T3-B
Roll.No:49

MUHAMMED HISHAM V.

SCIENTIFIC COMPUTING LAB

PRELIMINARY LAB REPORT

Lab 2: Familiarization of Scientific Computing.

Q1) Write a short note on different arithmetic functions such as abs, sin(), real(), imag(), complex() and sinc() in python.

i) abs( ) in Python.

The abs() function in python is used to return the absolute value of a number.

Syntax:
abs(number)
number : Can either be an integer, a floating point number or a complex number

ii) sin( ) in Python

In Python, math module contains a number of mathematical operations, which can be performed with ease using the module. math.sin() function returns the sine of value passed as argument. The value passed in this function should be in radians.

Syntax:
math.sin(x)
x : value to be passed to sin()
Returns: Returns the sine of value passed as argument

iii) real( ) in Python

The real part of a complex number can be accessed using the function real().

Syntax:
a.real()
a : The complex number of which real part is to be found.

iv) imag( ) in Python

The imaginary part of a complex number can be accessed using the function imag().

Syntax:

a.imag()
a : The complex number of which imaginary part is to be found.

v) complex( ) in Python

Python converts the real numbers x and y into complex using the function complex(x,y).

It takes two arguments where first one is the real part of the complex number and the second one is the imaginary part of the complex number. It returns the complex number.

Syntax :
z = complex(x,y)
Where
x : real part
y : imaginary part
z : complex number returned

vi) sinc( )

It is a function from NumPy module. Therefore to use it we should import NumPy module. This mathematical function in python helps user to calculate sinc function for all x(being the array elements). It takes an array of numbers and returns their sinc values. The input should be in radians.

Syntax :
arr = numpy.sinc(array)
arr : list of sinc values
array : Input list of numbers in radian.

Q2) Write a short note on vectorized computation in python.

In the context of high-level languages like Python, Matlab, and R, the term vectorization describes the use of optimized, pre-compiled code written in a low-level language (eg. C) to perform mathematical operations over a sequence of data. This is done in place of an explicit iteration written in the native language code (eg. a “for-loop” written in Python).

Vectorization is used to speed up the Python code without using loop. Using such a function can help in minimizing the running time of code efficiently. Various operations are being performed over vector such as dot product of vectors

which is also known as scalar product as it produces single output, outer products which results in square matrix of dimension equal to length X length of the vectors, Element wise multiplication which products the element of same indexes and dimension of the matrix remain unchanged.

Time complexity in the execution of any algorithm is very crucial deciding whether an application is reliable or not. To run a large algorithm in as much as optimal time possible is very important when it comes to real-time application of output. To do so, Python has some standard mathematical functions for fast operations on entire arrays of data without having to write loops. One of such library which contains such function is NumPy.

NumPy provides highly-optimized functions for performing mathematical operations on arrays of numbers. Performing extensive iterations (eg.via 'for-loops') in Python to perform repeated mathematical computations should nearly always be replaced by the use of vectorized functions on arrays. This informs the entire design paradigm of NumPy.

Q3) Write the algorithm/ flowchart for the experiments.

Experiment-1

Aim: To compute the factorial of an integer using a function.

Algorithm:

1 : Start
2 : Define function factorial(n)
   i. If n==1 then return 1
   ii. else
   iii. f = n * factorial(n – 1)
   iv. Return f
3 : Input a number and store it in 'n'
4 : Print factorial(n)
5 : Stop

Experiment-2

Aim: To compute the sum of first 'N' Fibonacci numbers using function.

Algorithm:

1 : Start

2 : Define function fsum(n)

- a =0, b=1
- if n==1 return 0
- elif n==2 return 1
- else repeat steps i – iv for n != 0
    i. c = a + b
    ii. sum = sum + c
    iii. a = b
    iv. b = c
    v. n = n-1
- return sum

3: Input an integer and store it in variable 'n'

4: Print fsum(n)

5:Stop

Experiment-3

Aim: Algorithm to Represent a complex number in python using built in function, display it's real part, imaginary part and absolute value using built in functions.

Algorithm:

1 : Start

2 : Import module 'cmath'

3 : Input 'x' as real part and 'y' as imaginary part.

4 : z = complex(x, y) #representing as a complex number.

5 : Print the real part using z.real()

6 : Print the imaginary part using z.imag()

7 : Stop

Experiment-4

Aim: To Plot a sine wave in python with frequency 100 Hz and sampling rate fs = 10000 Hz.

Algorithm:

1 : Start

2 : Import libraries numpy as np and matplotlib.pyplot as plot
3 : Assign sampling frequency fs = 10000 and message frequency fm = 100
4 : Discrete frequency, w = 2 * np.pi * fm/fs
5 : Get x values of the sine wave, time = np.arange(0, 100, 0.1)
5 : Plot a sine wave using time and amplitude obtained for the sine wave.
6 : Give a title,x-axis label and y-axis label for the sine wave plot
7 : Display the sine wave using plot.show().
8 : Stop.

Experiment-5

Aim: To plot a sinc function in python.

Algorithm:

1 : Start
2 : Import libraries numpy as np and matplotlib.pyplot as plt
3 : Define x axis using linespace() function in numpy as x = np.linespace(-2 * np.pi, 2 * np.pi , 500)
4 : Define y axis using sinc() function as y = np.sinc(x)
5 : Initialize the figure with figure( ) function in matplotlib.pyplot
6 : Plot the figure with plot( ) function
7 : Display the figure with show( ) function.
8 : Stop.

Experiment-6

Experiment-6.a.1

Aim : Add given two matrices using **non-vectorized** method.

Algorithm:

1 : Start
2 : Initialize x and y with two given matrices.
3 : i = 0,j = 0,z=[[]]
4 : while i < len(x) repeat steps 4 - 7
5 : while j < len(x[0]) repeat steps 5 - 6
6 : z[i][j] = x[i][j] + y[i][j]
7 : j = j + 1

8 : i = i + 1
9: Print z
10:Stop

Experiment-6.a.2

Aim : Add given two matrices using **vectorized** method.

Algorithm:

1 : Start
2 : Import library numpy
3 : Initialize x and y using numpy.array() method
4 : Add x and y using numpy.add() method with x and y as arguments.
5 : Print numpy.add(x, y)
6 : Stop

Experiment-6.b.1

Aim : Add '2' to each element in the first row of a matrix using **non-vectorized** method.

Algorithm:

1 : Start
2 : Initialize y with the given matrix and i = 0
3 : while i < 2 repeat steps 4 to 7
4 : while j < len(y[0]) repeat steps 5 to 6
5 : y[i][j] = y[i][j] + 2
6 : j = j +1
7 : i = i + 1
8 : Print y
9 : Stop

Experiment-6.b.2

Aim : Add '2' to each element in the first row of a matrix using **vectorized** method.

Algorithm:

1 : Start
2 : Import library numpy

3 : Initialize y using numpy.array() method
4 : y[0] = y[0] + 2
5 : Print y
6 : Stop

Experiment-6.c.1

Aim : Find the sum of all elements in a matrix using **non-vectorized** method.

Algorithm:

1 : Start
2 : Initialize x with the given matrix.
3 : i = 0,j = 0,sum=0
4 : while i < len(x) repeat steps 4 - 7
5 : while j < len(x[0]) repeat steps 5 - 6
6 : sum = sum + x[i][j]
7 : j = j + 1
8 : i = i + 1
9: Print sum
10: Stop

Experiment-6.c.2

Aim : Find the sum of all elements in a matrix using **vectorized** method.

Algorithm:

1 : Start
2 : Import library numpy
3 : Initialize y using numpy.array() method
4 : sum = numpy.sum(x)
5 : Print sum
6 : Stop

Experiment-6.d.1

Aim : Find a 1D array by summing over the rows within each column of a matrix using **non-vectorized** method.

Algorithm:

1 : Start
2 : Initialize y with the given matrix.
3 : i = 0,j = 0
4 : while i < len(x) repeat steps 4 – 10
5 : sum = 0
6 : while j < len(x[0]) repeat steps 6 - 8
7 : sum = sum + y[j][i]
8 : j = j + 1
9 : i = i + 1
10 : Insert sum into a list l
11:Print l
12 : Stop

Experiment-6.d.2

Aim : Find a 1D array by summing over the rows within each column of a matrix using **vectorized** method.

Algorithm:

1 : Start
2 : Import library numpy
3 : Initialize y using numpy.array() method
4 : n = numpy.sum(y, axis=0)
5 : Print n
6 : Stop

Experiment-6.e.1

Aim : Find element by element multiplication of two matrices using **non-vectorized** method.

Algorithm:

1 : Start
2 : Initialize x and y with two given matrices.
3 : i = 0,j = 0,z=[[]]
4 : while i < len(x) repeat steps 4 - 7
5 : while j < len(x[0]) repeat steps 5 - 6

6 : z[i][j] = x[i][j] * y[i][j]
7 : j = j + 1
8 : i = i + 1
9: Print z
10:Stop

Experiment-6.e.2

Aim : Find element by element multiplication of two matrices using **vectorized** method.

Algorithm:

1 : Start
2 : Import library numpy
3 : Initialize x and y using numpy.array() method
4 : Multiply element-wise x and y using numpy.multiply() method with x and y as arguments.
5 : Print numpy.multiply(x, y)
6 : Stop.

T3-B
Roll.No:49

MUHAMMED HISHAM V.

SCIENTIFIC COMPUTING LAB

FINAL LAB REPORT

Lab 2: Familiarization of Scientific Computing.

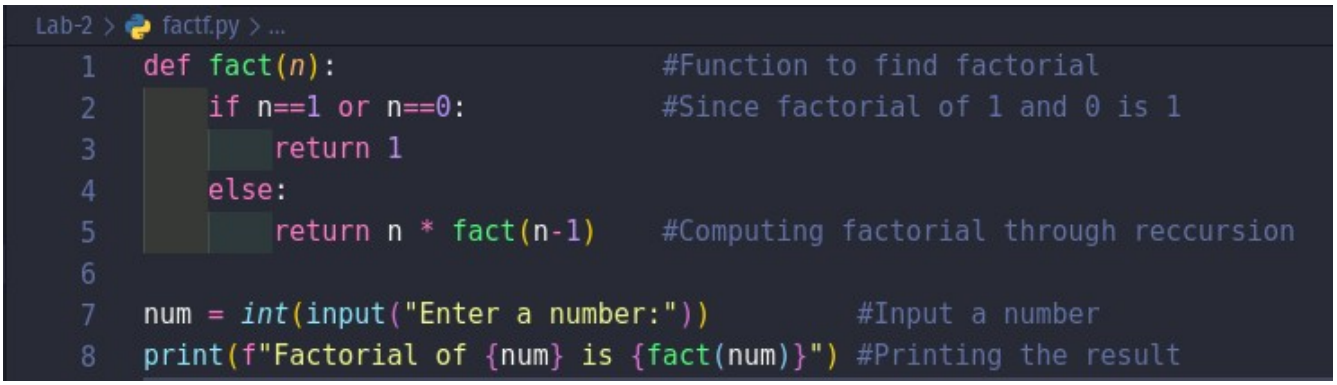
Experiment-1

Aim: To compute the factorial of an integer using a function.

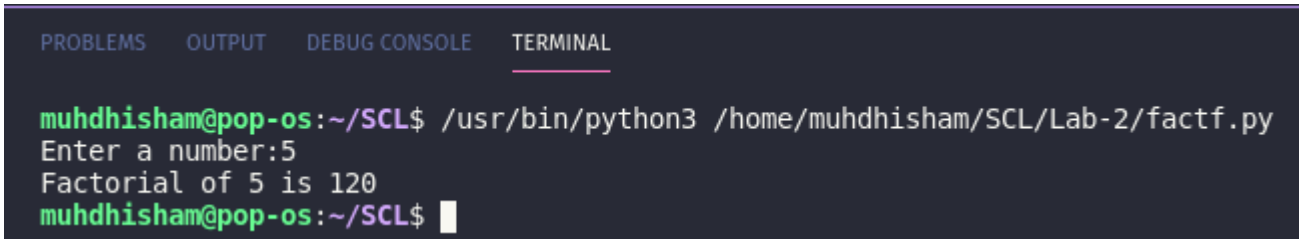
CODE:

```
def fact(n):                        #Function to find factorial
      if n==1 or n==0:          #Since factorial of 1 and 0 is 1
            return 1
      else:
            return n * fact(n-1)    #Computing factorial through reccursion

num = int(input("Enter a number:"))           #Input a number
print(f"Factorial of {num} is {fact(num)}")    #Printing the result
```



OUTPUT:

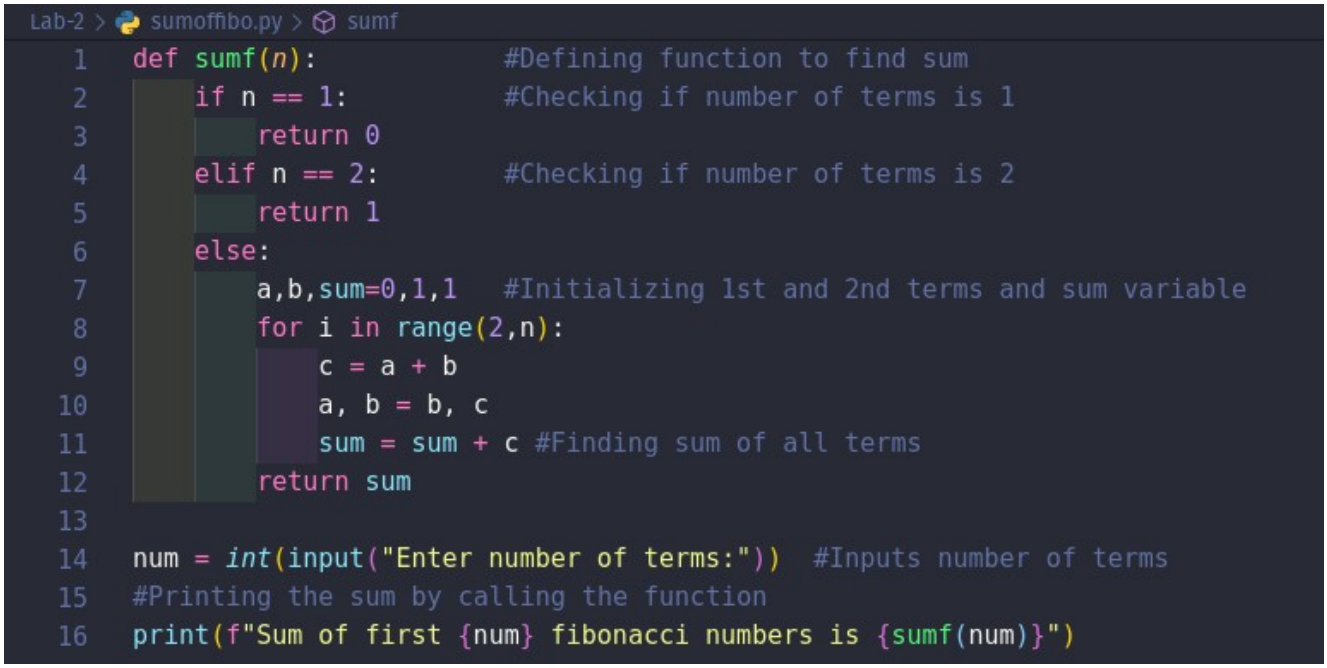


Experiment-2

Aim: To compute the sum of first 'N' Fibonacci numbers using function.

CODE:

```
def sumf(n):            #Defining function to find sum
      if n == 1:        #Checking if number of terms is 1
            return 0
      elif n == 2:      #Checking if number of terms is 2
            return 1
      else:
            a,b,sum=0,1,1           #Initializing 1st and 2nd terms and sum variable
            for i in range(2,n):
                  c = a + b
                  a, b = b, c
                  sum = sum + c     #Finding sum of all terms
            return sum
num = int(input("Enter number of terms:"))     #Inputs number of terms
#Printing the sum by calling the function
print(f"Sum of first {num} fibonacci numbers is {sumf(num)}")
```



OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/sumoffibo.py
Enter number of terms:5
Sum of first 5 fibonacci numbers is 7
muhdhisham@pop-os:~/SCL$
```

Experiment-3

Aim : Algorithm to Represent a complex number in python using built in function, display it's real part, imaginary part and absolute value using built in functions.

CODE:

```
r = int(input("Enter real part:"))                #Input real part
i = int(input("Enter imaginary part:"))     #Input imaginary part
z = complex(r, i)                                 #Converting into complex number
print(f"Complex number is {z}")                 #Printing the complex number
print(f"Absolute value of {z} is {abs(z)}") #Printing absolute value of complex number
print(f"Real part of complex number is {z.real}") #Printing real part of complex number
print(f"Imaginary part of complex number is {z.imag}")#Printing imaginary part of
                                                                                    complex
number
```

```
Lab-2 > complex.py > r
1  r = int(input("Enter real part:"))                #Input real part
2  i = int(input("Enter imaginary part:"))           #Input imaginary part
3  z = complex(r, i)                                 #Converting into complex number
4  print(f"Complex number is {z}")                   #Printing the complex number
5  print(f"Absolute value of {z} is {abs(z)}")      #Printing absolute value of complex number
6  print(f"Real part of complex number is {z.real}") #Printing real part of complex number
7  print(f"Imaginary part of complex number is {z.imag}")#Printing imaginary part of complex number
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/complex.py
Enter real part:4
Enter imaginary part:6
Complex number is (4+6j)
Absolute value of (4+6j) is 7.211102550927978
Real part of complex number is 4.0
Imaginary part of complex number is 6.0
muhdhisham@pop-os:~/SCL$
```

Experiment-4

Aim: To Plot a sine wave in python with frequency 100 Hz and sampling rate fs = 10000Hz.

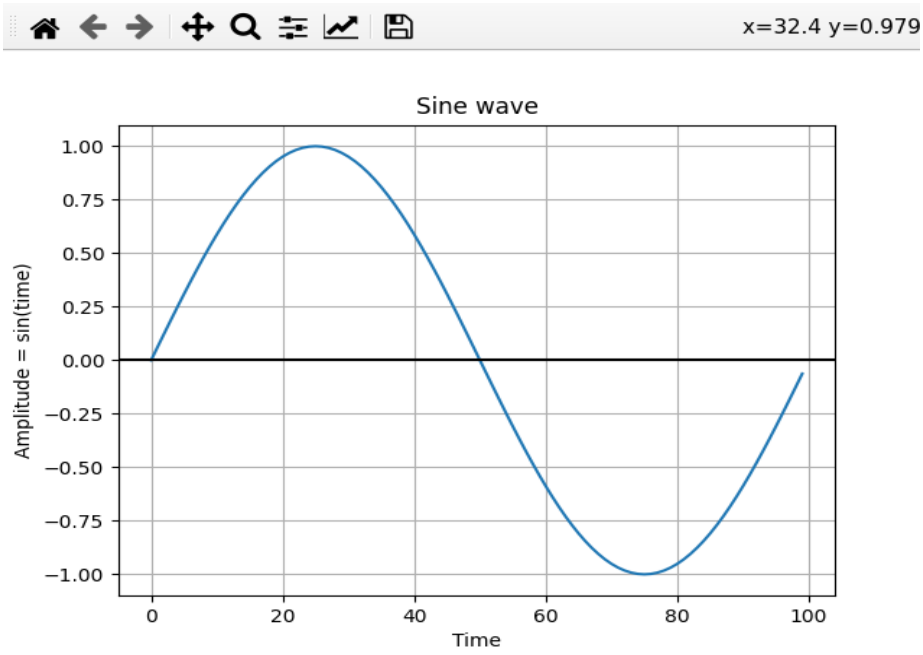
CODE:

```
import numpy as np
import matplotlib.pyplot as plot
fs = 10000                          #sampling freq
fm = 100                            #message freq
w = 2 * np.pi * fm / fs             #discrete frequency
N = 100                             #time period
time = np.arange(0, N, 1);          #x axis values
amplitude = np.sin( w * time)       #y axis values
plot.plot(time, amplitude)          #plot
plot.title('Sine wave')             #Give a title for the sine wave plot
plot.xlabel('Time')                 #Give x axis label for the sine wave plot
plot.ylabel('Amplitude = sin(time)') #Give y axis label for the sine wave plot
plot.grid(True, which='both')
plot.axhline(y=0, color='k')
plot.show()
```

Lab-2 > sinewave.py > ...

```
1   import numpy as np
2   import matplotlib.pyplot as plot
3   fs = 10000                          # sampling freq
4   fm = 100                            # message freq
5   w = 2 * np.pi * fm / fs             # discrete frequency
6   N = 100                             # time period
7   time = np.arange(0, N, 1);          #x axis values
8   amplitude = np.sin( w * time)       #y axis values
9   plot.plot(time, amplitude)          # plot
10  plot.title('Sine wave')             # Give a title for the sine wave plot
11  plot.xlabel('Time')                 # Give x axis label for the sine wave plot
12  plot.ylabel('Amplitude = sin(time)') # Give y axis label for the sine wave plot
13  plot.grid(True, which='both')
14  plot.axhline(y=0, color='k')
15  plot.show()
```

OUTPUT:



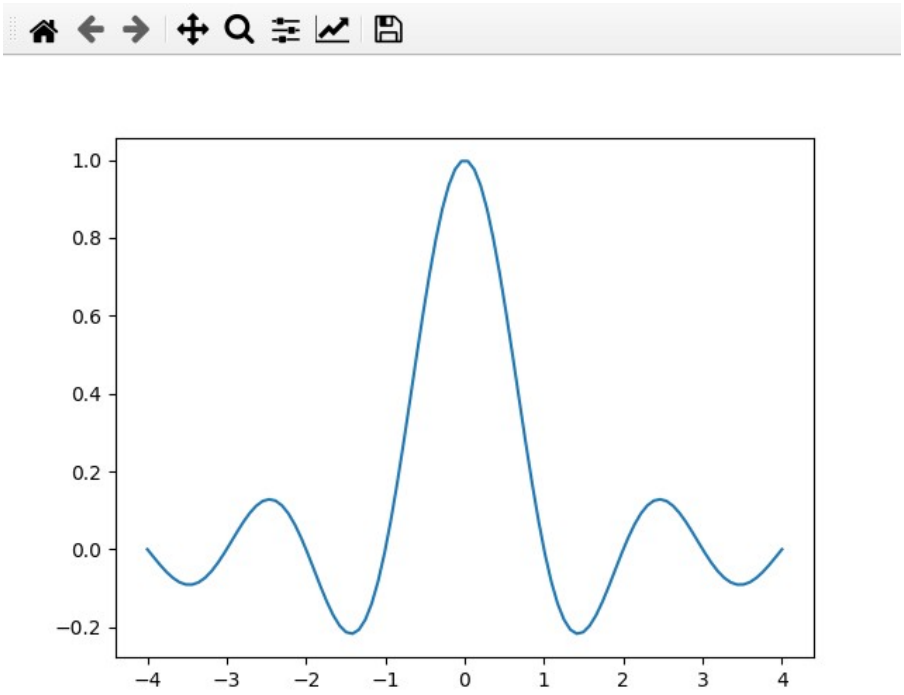
Experiment-5

Aim: To plot a sinc function in python.

CODE:

```
import numpy as np
from matplotlib.pyplot import plot, show
x = np.linspace(-4, 4, 100)     #x axis values
vals = np.sinc(x)               #y axis values => sinc
plot(x, vals)                   #plots the graph
show()                      #displays the figure
```

OUTPUT:



Experiment-6

Experiment-6.a.1

Aim : Add given two matrices using non-vectorized method.

CODE:

```
x =   [[0,1,2], #Initializing MATRIX-1
      [3,4,5],
      [6,7,8]]
y =   [[-4,-3,-2], #Initializing MATRIX-2
      [ 0, 2, 1],
      [ 1, 4, 2]]
sum=[[0,0,0], #Iitializing SUM MATRIX
      [0,0,0],
      [0,0,0]]
for i in range(len(x)): #Iterating through row of the matrix
for j in range(len(x[0])): #Iterating through column of matrix
sum[i][j] = x[i][j] + y [i][j] #Finding sum and storing result into another matrix
print("Sum of the matrices:")
for s in sum: #Printing the sum matrix
print(s)
```

Lab-2 > v_add-matrix.py > x

```python
x =[[0,1,2],            #Initializing MATRIX-1
    [3,4,5],
    [6,7,8]]
y =[[-4,-3,-2],         #Initializing MATRIX-2
    [ 0, 2, 1],
    [ 1, 4, 2]]
sum=[[0,0,0],           #Iitializing SUM MATRIX
    [0,0,0],
    [0,0,0]]
for i in range(len(x)):                 #Iterating through row of the matrix
    for j in range(len(x[0])):          #Iterating through column of matrix
        sum[i][j] = x[i][j] + y [i][j]  #Finding sum and storing result into another matrix
print("Sum of the matrices:")
for s in sum:                           #Printing the sum matrix
    print(s)
```

OUTPUT:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/v_add-matrix.py
Sum of the matrices:
[-4, -2, 0]
[3, 6, 6]
[7, 11, 10]
muhdhisham@pop-os:~/SCL$
```

Experiment-6.a.2

Aim : Add given two matrices using **vectorized** method.

CODE:

```python
import numpy as np
x = np.array([ [0,1,2], #Initializing MATRIX-1
               [3,4,5],
               [6,7,8]])
y = np.array([ [-4,-3,-2], #Initializing MATRIX-2
               [ 0, 2, 1],
               [ 1, 4, 2]])
print("Sum of matrices:\n",np.add(x,y)) #Printing Result
```

```
Lab-2 > vect_add-matrix.py > ...
1  import numpy as np
2  x = np.array([  [0,1,2],          #Initializing MATRIX-1
3                  [3,4,5],
4                  [6,7,8]])
5  y = np.array([  [-4,-3,-2],       #Initializing MATRIX-2
6                  [ 0, 2, 1],
7                  [ 1, 4, 2]])
8  print("Sum of matrices:\n",np.add(x,y)) #Printing Result
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/vect_add-matrix.py
Sum of matrices:
 [[-4 -2  0]
 [ 3  6  6]
 [ 7 11 10]]
muhdhisham@pop-os:~/SCL$
```

Experiment-6.b.1

Aim : Add '2' to each element in the first row of a matrix using **non-vectorized** method.

CODE:

```
y =   [[-4,-3,-2],                  #Initializing MATRIX-y
      [ 0, 2, 1],
      [ 1, 4, 2]]
for i in range(1):                  #Iterating through row of the matrix
      for j in range(len(y[0])):    #Iterating through column of matrix
            y[i][j] = y [i][j] + 2  #Adding two to each element of 1st row

print("MATRIX-y after addition:")  #Printing the resultant
for i in y:
      print(i)
```

```
Lab-2 > v_add2.py > ...
1   y =[[-4,-3,-2],                    #Initializing MATRIX-y
2       [ 0, 2, 1],
3       [ 1, 4, 2]]
4   for i in range(1):                 #Iterating through row of the matrix
5       for j in range(len(y[0])):     #Iterating through column of matrix
6           y[i][j] = y [i][j] + 2     #Adding two to each element of 1st row
7
8   print("MATRIX-y after addition:")  #Printing the resultant
9   for i in y:
10      print(i)
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/v_add-matrix.py
Sum of the matrices:
[-4, -2, 0]
[3, 6, 6]
[7, 11, 10]
muhdhisham@pop-os:~/SCL$
```

Experiment-6.b.2

Aim : Add '2' to each element in the first row of a matrix using vectorized method.

CODE:

```
import numpy as np
y  =  np.array([[-4,-3,-2],                    #Initializing MATRIX-y
                [ 0, 2, 1],
                [ 1, 4, 2]])
y[0] = y[0] + 2                                #Adding 2 to elements in 1st row
print("Matrix after adding 2 in first row:")   #Printing final matrix
for i in y:
    print(i)                                   #Printing resultant matrix
```

```
Lab-2 > vect_add2.py > ...
1  import numpy as np
2  y = np.array(  [[-4,-3,-2],          #Initializing MATRIX-y
3                 [ 0, 2, 1],
4                 [ 1, 4, 2]])
5  y[0] = y[0] + 2                      #Adding 2 to elements in 1st row
6  print("Matrix after adding 2 in first row:")    #Printing final matrix
7  for i in y:
8      print(i)                         #Printing resultant matrix
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/vect_add2.py
Matrix after adding 2 in first row:
[-2 -1  0]
[0 2 1]
[1 4 2]
muhdhisham@pop-os:~/SCL$
```

Experiment-6.c.1

Aim : Find the sum of all elements in a matrix using **non-vectorized** method.

CODE:

```
x =   [[0,1,2],          #Initializing MATRIX-x
      [3,4,5],
      [6,7,8]]
sum = 0
for i in range(len(x)):              #Iterating through row of the matrix
      for j in range(len(x[0])):     #Iterating through column of matrix
            sum = x[i][j] + sum
print("\nSum of Elements of Matrix is",sum)    #Printing Sum
```

```
Lab-2 > v_allsum.py > ...
1  x =[[0,1,2],                              #Initializing MATRIX-x
2      [3,4,5],
3      [6,7,8]]
4  sum = 0
5  for i in range(len(x)):                   #Iterating through row of the matrix
6      for j in range(len(x[0])):            #Iterating through column of matrix
7          sum = x[i][j] + sum
8  print("\nSum of Elements of Matrix is",sum)   #Printing Sum
```

OUTPUT:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/v_allsum.py
Sum of Elements of Matrix is 36
muhdhisham@pop-os:~/SCL$
```

Experiment-6.c.2

Aim : Find the sum of all elements in a matrix using **vectorized** method.

CODE:

```
import numpy as np
x = np.array([ [0,1,2], #Initializing MATRIX-x
               [3,4,5],
               [6,7,8]])
print("Sum of elements of the matrix",np.sum(x)) #Print Sum
```

```
Lab-2 > vect_allsum.py > ...
1  import numpy as np
2  x = np.array([  [0,1,2],          #Initializing MATRIX-x
3                  [3,4,5],
4                  [6,7,8]])
5  print("Sum of elements of the matrix",np.sum(x))    #Print Sum
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/vect_allsum.py
Sum of elements of the matrix 36
muhdhisham@pop-os:~/SCL$
```

Experiment-6.d.1

Aim : Find a 1D array by summing over the rows within each column of a matrix using **non-vectorized** method.

CODE:

```
y =[ [-4,-3,-2],                    #Initializing MATRIX-y
     [ 0, 2, 1],
     [ 1, 4, 2]]
r = [0,0,0]
for i in range(len(y)):                    #Iterating through row of the matrix
     sum = 0
     for j in range(len(y[0])):            #Iterating through column of the matrix
          r[i] = r[i] + y[j][i]            #Finding sum of each column
print('\n',r)                              #Printing new 1-D matrix
```

Lab-2 > v_colsum.py > ...

```
1  y =[[-4,-3,-2],                    #Initializing MATRIX-y
2      [ 0, 2, 1],
3      [ 1, 4, 2]]
4  r = [0,0,0]
5  for i in range(len(y)):             #Iterating through row of the matrix
6      sum = 0
7      for j in range(len(y[0])):      #Iterating through column of the matrix
8          r[i] = r[i] + y[j][i]       #Finding sum of each column
9  print('\n',r)                       #Printing new 1-D matrix
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/v_colsum.py
 [-3, 3, 1]
muhdhisham@pop-os:~/SCL$
```

Experiment-6.d.2

Aim : Find a 1D array by summing over the rows within each column of a matrix using **vectorized** method.

CODE:

```
import numpy as np
y = np.array( [[-4,-3,-2], #Initializing MATRIX-y
[ 0, 2, 1],
[ 1, 4, 2]])
print("Resultant 1-D Matrix is",np.sum(y, axis=0)) #Printing the resultant matrix
```

Lab-2 > vect_colsum.py > ...

```
1  import numpy as np
2  y = np.array(  [[-4,-3,-2],         #Initializing MATRIX-y
3                  [ 0, 2, 1],
4                  [ 1, 4, 2]])
5  print("Resultant 1-D Matrix is",np.sum(y, axis=0)) #Printing the resultant matrix
```

OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/vect_colsum.py
Resultant 1-D Matrix is [-3  3  1]
muhdhisham@pop-os:~/SCL$
```

Experiment-6.e.1

Aim : Find element by element multiplication of two matrices using **non-vectorized** method.

CODE:

```
x =[ [0,1,2],          #Initializing MATRIX-1
     [3,4,5],
     [6,7,8]]
y =[ [-4,-3,-2],             #Initializing MATRIX-2
     [ 0, 2, 1],
     [ 1, 4, 2]]
pro=[[0,0,0],          #Initializing Resultant MATRIX
     [0,0,0],
     [0,0,0]]
for i in range(len(x)):      #Iterating through row of the matrix
     for j in range(len(x[0])): #Iterating through column of matrix
```

```
            pro[i][j] = x[i][j] * y [i][j] #Finding product and storing result into
                                                                another matrix
print("Element vise product of the matrices:")
for p in pro: #Printing the Resultant matrix
      print(p)
```

Lab-2 > v_multi.py > x

```
1   x =[[0,1,2],            #Initializing MATRIX-1
2       [3,4,5],
3       [6,7,8]]
4   y =[[-4,-3,-2],         #Initializing MATRIX-2
5       [ 0, 2, 1],
6       [ 1, 4, 2]]
7   pro=[[0,0,0],           #Iitializing Resultant MATRIX
8       [0,0,0],
9       [0,0,0]]
10  for i in range(len(x)):                 #Iterating through row of the matrix
11      for j in range(len(x[0])):          #Iterating through column of matrix
12          pro[i][j] = x[i][j] * y [i][j]  #Finding product and storing result into another matrix
13  print("Element vise product of the matrices:")
14  for p in pro:                           #Printing the Resultant matrix
15      print(p)
```

OUTPUT:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/v_multi.py
Element vise product of the matrices:
[0, -3, -4]
[0, 8, 5]
[6, 28, 16]
muhdhisham@pop-os:~/SCL$
```

Experiment-6.e.2

Aim : Find element by element multiplication of two matrices using **vectorized** method.

CODE:

```
import numpy as np
x = np.array([ [0,1,2], #Initializing MATRIX-1
[3,4,5],
[6,7,8]])
y = np.array([ [-4,-3,-2], #Initializing MATRIX-2
[ 0, 2, 1],
```

```
[ 1, 4, 2]])
#Printing the product
print("Element vise product of the matrices:\n",np.multiply(x,y))
```

Lab-2 > vect_multi.py > ...

```
1  import numpy as np
2  x = np.array([  [0,1,2],           #Initializing MATRIX-1
3                  [3,4,5],
4                  [6,7,8]])
5  y = np.array([  [-4,-3,-2],        #Initializing MATRIX-2
6                  [ 0, 2, 1],
7                  [ 1, 4, 2]])
8  #Printing the product
9  print("Element vise product of the matrices:\n",np.multiply(x,y))
```

OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
muhdhisham@pop-os:~/SCL$ /usr/bin/python3 /home/muhdhisham/SCL/Lab-2/vect_multi.py
Element vise product of the matrices:
 [[ 0 -3 -4]
 [ 0  8  5]
 [ 6 28 16]]
muhdhisham@pop-os:~/SCL$
```