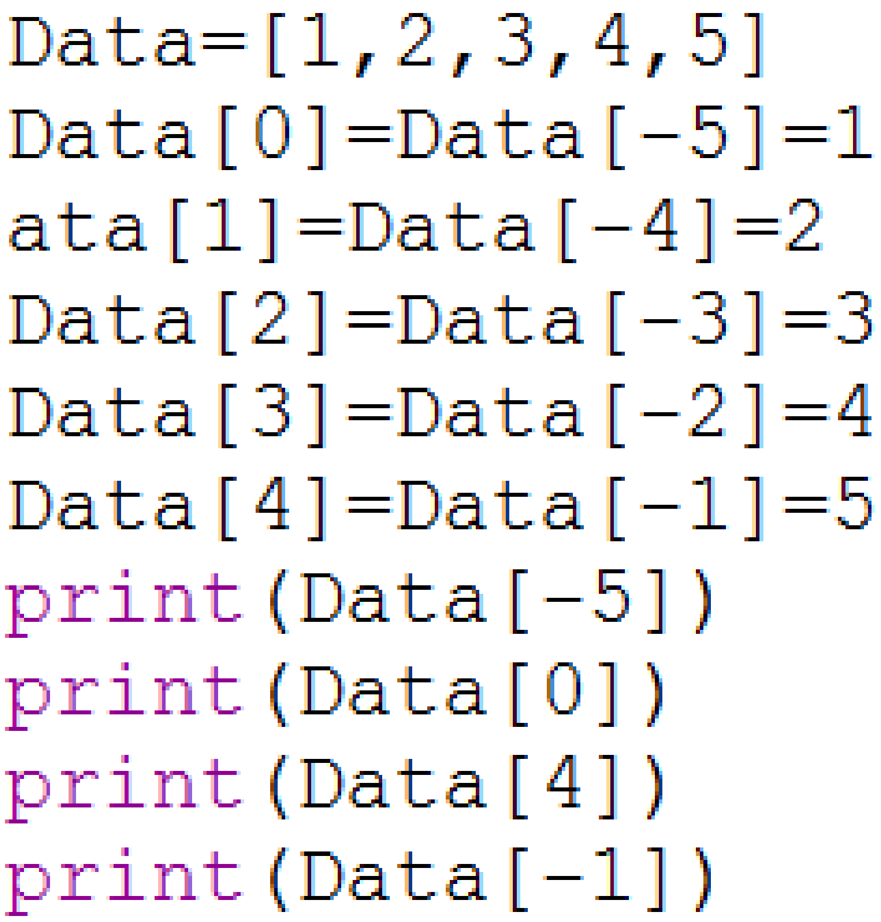
For accessing list :

<list_name>[index]



```
list1=['python','R',3.2,3.7]
list2=[1,2,3,4,5,6,7]
print('list1[0]:',list1[0])
print('list2[1:5]:',list2[1:5])
print('list1[0:]:',list1[0:])
```

Elements in a Lists



```
Data=[1,2,3,4,5]
Data[0]=Data[-5]=1
ata[1]=Data[-4]=2
Data[2]=Data[-3]=3
Data[3]=Data[-2]=4
Data[4]=Data[-1]=5
print(Data[-5])
print(Data[0])
print(Data[4])
print(Data[-1])
```

- **Note:**

  Internal Memory Organization: List do not store the elements directly at the index. In fact a reference is stored at each index which subsequently refers to the object stored somewhere in the memory. This is due to the fact that some objects may be large enough than other objects and hence they are stored at some other memory location.

List operations

- **Adding Lists :** The + operator concatenates lists:

```
a=[1,2,3]#list1
b=[4,5,6]#list2
c=a+b#list3
print('c:',c)
```

```
list1=[10,20]
list1+30
print(list1)
```

Note: '+'operator implies that both the operands passed must be list else error

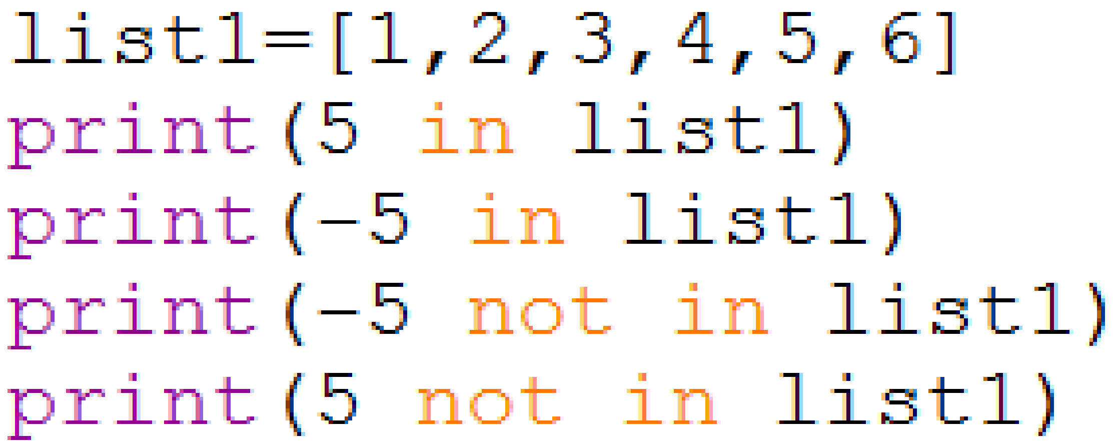
```
TypeError: can only concatenate list (not "int") to list
```

- **Replicating lists:**Similarly, the * operator repeats a list a given number of times:

```
a=[7,8,9]
b=a*3
print('a:',a)
print('b:',b)
```

List membership

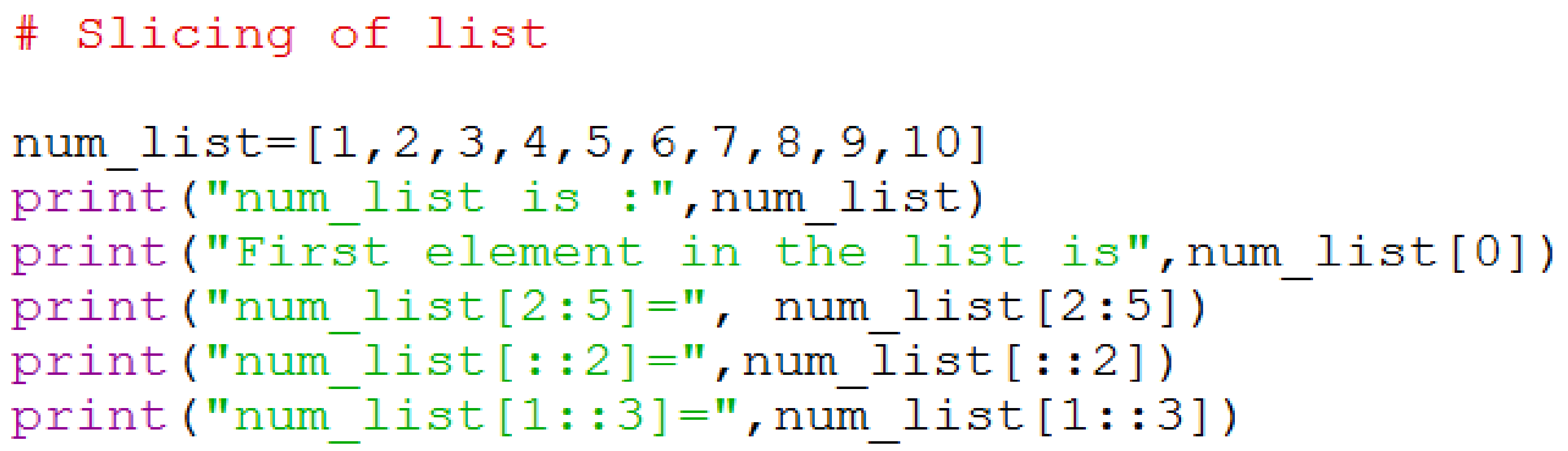
- **in** and **not in** are Boolean operators that test membership in a sequence.
  Example



```
list1=[1,2,3,4,5,6]
print(5 in list1)
print(-5 in list1)
print(-5 not in list1)
print(5 not in list1)
```

List slicing

- A subpart of a list can be retrieved on the basis of index or indices. This subpart is known as list slice.
- **Syntax:** seq=list[start:stop:step]
- **Ex: seq=List[::2] #get every other element, starting with index 0**
- **Seq=List[1::2]**



```
# Slicing of list

num_list=[1,2,3,4,5,6,7,8,9,10]
print("num_list is :",num_list)
print("First element in the list is",num_list[0])
print("num_list[2:5]=", num_list[2:5])
print("num_list[::2]=",num_list[::2])
print("num_list[1::3]=",num_list[1::3])
```

```
fruits=["bana","orange","apple","kiwi"]
print(fruits[:])
print(fruits[0])
print(fruits[1])
print(fruits[0:3])
print(fruits[1:1])
print(fruits[1:2])
print(fruits[1:3])
print(fruits[:3])
print(fruits[0:4])
print(fruits[-1])
```

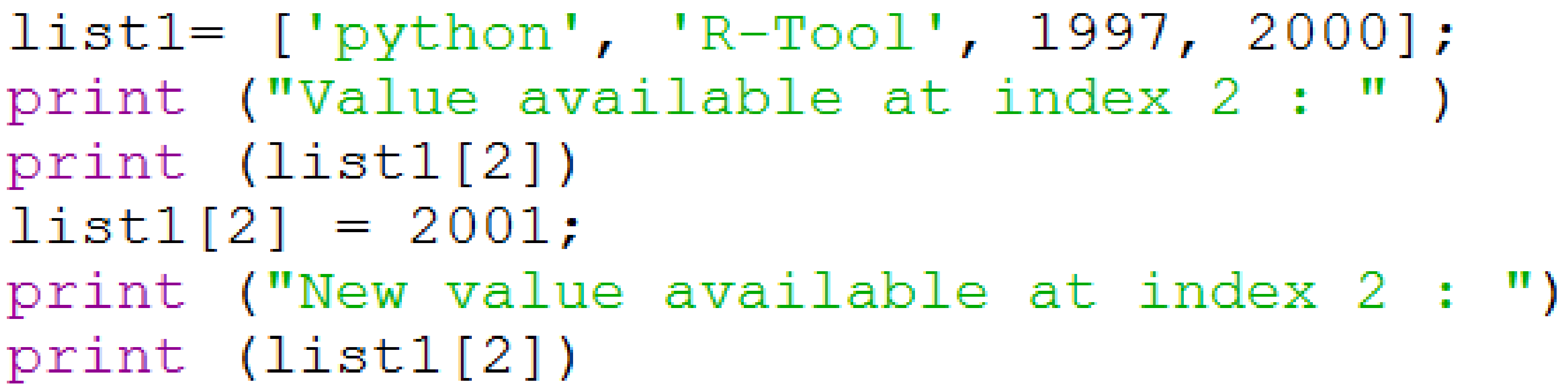
Other Operations:

- Apart from above operations various other functions can also be performed on List such as Updating, Appending and Deleting elements from a List:

  a) Updating elements in a List:

- To update or change the value of particular index of a list, assign the value to that particular index of the List.
- **Syntax:**

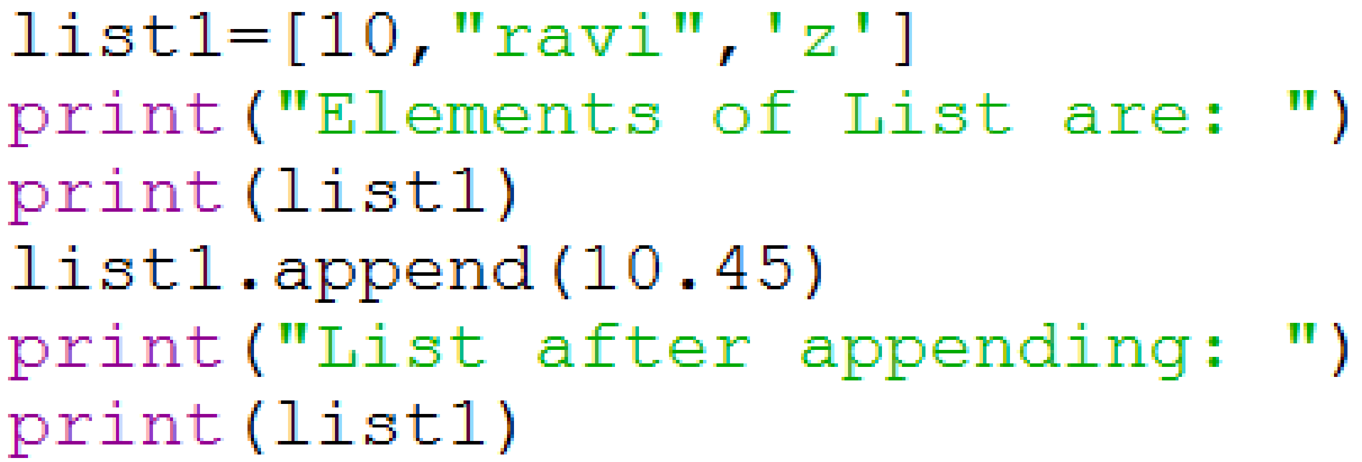
  <list_name>[index]=<value>



```
list1= ['python', 'R-Tool', 1997, 2000];
print ("Value available at index 2 : " )
print (list1[2])
list1[2] = 2001;
print ("New value available at index 2 : ")
print (list1[2])
```

Appending elements to a List

- append( ) method is used to append i.e., add an element at the end of the existing elements.
- **Syntax:** <list_name>.append(item)
- **Eg**

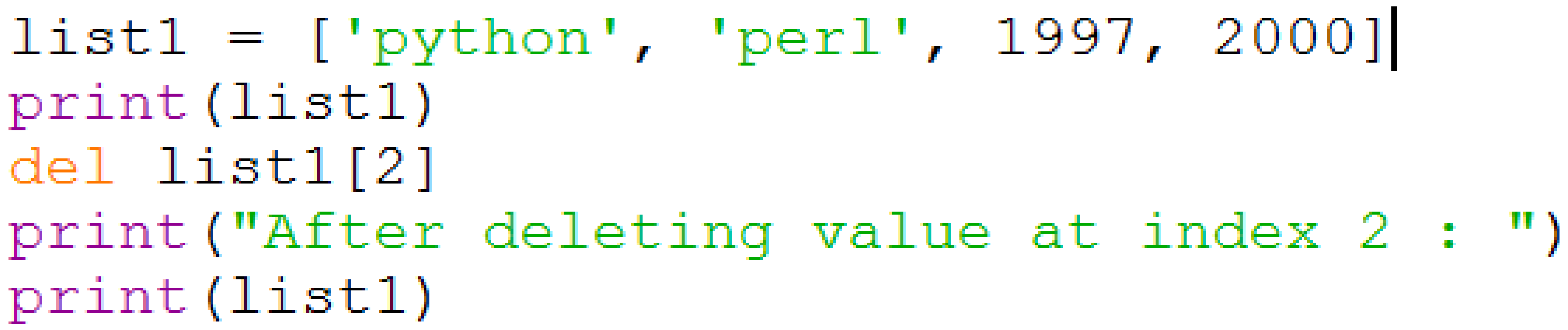


```
list1=[10,"ravi",'z']
print("Elements of List are: ")
print(list1)
list1.append(10.45)
print("List after appending: ")
print(list1)
```

Delete List Elements

- del statement can be used to delete an element from the list. It can also be used to delete all items from startIndex to endIndex.

  Ex: del <list_name> [value]



```
list1 = ['python', 'perl', 1997, 2000]
print(list1)
del list1[2]
print("After deleting value at index 2 : ")
print(list1)
```

Basic Lists Operations

- There are many Built-in functions for Lists. They are as follows:
- **There are following List functions:**

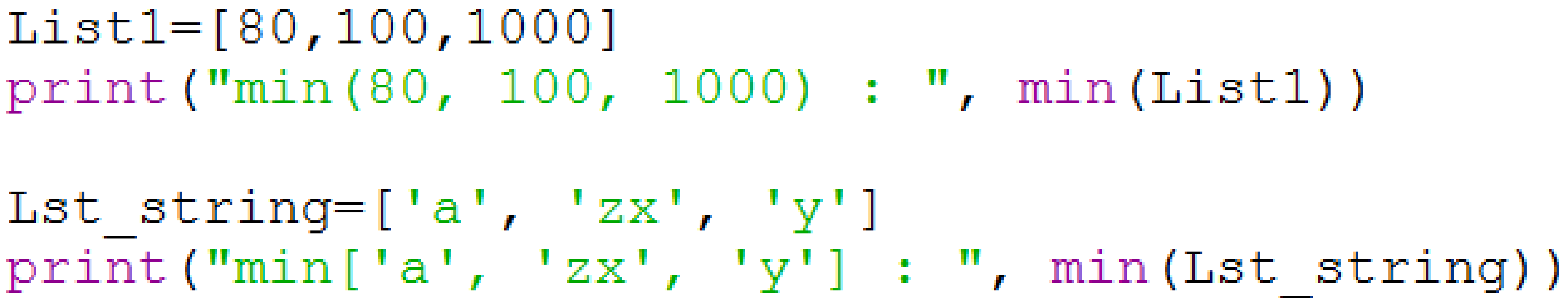
Function	Description
min(list)	Returns the minimum value from the list given.
max(list)	Returns the largest value from the given list.
len(list)	Returns number of elements in a list.
sum(list)	Adds the values in the list that has numbers
list(sequence)	Takes sequence types and converts them to lists.
sorted(list)	Returns a new sorted list. The Original list is not sorted

min(list)

- The method min() returns the elements from the list with minimum value.

  Syntax

  min(list)



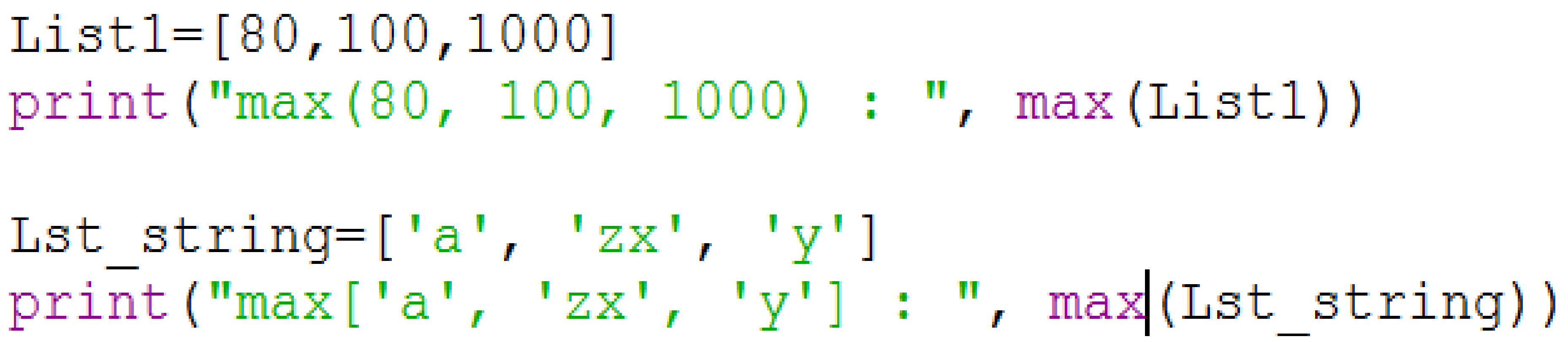
```
List1=[80,100,1000]
print("min(80, 100, 1000) : ", min(List1))

Lst_string=['a', 'zx', 'y']
print("min['a', 'zx', 'y'] : ", min(Lst_string))
```

max(list)

The method max returns the elements from the list with maximum value.

Syntax

max(list)



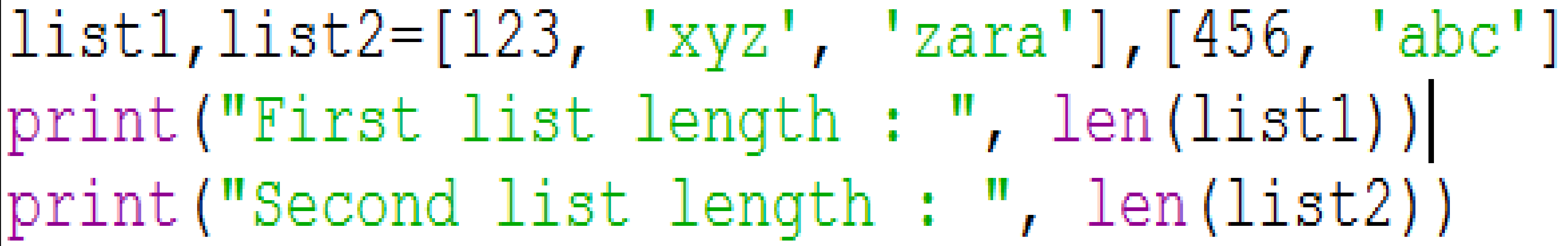
```
List1=[80,100,1000]
print("max(80, 100, 1000) : ", max(List1))

Lst_string=['a', 'zx', 'y']
print("max['a', 'zx', 'y'] : ", max(Lst_string))
```

len(list)

- The method len( ) returns the number of elements in the list.

Syntax

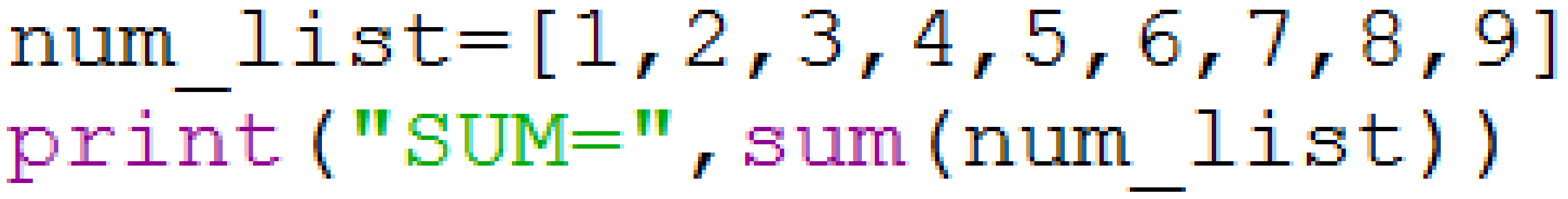
len(list)



```
list1,list2=[123, 'xyz', 'zara'],[456, 'abc']
print("First list length : ", len(list1))
print("Second list length : ", len(list2))
```

sum(list)

- It adds the values in the list that has numbers
- **Syntax: sum(list)**



```
num_list=[1,2,3,4,5,6,7,8,9]
print("SUM=",sum(num_list))
```

list(seq)

- The method list( ) takes sequence types and converts them to lists. This is used to convert a given tuple,string,dictionary,set into list.
- **Note:** Tuple are very similar to lists with only difference that element values of a tuple can not be changed and tuple elements are put between parentheses instead of square bracket.

Syntax

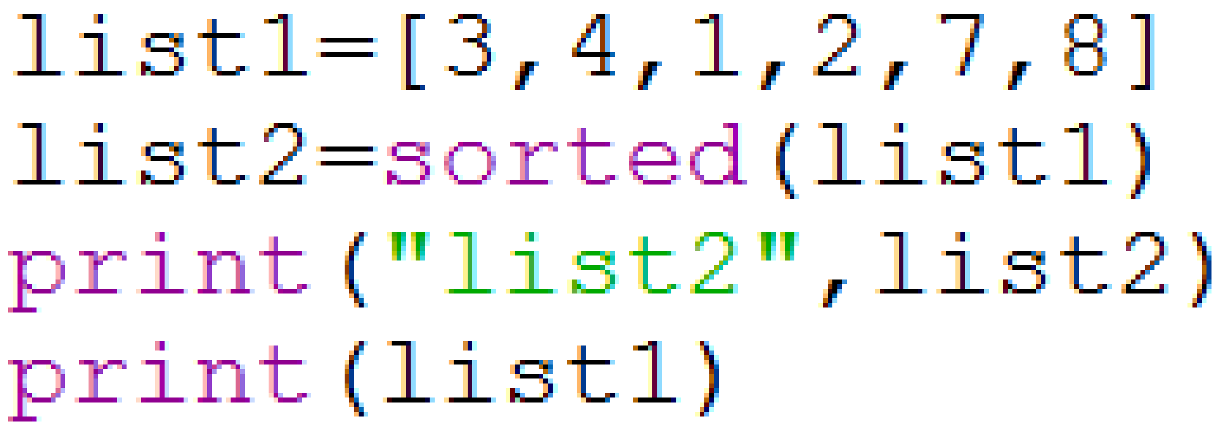
list(seq)

```
l1=list("HELLO")
print("l1:",l1)
```

```
aTuple=(123, 'xyz', 'zara', 'abc')
aList=list(aTuple)
print("List elements : ", aList)
```

sorted(list)

- Returns a new sorted lsit. The original list is not sorted.
- **Syntax: sorted(list)**



```
list1=[3,4,1,2,7,8]
list2=sorted(list1)
print("list2",list2)
print(list1)
```

There are following built-in methods of List

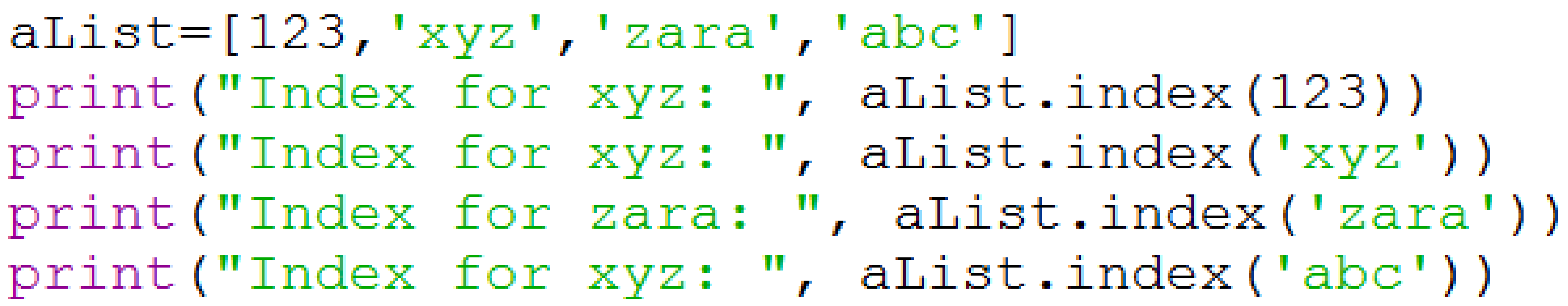
Methods	Description
index(object)	Returns the index value of the object.
count(object)	It returns the number of times an object is repeated in list.
pop()/pop(index)	Returns the last object or the specified indexed object. It removes the popped object.
insert(index,object)	Insert an object at the given index.
extend(sequence)	It adds the sequence to existing list.
remove(object)	It removes the object from the given List.
reverse()	Reverse the position of all the elements of a list.
sort()	It is used to sort the elements of the List.
append()	It appends an element to the list,element is added at the end

index( )

- The method index() returns the lowest index in list that obj appears.

Syntax

list.index(obj)



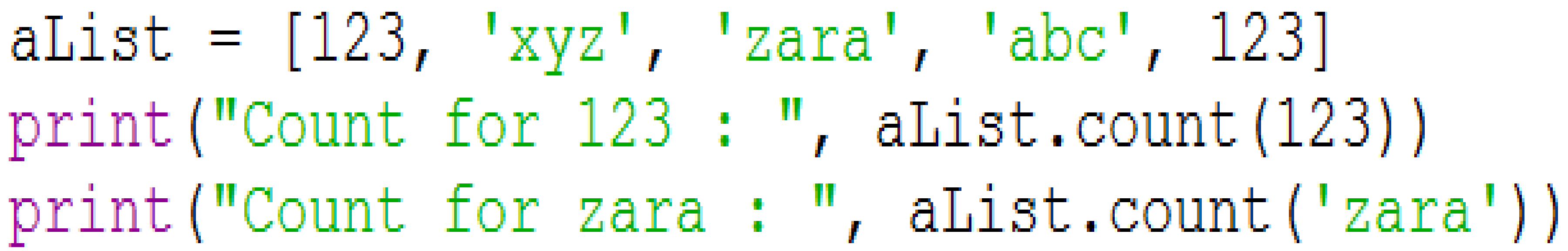
```
aList=[123,'xyz','zara','abc']
print("Index for xyz: ", aList.index(123))
print("Index for xyz: ", aList.index('xyz'))
print("Index for zara: ", aList.index('zara'))
print("Index for xyz: ", aList.index('abc'))
```

count( )

The method count() returns count of how many times obj occurs in list.

Syntax

list.count(obj)



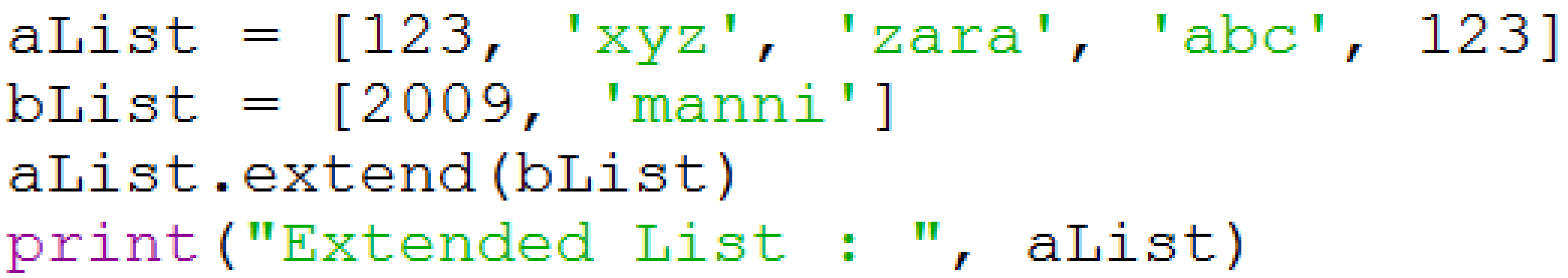
```
aList = [123, 'xyz', 'zara', 'abc', 123]
print("Count for 123 : ", aList.count(123))
print("Count for zara : ", aList.count('zara'))
```

extend( )

- The method extend( ) appends the contents of seq to list.

Syntax

list.extend(seq)



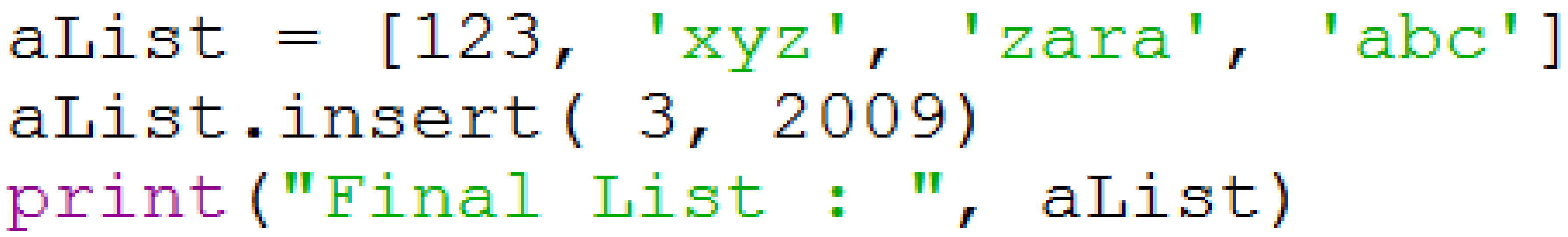
```
aList = [123, 'xyz', 'zara', 'abc', 123]
bList = [2009, 'manni']
aList.extend(bList)
print("Extended List : ", aList)
```

insert( )

The method insert() inserts object obj into list at offset index.

Syntax

list.insert(index, obj)



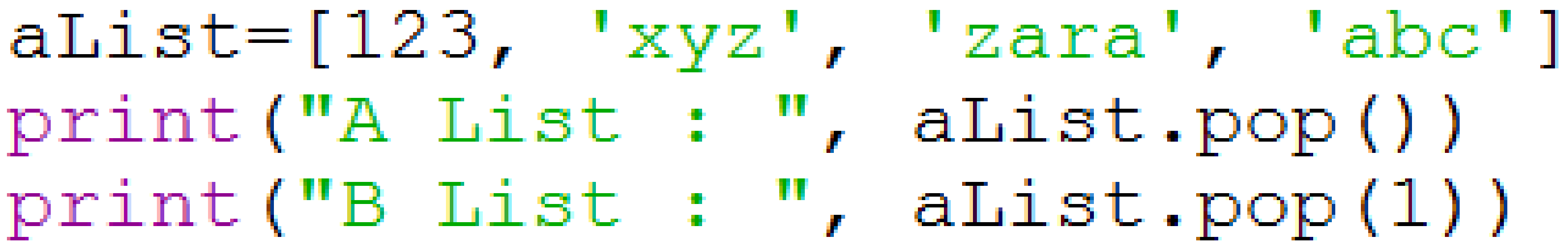
```
aList = [123, 'xyz', 'zara', 'abc']
aList.insert( 3, 2009)
print("Final List : ", aList)
```

pop( )

The method pop() removes and returns last object or obj from the list.

Syntax

list.pop(obj=list[-1])



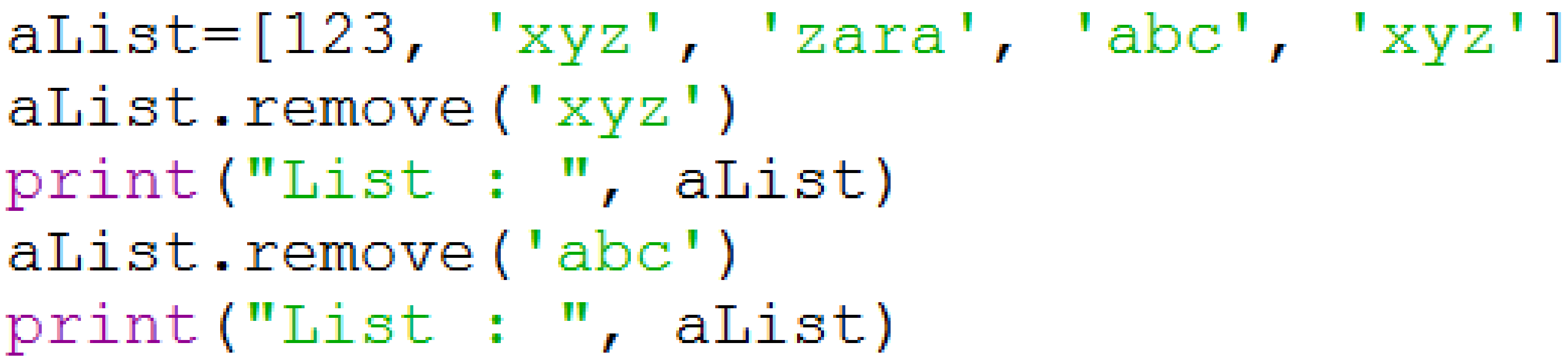
```
aList=[123, 'xyz', 'zara', 'abc']
print("A List : ", aList.pop())
print("B List : ", aList.pop(1))
```

remove( )

- This is the object to be removed from the list.
- This method does not return any value but removes the given object from the list.

Syntax

- list.remove(obj)



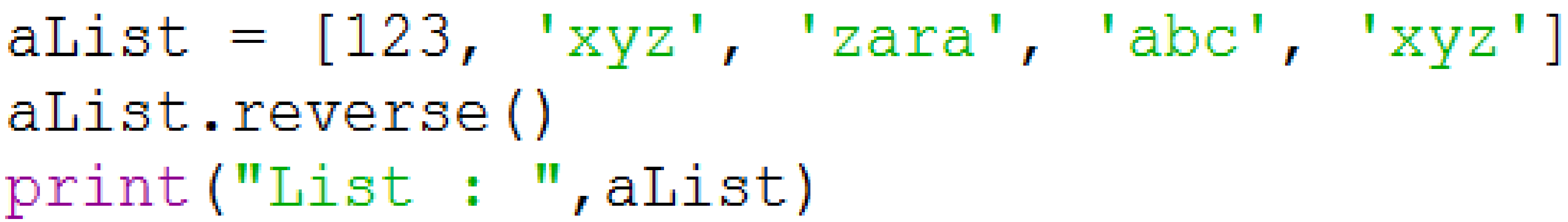
```
aList=[123, 'xyz', 'zara', 'abc', 'xyz']
aList.remove('xyz')
print("List : ", aList)
aList.remove('abc')
print("List : ", aList)
```

reverse( )

- The method reverse( ) reverses objects of list in place.
- This method does not return any value but reverse the given object from the list.

Syntax

list.reverse( )



```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.reverse()
print("List : ",aList)
```

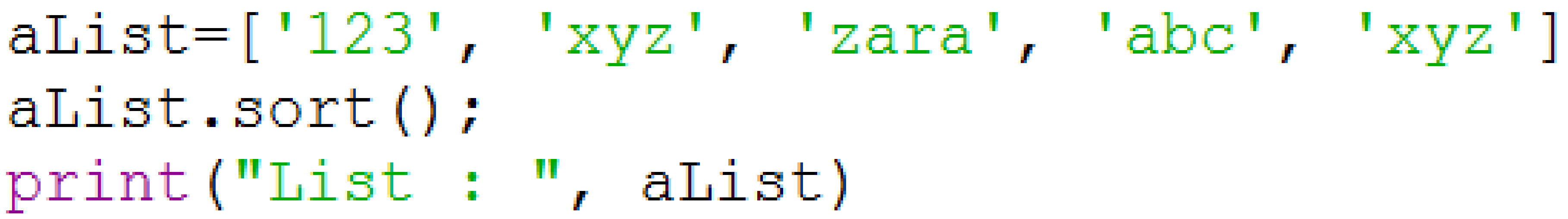
sort( )

- The method sort( ) sorts objects of list, use compare func if given.

•

Syntax

list.sort([func])



```
aList=['123', 'xyz', 'zara', 'abc', 'xyz']
aList.sort();
print("List : ", aList)
```

all( )

- Returns True if all elements of the list are TRUE(or if the list is empty)

```
list_num=[0,1,2,3,4,5]
list1_num=[1,2,3,4,5]
print(all(list_num))
print(all(list1_num))
```

any( )

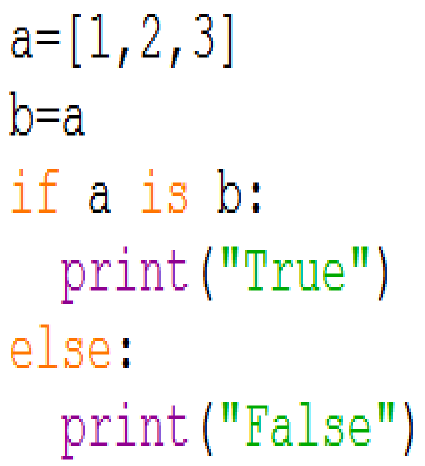
- Returns True if any element of the list are TRUE.If the list is empty, returns False

```
list_num=[0,1,2,3,4,5]
list1_num=[]
print(any(list_num))
print(any(list1_num))
```

Aliasing

- Since variables refer to objects, if we assign one variable to another, both variables refer to the same object

Example



```
a=[1,2,3]
b=a
if a is b:
  print("True")
else:
  print("False")
```

Output

True

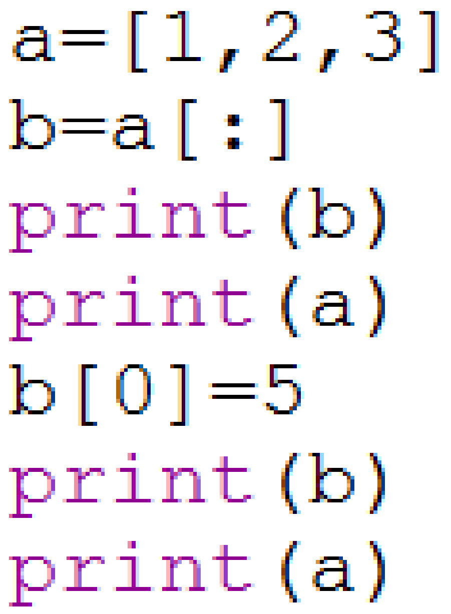
In this case, the state snapshot looks like this:

- a-------[1,2,3]
- b-------[1,2,3]
- Because the same list has two different names, a and b , we say that it is aliased. Changes made with one alias affect the other

Cloning

- If we want to modify a list and also keep a copy of the original, we need to be able to make a copy of the list itself, not just the reference. This process is sometimes called cloning, to avoid the ambiguity of the word copy.
- The easiest way to clone a list is to use the slice operator:

Ex:



```
a=[1,2,3]
b=a[:]
print(b)
print(a)
b[0]=5
print(b)
print(a)
```

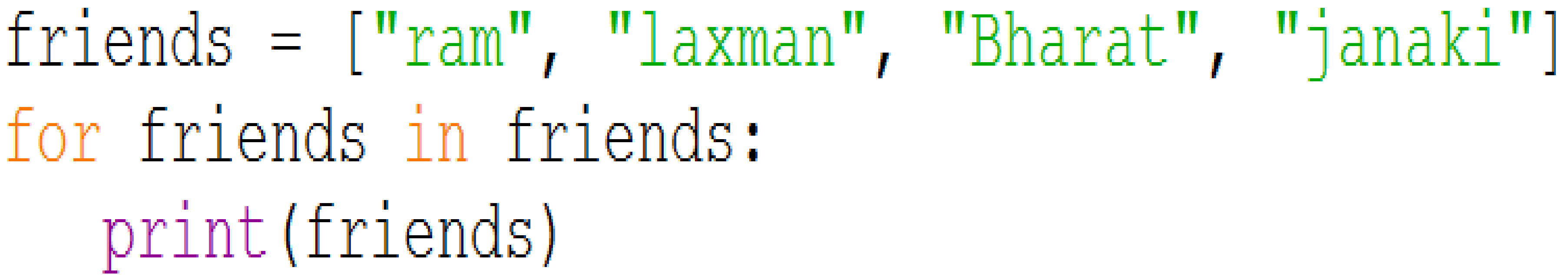
For loop

Syntax

```
for VARIABLE in LIST:
    BODY
```

Note:

Each time through the loop, the variable i is used as an index into the list, printing the i ‘th element. This pattern of computation is called a list traversal.

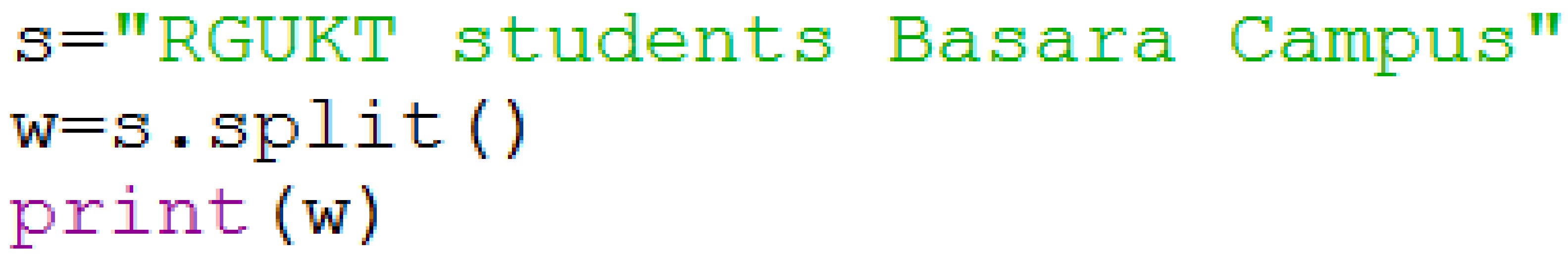


```
friends = ["ram", "laxman", "Bharat", "janaki"]
for friends in friends:
  print(friends)
```

Strings and lists

- split method breaks a string into a list of words. By default, any number of whitespace characters is considered a word boundary:

Ex:

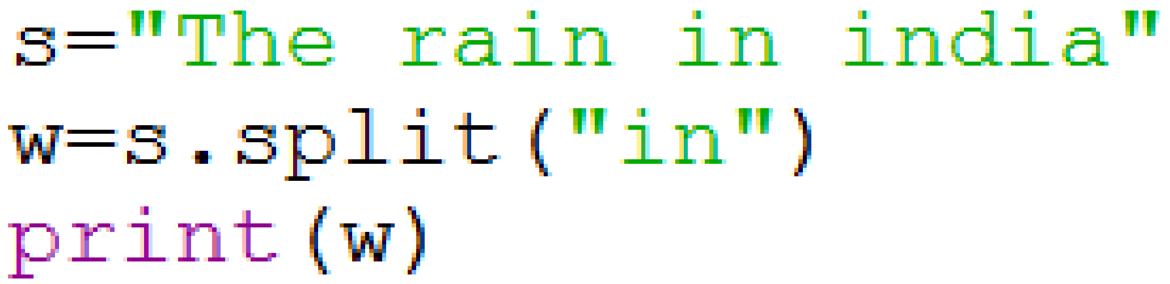


```
s="RGUKT students Basara Campus"
w=s.split()
print(w)
```

Strings and lists

- An optional argument called a delimiter can be used to specify which string to use as the boundary marker between substrings. The following example uses the string **in** as the delimiter:

Ex:

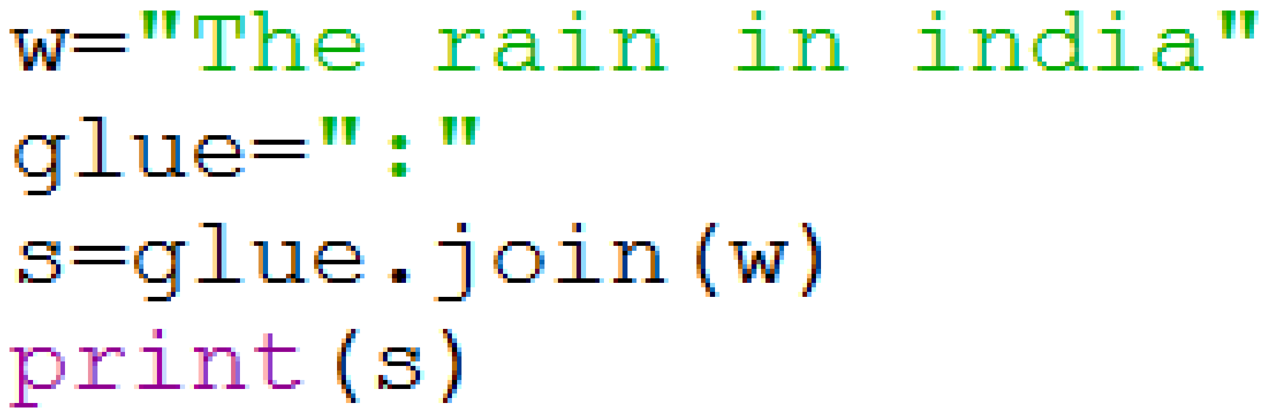


```
s="The rain in india"
w=s.split("in")
print(w)
```

Strings and lists

- The inverse of the split method is join . You choose a desired separator string, (often called the glue) and join the list with the glue between each of the elements:

Ex:



```
w="The rain in india"
glue=":"
s=glue.join(w)
print(s)
```

Nested lists

- A nested list is a list that appears as an element in another list. In this list, the element with index 3 is a nested list

Example

```
nested=["hello", 2.0, 5, [10, 20]]
print(nested[3])
print(nested[2])
```

- To extract an element from the nested list, we can proceed in two steps

```
print(nested[3][0])
```

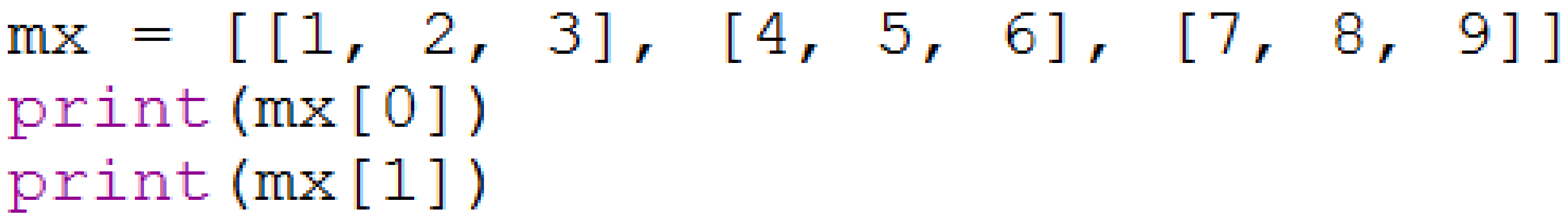
Matrices

- Nested lists are often used to represent matrices.
- For example, the matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

mx = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

- mx is a list with three elements, where each element is a row of the matrix. We can select an entire row from the matrix in the usual way:

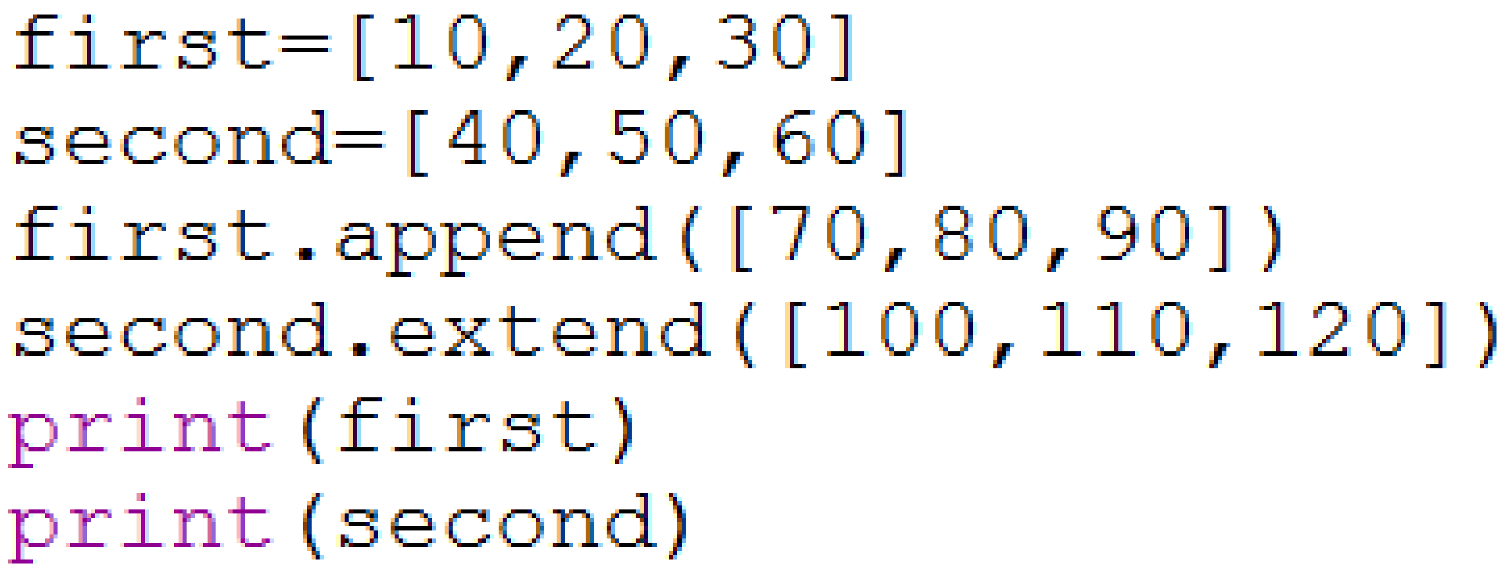


```
mx = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(mx[0])
print(mx[1])
```

difference between append( ) and extend( )

- The method **append** adds its parameter as a single element to the list, while **extend** gets a list and adds its content.

Ex:



```
first=[10,20,30]
second=[40,50,60]
first.append([70,80,90])
second.extend([100,110,120])
print(first)
print(second)
```

Difference between del( ), remove( ), pop( )

remove Takes value, removes first occurrence and returns nothing

Example

```
a=[0, 2, 2, 21]
a.remove(21)
print(a)
```

del takes index, removes value at that index and returns nothing

Ex:

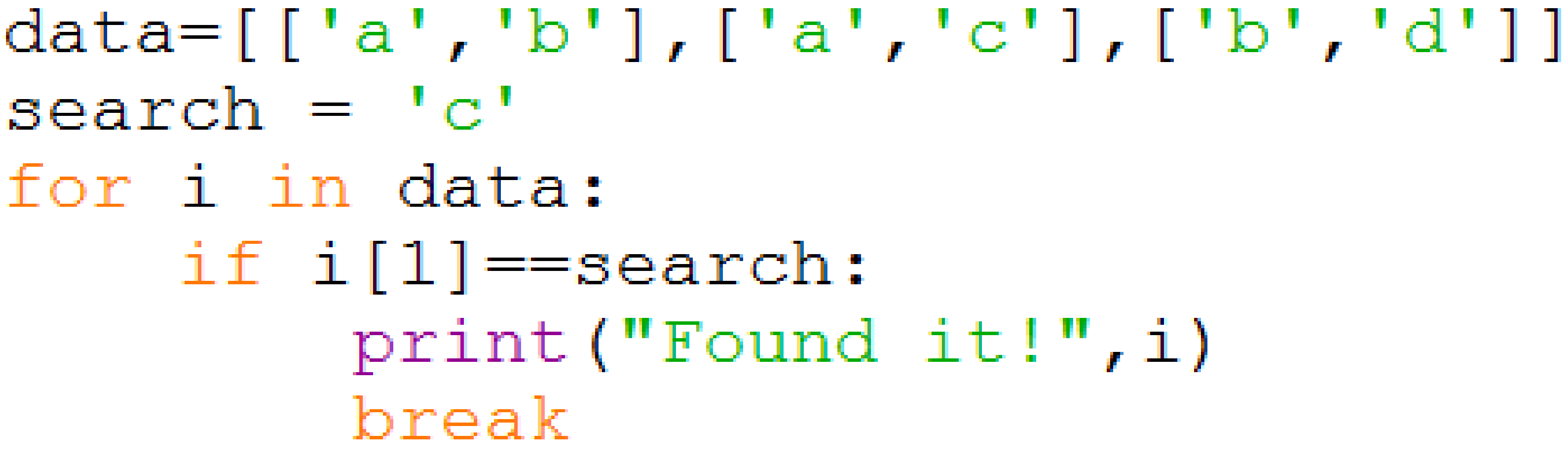
```
a=[0, 2, 2, 21]
del a[1]
print(a)
```

pop takes Index & returns Value

Ex:

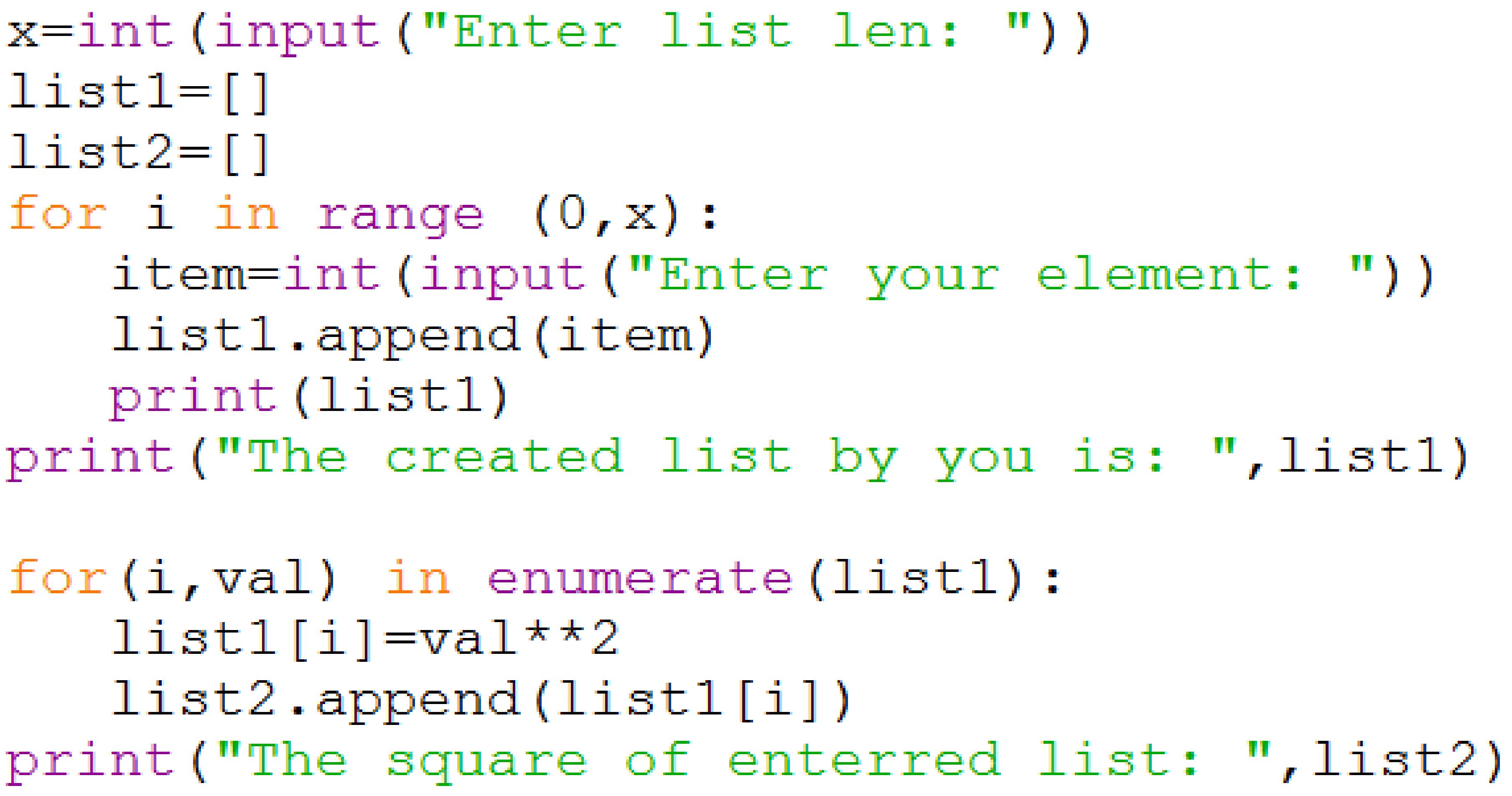
```
a=[4, 3, 2, 1]
a.pop(1)
print(a)
print(a.pop(1))
print(a)
```

Searching list



```
data=[['a','b'],['a','c'],['b','d']]
search = 'c'
for i in data:
    if i[1]==search:
        print("Found it!",i)
        break
```

Write program to create a list by giving user input and do square of a each element?



```
x=int(input("Enter list len: "))
list1=[]
list2=[]
for i in range (0,x):
    item=int(input("Enter your element: "))
    list1.append(item)
    print(list1)
print("The created list by you is: ",list1)

for(i,val) in enumerate(list1):
    list1[i]=val**2
    list2.append(list1[i])
print("The square of enterred list: ",list2)
```

Write program to create a list by giving user input and do addition of two lists

```
x=int(input("Enter list len: "))
list1=[]
list2=[]
list3=[]
for i in range (0,x):
   item=int(input("Enter your element: "))
   list1.append(item)
   print("list1:",list1)
for i in range (0,x):
   item=int(input("Enter your element: "))
   list2.append(item)
   print("list2:",list2)
print("The created list by you is: ",list2)

for i in range(0,x):
    ele=list1[i]+list2[i]
    list3.append(ele)
print("Adding the elements of two lists",list1,'+',list2,'=',list3)
```

Basic List Operations

Operation	Python Expression	Result
Length	Len([4,5,6])	3
Concatenation	[1,3,7]+[8,9,9]	[1,3,7,8,9,9]
Repetition	[‘Hello’] * 3	[‘Hello’,’Hello’,’Hello’]
Membership	7 in [1,3,7]	True
Iteration	for n in [1,3,7] : print(n)	1 3 7
Indexing : Offset starts at 0	n=[1,3,7] print(n[2])	7
Negative Slicing : Cont from right	n= [1,3,7] print(n[-2])	3
Slicing	n= [1,3,7] print(n[1:])	[3,7]

IIIT BASARA

Infosys | Campus Connect

For Details Contact Me @ :

9247448766

ravikanth27787@gmail.com

python

Programming Language