List :

List is a collection of different values or different types of items.

Properties of list:

1.Mutable

2.Ordered

3.Heterogenous

4.Duplicates

Type *Markdown* and LaTeX: α^2

Create List

The list can be created using either the list constructor or using square brackets [ ].

In [1]:

```
# Using square brackets[]
my_list1 = [1, 2, 3]
print(my_list1)
```

```
[1, 2, 3]
```

In [2]:

```
#What is the data type?
type(my_list1)
```

Out[2]:

```
list
```

In [3]:

```
# Using list constructor
my_list2 = list((1, 2, 3))
print(my_list2)
```

```
[1, 2, 3]
```

In [4]:

```
type(my_list2)
```

Out[4]:

```
list
```

In [5]:

```
#List allows duplicate values
my_list3 = [1, 2, 3, 2, 1]
print(my_list3)
```

```
[1, 2, 3, 2, 1]
```

In [6]:

```
#In order to find the number of items present in a list OR size of list.

my_list = [1, 2, 3, 4, 5, 6]
print(len(my_list))
```

```
6
```

In [7]:

```
#List items can be of any data type:

list1 = ["apple", "banana", "cherry"] #String type
list2 = [1, 5, 7, 9, 3] #integer type
list3 = [True, False, False] #boolean type
```

In [8]:

```
print(list1)
print(list2)
print(list3)
```

```
['apple', 'banana', 'cherry']
[1, 5, 7, 9, 3]
[True, False, False]
```

In [9]:

```
#(heterogenous) list can contain different data types:

list1 = ["abc", 34, True, 40, "male"]
print(list1)
```

```
['abc', 34, True, 40, 'male']
```

In [10]:

```
type(list1)
```

Out[10]:

```
list
```

Accessing items of a List

The items in a list can be accessed through **indexing and slicing.**

In [11]:

```
list1 = ["abc", 34, True, 40, "male"]
print(list1)
```

```
['abc', 34, True, 40, 'male']
```

In [12]:

```
#positive or forward indexing
print(list1[0])
```

```
abc
```

In [13]:

```
print(list1[1])
```

```
34
```

In [14]:

```
print(list1[2])
```

```
True
```

In [15]:

```
print(list1[5])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-4183b00bcc53> in <module>
----> 1 print(list1[5])

IndexError: list index out of range
```

In [20]:

```
#Negative and backward indexing
list1 = ["abc", 34, True, 40, "male"]
print(list1[-1])
```

```
male
```

In [21]:

```
print(list1[-2])
```

```
40
```

In [22]:

```
print(list1[-4])
```

```
34
```

In [23]:

```
#To check position of an element in list
list1.index("male")
```

Out[23]:

```
4
```

In [24]:

```
list1.index(True)
```

Out[24]:

```
2
```

In [25]:

```
print(list1[a])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-2fc46ca457fa> in <module>
----> 1 print(list1[a])

NameError: name 'a' is not defined
```

In [26]:

```
#slicing
#list1[_:_:_] = list1[start:end:step]
list1 = ["abc", 34, True, 40, "male"]
print(list1[0:])
```

```
['abc', 34, True, 40, 'male']
```

In [27]:

```
print(list1[:3])
```

```
['abc', 34, True]
```

In [28]:

```
print(list1[:])
```

```
['abc', 34, True, 40, 'male']
```

In [29]:

```
print(list1[2:5])
```

```
[True, 40, 'male']
```

In [30]:

```
print(list1[::2])
```

```
['abc', True, 'male']
```

In [31]:

```
print(list1[::-1])
```

```
['male', 40, True, 34, 'abc']
```

In [32]:

```
print(list1[::-2])
```

```
['male', True, 'abc']
```

In [33]:

```
print(list1[1:-1])#skip 1st and last element
```

```
[34, True, 40]
```

In [34]:

```
print(list1[-4:5])
```

```
[34, True, 40, 'male']
```

In [35]:

```
print(list1[-4:-1])
```

```
[34, True, 40]
```

In [36]:

```
print(list1[-4:-1:2])
```

```
[34, 40]
```

In [37]:

```
print(list1[-1:-4])
```

```
[]
```

In [38]:

```
print(list1[1:-2])
```

```
[34, True]
```

In [39]:

```
print(list1[0:-4])
```

```
['abc']
```

In [40]:

```
print(list1[-1:-4:-2])
```

```
['male', True]
```

In [41]:

```
#print list 2 times
list1*2
```

Out[41]:

```
['abc', 34, True, 40, 'male', 'abc', 34, True, 40, 'male']
```

In [42]:

```
list1 = ["apple", "banana", ["watermelon", "kiwi"],"cherry"]
print(list1[2][0])
```

```
watermelon
```

In [43]:

```
list1 = ["apple", "banana", ["watermelon", "kiwi"],"cherry"]
print(list1[2][0][4])
```

```
r
```

In [44]:

```
list1 = ["apple", "banana", ["watermelon", ["mango","orange"],"coconut","kiwi"],"cherry"]
list1[2][1][0]
```

Out[44]:

```
'mango'
```

In [45]:

```
list1[2][1][1][0:3]
```

Out[45]:

```
'ora'
```

In [ ]:

```

```

Change elements/values in the list(Mutable)

In [46]:

```
a = ["apple","banana","cherry"]
print(a)
```

```
['apple', 'banana', 'cherry']
```

In [47]:

```
a[0] = "kiwi"
print(a)
```

```
['kiwi', 'banana', 'cherry']
```

In [48]:

```
b = ["apple","banana","cherry","mango"]
b[1:3]=["watermelon","kiwi"]
```

In [49]:

```
print(b)
```

```
['apple', 'watermelon', 'kiwi', 'mango']
```

In [50]:

```
b = ["apple","banana","cherry","mango"]
b[1:2] = ["papaya","orange"]
```

In [51]:

```
print(b)
```

```
['apple', 'papaya', 'orange', 'cherry', 'mango']
```

In [52]:

```
b = ["apple","banana","cherry","mango"]
b[1:4] = ["watermelon"]
```

In [53]:

```
print(b)
```

```
['apple', 'watermelon']
```

In [54]:

```
new = ['apple', 'banana', ['watermelon',["grape","mango"],'kiwi'], 'cherry']
print(new[2][1][0][-2])
```

```
p
```

In [57]:

```
print(new[2][0][4])
```

```
r
```

In [58]:

```
print(new[3][-2])
```

```
r
```

List Methods

Update & Delete from list

Add List Items

1. To insert a new list item, without replacing any of the existing values, we can use the insert() method.

In [59]:

```
a = ["apple","banana","cherry"]
a.insert(2, "watermelon")
print(a)
```

```
['apple', 'banana', 'watermelon', 'cherry']
```

In [60]:

```
b = ["apple","banana","cherry"]
b.insert(2, ["watermelon","kiwi"])
print(b)
```

```
['apple', 'banana', ['watermelon', 'kiwi'], 'cherry']
```

2. append() : Accept only one parameter and add it at the end of the list.

In [61]:

```
a = ["apple","banana","cherry"]
a.append("watermelon")
print(a)
```

```
['apple', 'banana', 'cherry', 'watermelon']
```

In [62]:

```
b = ["apple","banana","cherry"]
b.append(["watermelon","kiwi"])
print(b)
```

```
['apple', 'banana', 'cherry', ['watermelon', 'kiwi']]
```

3. extend() : Accept the list of elements and add them at the end of the list. we can even add another list by using this method

In [63]:

```
a = ["apple","banana","cherry"]
a.extend(["watermelon"])
print(a)
```

```
['apple', 'banana', 'cherry', 'watermelon']
```

In [64]:

```
b = ["apple","banana","cherry"]
b.extend(["watermelon","kiwi"])
print(b)
```

```
['apple', 'banana', 'cherry', 'watermelon', 'kiwi']
```

In [65]:

```
a = ["apple","banana","cherry"]
b = [1, 2, 3]
a.extend(b)
print(a)
```

```
['apple', 'banana', 'cherry', 1, 2, 3]
```

Remove List Items

4. The remove() method removes the specified item.

In [66]:

```
a = ["apple", "banana", "cherry"]
a.remove("banana")
print(a)
```

```
['apple', 'cherry']
```

In [67]:

```
b = ["True", False, True]
b.remove(False)
print(b)
```

```
['True', True]
```

In [68]:

```
a = ["apple", "banana", "cherry"]
a.remove("apple")
print(a)
```

```
['banana', 'cherry']
```

5. The pop() method removes the specified index.

In [69]:

```
a = ["apple", "banana", "cherry"]
a.pop(1)
print(a)
```

```
['apple', 'cherry']
```

In [70]:

```
#If you do not specify the index, the pop() method removes the last item.
a = ["apple", "banana", "cherry"]
a.pop()
print(a)
```

```
['apple', 'banana']
```

6. The del keyword also removes the specified index.

In [71]:

```
a = ["apple", "banana", "cherry"]
del a[1]
print(a)
```

```
['apple', 'cherry']
```

In [72]:

```
#The del keyword can also delete the list completely.
list1 = ["apple", "banana", "cherry"]
del list1
```

In [73]:

```
print(list1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-73-7fd613c3c7e4> in <module>
----> 1 print(list1)

NameError: name 'list1' is not defined
```

7. The clear() method empties the list. The list still remains, but it has no content.

In [74]:

```
list2 = ["apple", "banana", "cherry"]
list2.clear()
print(list2)
```

```
[]
```

Concatenate of two lists

The concatenation of two lists means merging of two lists.

1. using '+' operator
2. using extend()

In [75]:

```
my_list1 = [1, 2, 3]
my_list2 = [4, 5, 6]

# Using + operator
my_list3 = my_list1 + my_list2
print(my_list3)
```

```
[1, 2, 3, 4, 5, 6]
```

In [76]:

```
a = ["apple","banana","cherry"]
b = [1, 2, 3]
a.extend(b)
print(a)
```

```
['apple', 'banana', 'cherry', 1, 2, 3]
```

Copying a List

1. using '=' operator # deep copying(The changes that we make in the original list will be reflected in the new list.)
2. Using the copy() method
3. using the list() method

In [77]:

```
my_list1 = [1, 2, 3]
# Using = operator
new_list = my_list1
# printing the new list
print(new_list)
```

```
[1, 2, 3]
```

In [78]:

```
# making changes in the original list
my_list1.append(4)
print(my_list1)
print(new_list)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```

In [79]:

```
#using copy method
thislist = ["apple", "banana", "cherry"]
newlist = thislist.copy()
print(newlist)
```

```
['apple', 'banana', 'cherry']
```

In [80]:

```
thislist.append("mango")
print(thislist)
print(newlist)
```

```
['apple', 'banana', 'cherry', 'mango']
['apple', 'banana', 'cherry']
```

In [81]:

```
a = list((1,2,3))
print(a)
```

```
[1, 2, 3]
```

In [82]:

```
#copying list
#using list() method
thislist = ["apple", "banana", "cherry"]
newlist = list(thislist)
print(newlist)
```

```
['apple', 'banana', 'cherry']
```

In [83]:

```
thislist.append("kiwi")
print(thislist)
print(newlist)
```

```
['apple', 'banana', 'cherry', 'kiwi']
['apple', 'banana', 'cherry']
```

Sort and Reverse List

1. sort() : The sort function sorts the elements in the list in ascending order.
2. reverse() : The reverse function is used to reverse the elements in the list.

In [84]:

```
mylist = [3,5,7,2,4,8]
mylist.sort()
```

In [85]:

```
print(mylist)
```

```
[2, 3, 4, 5, 7, 8]
```

In [86]:

```
list1 = ['g','w','o','a','m']
list1.sort()
```

In [87]:

```
print(list1)
```

```
['a', 'g', 'm', 'o', 'w']
```

In [88]:

```
mylist = [2,4,6,8,10]
mylist.reverse()
```

In [89]:

```
print(mylist)
```

```
[10, 8, 6, 4, 2]
```

Functions used with list

1. max() : The max function returns the maximum value in the list.

In [90]:

```
mylist = [3, 4, 5, 6, 1]
print(max(mylist))
```

```
6
```

2. min() : The min function returns the minimum value in the list.

In [91]:

```
mylist = [3, 4, 5, 6, 1]
print(min(mylist))
```

```
1
```

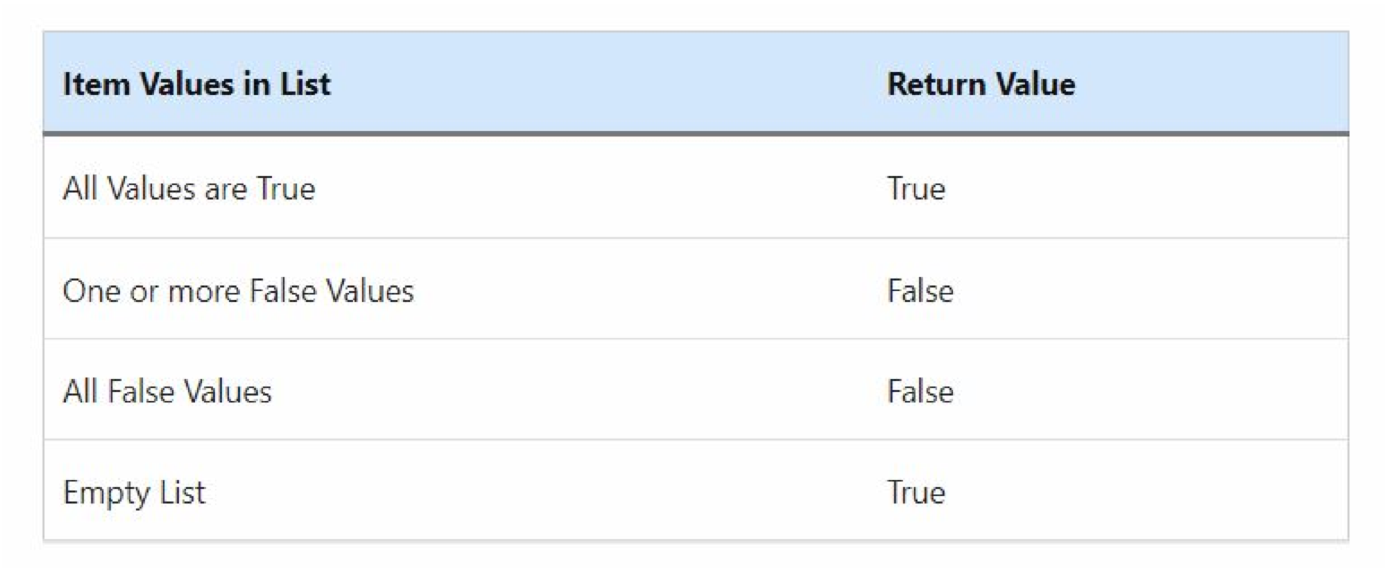
3. sum() : The sum function returns the sum of all the elements in the list.

In [92]:

```
mylist = [3, 4, 5, 6]
print(sum(mylist))
```

```
18
```

4 . all() :



Item Values in List	Return Value
All Values are True	True
One or more False Values	False
All False Values	False
Empty List	True

In [93]:

```
#with all true values
samplelist1 = [1,1,True]
print(all(samplelist1))
```

```
True
```

In [94]:

```
#with one false
samplelist2 = [0,1,True,1]
print(all(samplelist2))
```

```
False
```

In [95]:

```
#with all false
samplelist3 = [0,0,False]
print(all(samplelist3))
```

```
False
```

In [96]:

```
#with empty list
samplelist4 = []
print(all(samplelist4))
```

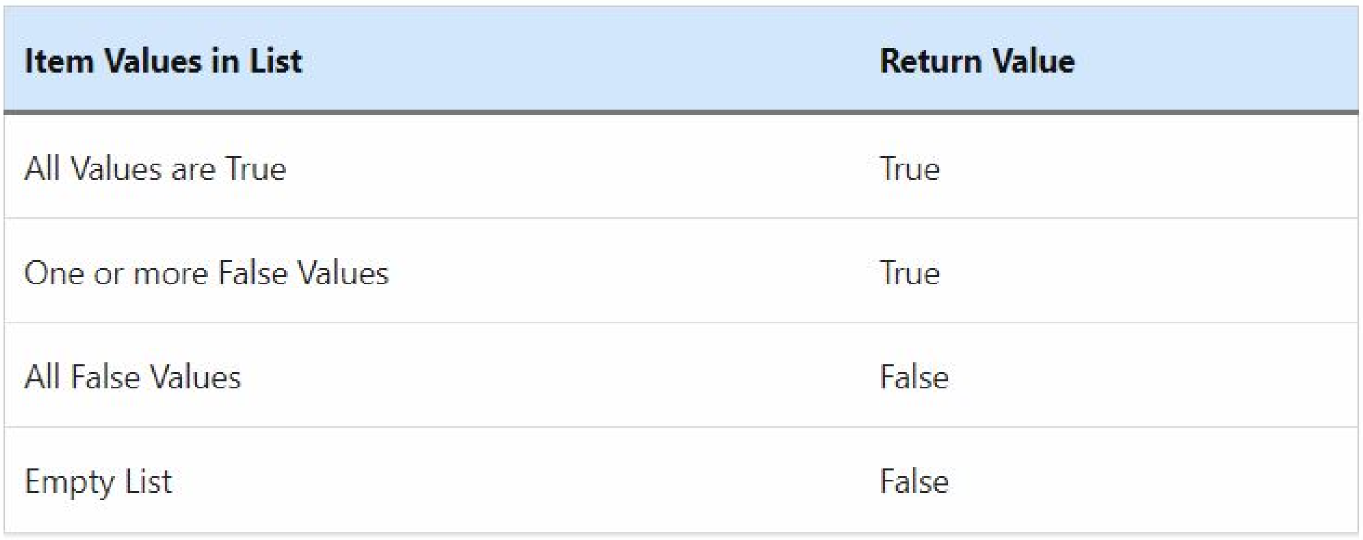
```
True
```

5. any() :

In [ ]:

```

```



Item Values in List	Return Value
All Values are True	True
One or more False Values	True
All False Values	False
Empty List	False

In [97]:

```
#with all true values
samplelist1 = [1,1,True]
print(any(samplelist1))
```

```
True
```

In [98]:

```
#with one false
samplelist2 = [0,1,True,1]
print(any(samplelist2))
```

```
True
```

In [99]:

```
#with all false
samplelist3 = [0,0,False]
print(any(samplelist3))
```

```
False
```

In [100]:

```
#with empty list
samplelist4 = []
print(any(samplelist4))
```

```
False
```

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

newlist = [expression for item in iterable if condition]

Type *Markdown* and LaTeX: α^2

In [101]:

```
#eg. 1
squares = []
for i in range(10):
  squares.append(i**2)
squares
```

Out[101]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [102]:

```
squares = [i**2 for i in range(10)]
squares
```

Out[102]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [103]:

```
a = []
for i in range(1,6):
  a.append(i*2)
a
```

Out[103]:

```
[2, 4, 6, 8, 10]
```

In [104]:

```
a = [i*2 for i in range(1,6)]
a
```

Out[104]:

```
[2, 4, 6, 8, 10]
```

In [105]:

```
#Based on a list of fruits, you want a new list, containing only the fruits with the letter
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [106]:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [107]:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
    newlist.append(x.upper())
print(newlist)
```

```
['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
```

In [108]:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x.upper() for x in fruits]
print(newlist)
```

```
['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
```

Implementation of Stack and Queue using List

Stack : A stack is a linear data structure that stores items in a Last-In/First-Out (LIFO). In stack, a new element is added at one end and an element is removed from that end only.

In [109]:

```
#working of append() and pop() function:
# stack using list
stack = ["Amar", "Akbar", "Anthony"]
stack.append("Ram")
stack.append("Iqbal")
print(stack)
```

```
['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']
```

In [110]:

```
# Removes the last item
print(stack.pop())
print(stack.pop())
```

```
Iqbal
Ram
```

In [111]:

```
print(stack)
```

```
['Amar', 'Akbar', 'Anthony']
```

In [112]:

```
fruits = []

# Let's push/add some fruits into our list
fruits.append('banana')
fruits.append('grapes')
fruits.append('mango')
fruits.append('orange')
print(fruits)

# Now let's pop our fruits, we should get 'banana'
first_item = fruits.pop()
print(first_item)
print(fruits)

# If we pop again we'll get 'grapes'
first_item = fruits.pop()
print(first_item)


# 'mango' and 'orange' remain
print(fruits)
```

```
['banana', 'grapes', 'mango', 'orange']
orange
['banana', 'grapes', 'mango']
mango
['banana', 'grapes']
```

Queue : Queue works on the principle of “First-in, first-out”.

In [113]:

```
# Queue using list
queue = ["Amar", "Akbar", "Anthony"]
queue.append("Ram")
queue.append("Iqbal")
print(queue)

# Removes the first item
print(queue.pop(0))

print(queue)

# Removes the first item
print(queue.pop(0))

print(queue)
```

```
['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']
Amar
['Akbar', 'Anthony', 'Ram', 'Iqbal']
Akbar
['Anthony', 'Ram', 'Iqbal']
```

In [114]:

```
fruits = []

# Let's enqueue/add some fruits into our list
fruits.append('banana')
fruits.append('grapes')
fruits.append('mango')
fruits.append('orange')
print(fruits)
# Now let's dequeue/pop our fruits, we should get 'banana'
first_item = fruits.pop(0)
print(first_item)
print(fruits)

# If we dequeue again we'll get 'grapes'
first_item = fruits.pop(0)
print(first_item)

# 'mango' and 'orange' remain
print(fruits)
```

```
['banana', 'grapes', 'mango', 'orange']
banana
['grapes', 'mango', 'orange']
grapes
['mango', 'orange']
```

Use of zip()

In [115]:

```
#Python zip two lists
name = [ "Manjeet", "Nikhil", "Shambhavi", "Astha" ]
roll_no = [ 4, 1, 3, 2 ]

# using zip() to map values
mapped = zip(name, roll_no)
mapped = list(mapped)
print(mapped)
```

```
[('Manjeet', 4), ('Nikhil', 1), ('Shambhavi', 3), ('Astha', 2)]
```

In [116]:

```
# Using list constructor
my_list2 = list((1, 2, 3))
print(my_list2)
```

```
[1, 2, 3]
```

In [117]:

```
# initializing lists
name = ["Manjeet", "Nikhil", "Shambhavi", "Astha"]
roll_no = [4, 1, 3, 2]
marks = [40, 50, 60, 70]

# using zip() to map values
mapped = zip(name, roll_no, marks)

# converting values to print as list
mapped = list(mapped)
print(mapped)
```

```
[('Manjeet', 4, 40), ('Nikhil', 1, 50), ('Shambhavi', 3, 60), ('Astha', 2, 7
0)]
```

In [118]:

```
# unzipping values
namez, roll_noz, marksz = zip(*mapped)
print(namez)
print(roll_noz)
print(marksz)
```

```
('Manjeet', 'Nikhil', 'Shambhavi', 'Astha')
(4, 1, 3, 2)
(40, 50, 60, 70)
```

Matrix operations using List

In [119]:

```
# Program to add two matrices using nested loop

X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

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

result = [[0,0,0],
         [0,0,0],
         [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X)):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

```
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
```

In [120]:

```
# Program to add two matrices using list comprehension

X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

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

result = [[X[i][j] + Y[i][j]  for j in range(len(X[0]))] for i in range(len(X))]

for r in result:
   print(r)
```

```
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
```

In [121]:

```
# Program to multiply two matrices using nested loops

# 3x3 matrix
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]
# result is 3x4
result = [[0,0,0,0],
         [0,0,0,0],
         [0,0,0,0]]

# iterate through rows of X
for i in range(len(X)):
   # iterate through columns of Y
   for j in range(len(Y[0])):
       # iterate through rows of Y
       for k in range(len(Y)):
           result[i][j] += X[i][k] * Y[k][j]

for r in result:
   print(r)
```

```
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
```