D**ATA SCIENCE & MACHINE LEARNING**

**LAB CYCLE 2**

1.Create a three dimensional array specifying float data type and print it.

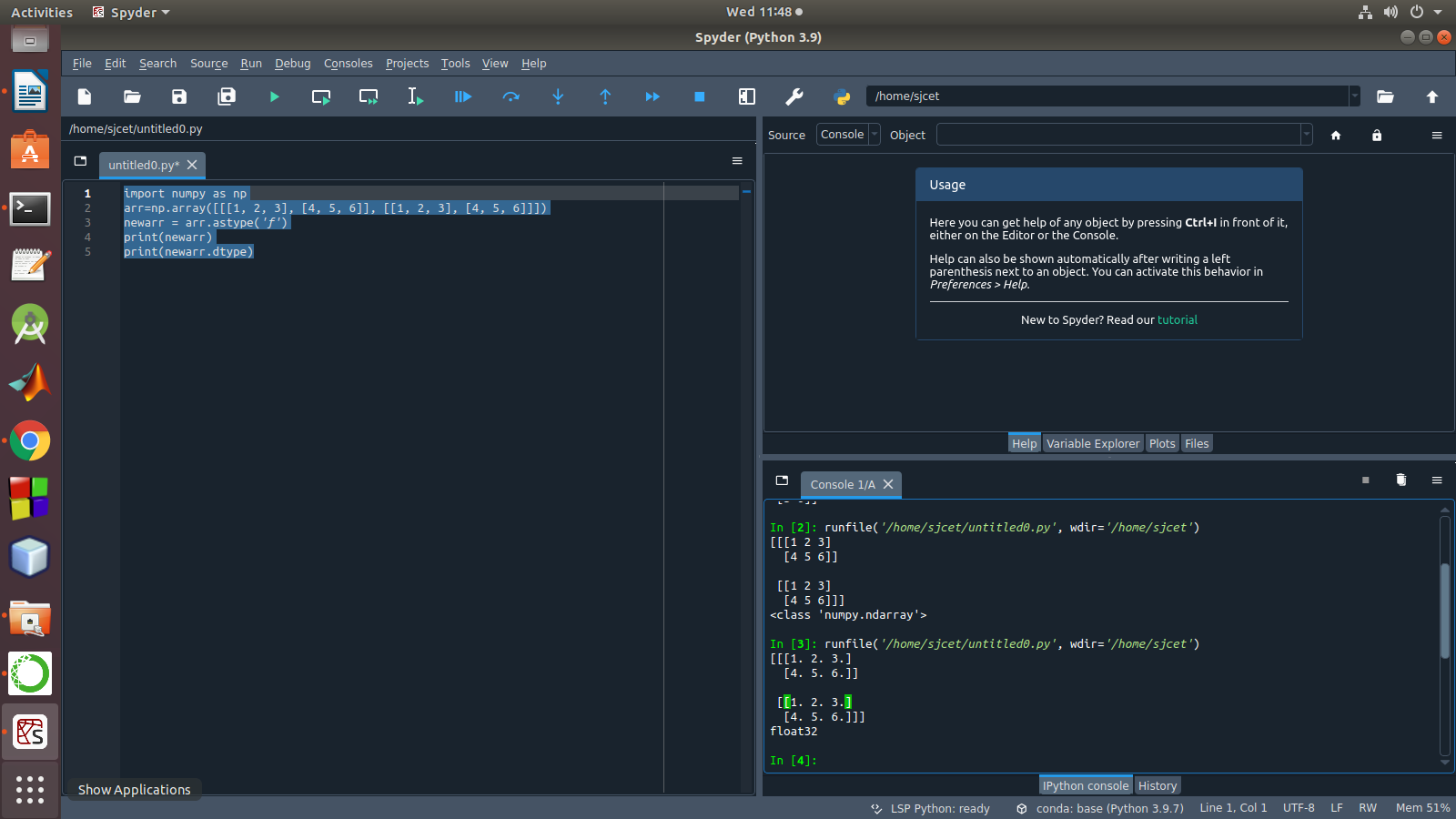
**Program**

import numpy as np

arr=np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]],dtype='f')

print(arr)

**output**



2. Create a 2 dimensional array (2X3) with elements belonging to complex data

type and print it. Also display

a. the no: of rows and columns

b. dimension of an array

c. reshape the same array to 3X2

**program**

import numpy as np

x = np.array([[2, 4, 6], [6.5, 8, 10]])

print(type(x))

print(x)

numOfRows = np. size(x, 0)

print(numOfRows)

numOfColumns = np. size(x, 1)

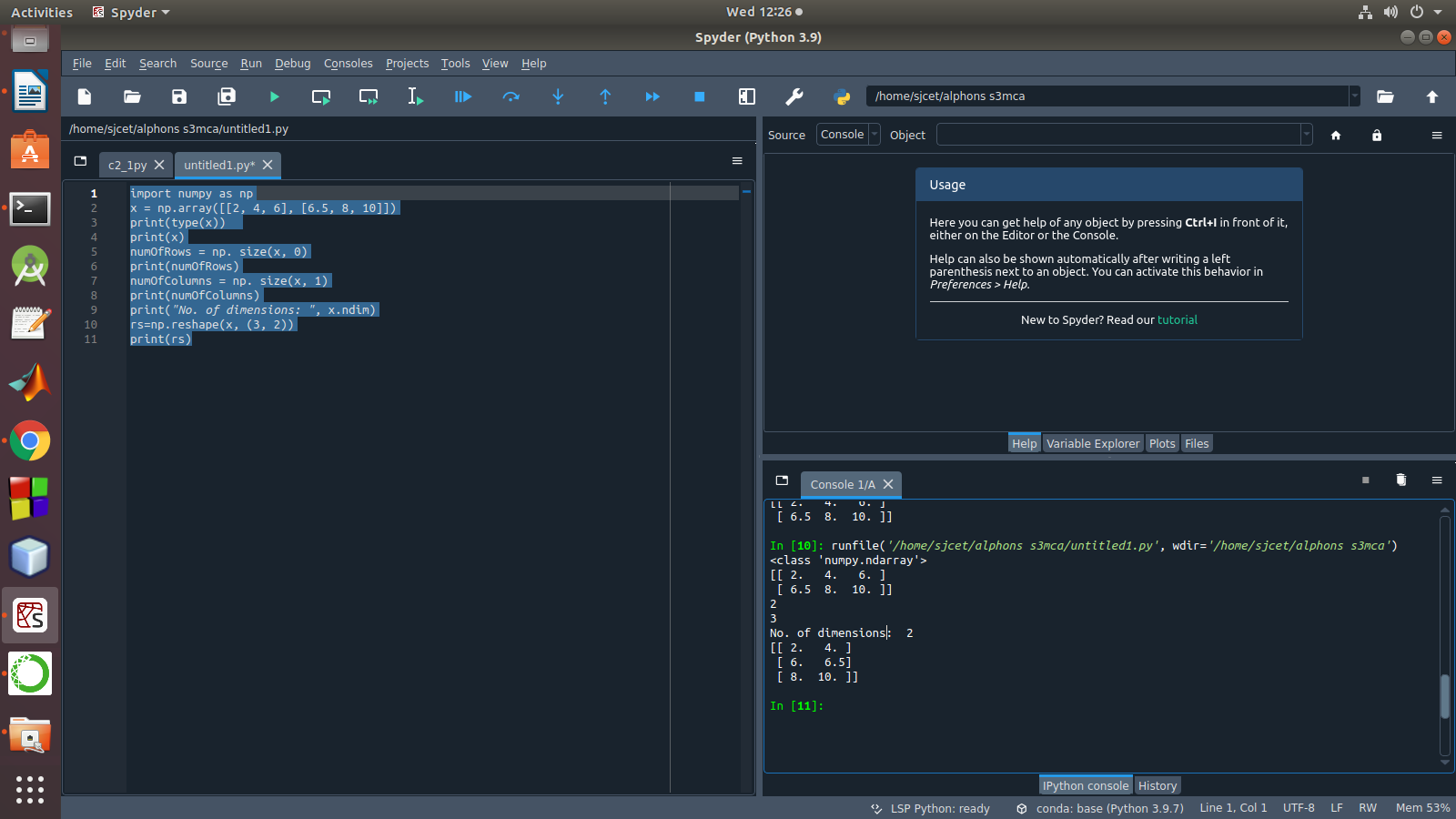
print(numOfColumns)

print("No. of dimensions: ", x.ndim)

rs=np.reshape(x, (3, 2))

print(rs)

**output**



3. Familiarize with the functions to create

a) an uninitialized array

b) array with all elements as 1,

c) all elements as 0

**program**

import numpy as np

x=np.empty([2, 2])

print(x)

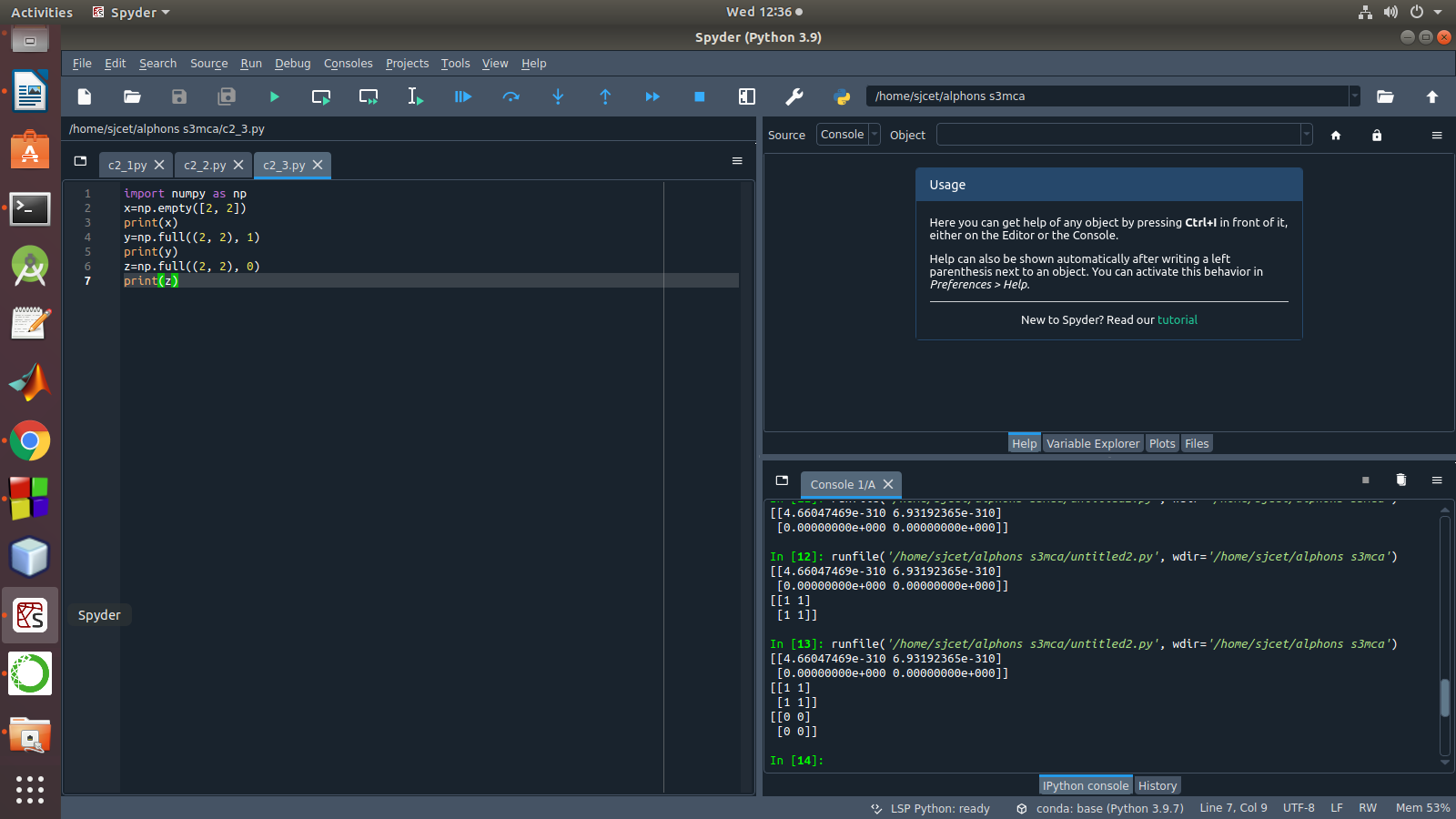
y=np.full((2, 2), 1)

print(y)

z=np.full((2, 2), 0)

print(z)

**output**



4. Create an one dimensional array using arange function containing 10 elements.

Display

a. First 4 elements

b. Last 6 elements

c. Elements from index 2 to 7

**Program**

import numpy as np

a = np.arange(1, 11, 1)

print(a)

first\_element = a[:4]

print(first\_element)

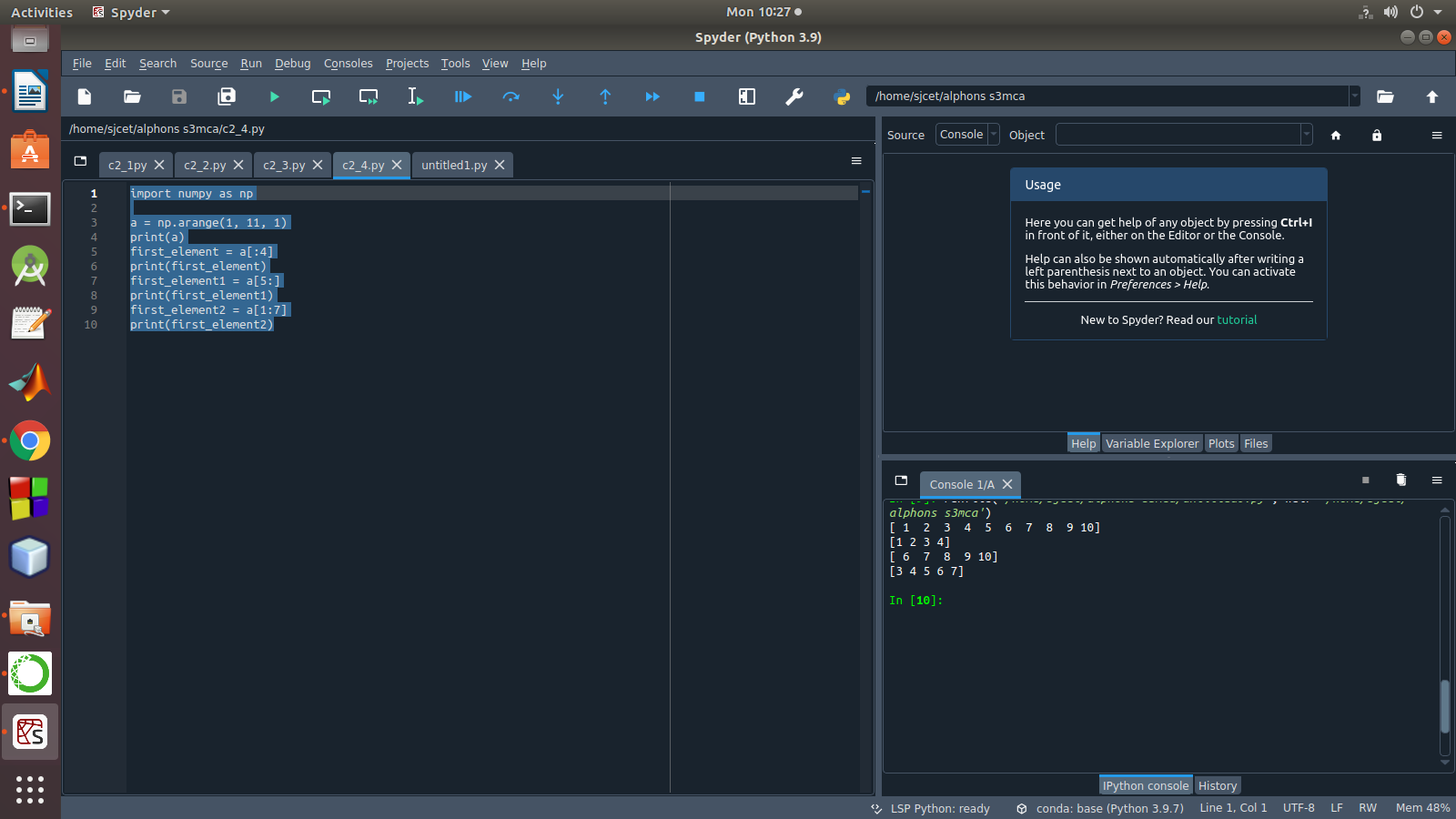
first\_element1 = a[5:]

print(first\_element1)

first\_element2 = a[1:7]

print(first\_element2)

**Output**



5. Create an 1D array with arange containing first 15 even numbers as elements

a. Elements from index 2 to 8 with step 2(also demonstrate the same

using slice function)

b. Last 3 elements of the array using negative index

c. Alternate elements of the array

d. Display the last 3 alternate elements

**Program**

mport numpy as np

a = np.arange(0, 15, 2)

print(a)

print("Elements from index 2 to 8 with step 2")

s2 = slice(2, 8, 2)

print(a[s2])

print("Last 3 elements of the array using negative index",a[-3:-1])

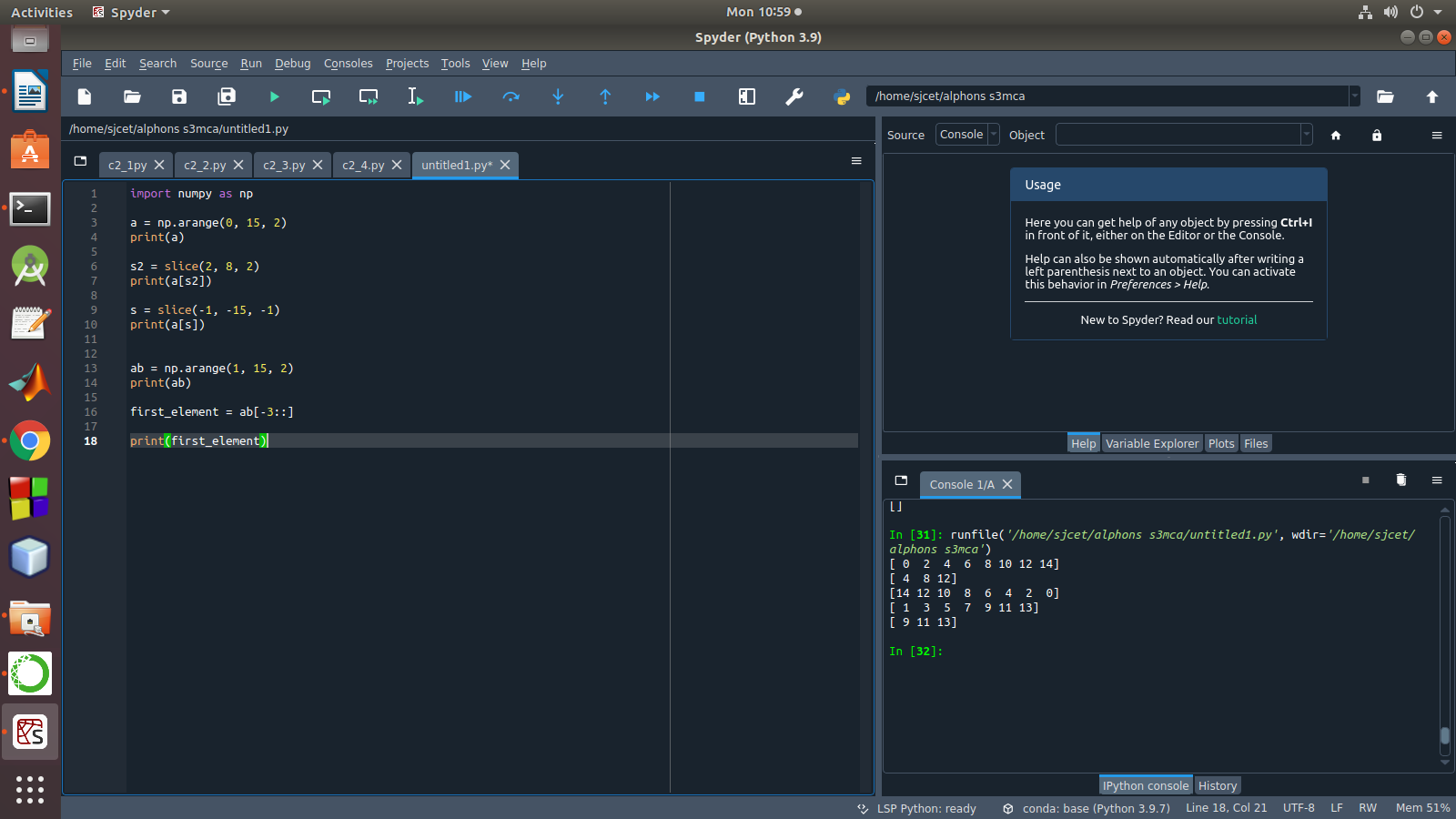
print("Alternate elements of the array")

ab = np.arange(1, 15, 2)

print(ab)

print("Display the last 3 alternate elements",a[-3:-1:2])

**Output**



6. Create a 2 Dimensional array with 4 rows and 4 columns.

a. Display all elements excluding the first row

b. Display all elements excluding the last column

c. Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row

d. Display the elements of 2 nd and 3 rd column

e. Display 2 nd and 3 rd element of 1 st row

f. Display the elements from indices 4 to 10 in descending order(use

–values)

**Program**

import numpy as np

x = np.array([[2, 4, 6,1], [6, 8, 10,1],[1, 2, 1,1], [1, 1, 1,1]])

print(x)

print("Display all elements excluding the first row")

print(x[1:])

print("Display all elements excluding the last column")

print(x[:, :3])

print("Display the elements of 2 nd and 3 rd column")

print(x[:, 1:3])

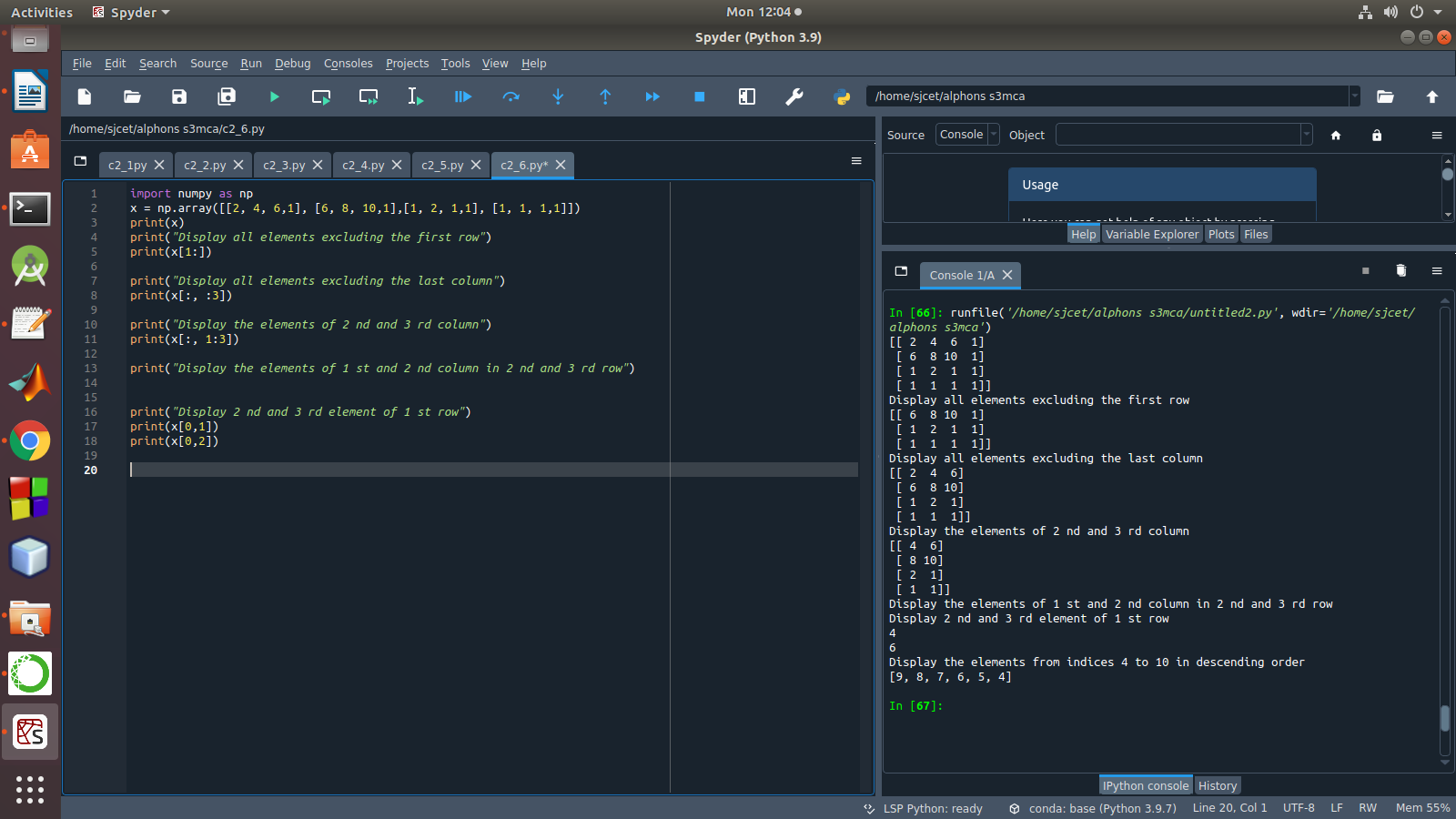
print("Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row")

print("Display 2 nd and 3 rd element of 1 st row")

print(x[0,1])

print(x[0,2])

**Output**



7. Create two 2D arrays using array object and

a. Add the 2 matrices and print it

b. Subtract 2 matrices

c. Multiply the individual elements of matrix

d. Divide the elements of the matrices

e. Perform matrix multiplication

f. Display transpose of the matrix

g. Sum of diagonal elements of a matrix

**Program**

import numpy as np

M1 = np.array([[3, 6], [14, 21]])

M2 = np.array([[9, 27], [11, 22]])

M3 = M1 + M2

print("Matrix addition")

print(M3)

M1 = np.array([[3, 6], [14, 21]])

M2 = np.array([[9, 27], [11, 22]])

M3 = M1 - M2

print("Matrix Substract")

print(M3)

M1 = np.array([[3, 6], [14, 21]])

M2 = np.array([[9, 27], [11, 22]])

M3 = M1 / M2

print("Divide the elements of the matrices")

print(M3)

M1 = np.array([[3, 6], [5, -10]])

M2 = np.array([[9, -18], [11, 22]])

M3 = M1 \* M2

print("Multiply the individual elements of matrix")

print(M3)

M1 = np.array([[3, 6], [5, -10]])

M2 = np.array([[9, -18], [11, 22]])

M3 = M1.dot(M2)

print("matrix multiplication")

print(M3)

M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])

M2 = M1.transpose()

print("Transpose of the matrix")

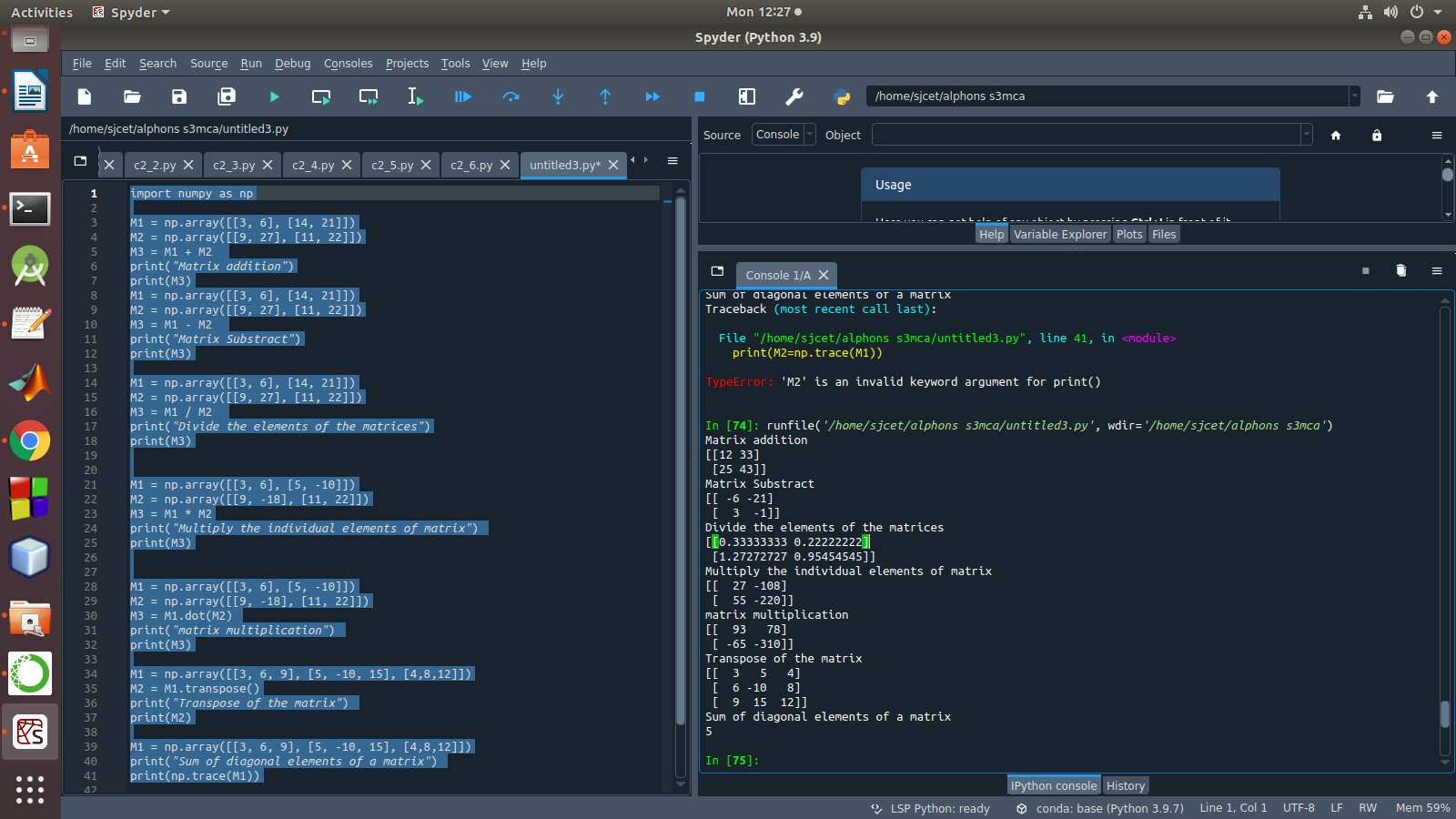
print(M2)

M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])

print("Sum of diagonal elements of a matrix")

print(np.trace(M1))

**Output**



8. Demonstrate the use of insert() function in 1D and 2D array

**Program**

import numpy as np

arr1 = np.arange(10, 16)

print("1D ARRAY ")

print("The array is: ", arr1)

obj = 2

value = 40

arr = np.insert(arr1, obj, value, axis=None)

print("After inserting the new array is: ")

print(arr)

print("Shape of the new array is : ", np.shape(arr))

print("2D ARRAY ")

arr1 = np.array([(1, 2, 3), (4, 5, 6), (7, 8, 9), (50, 51, 52)])

print("The array is: ")

print(arr1)

print("The shape of the array is: ", np.shape(arr1))

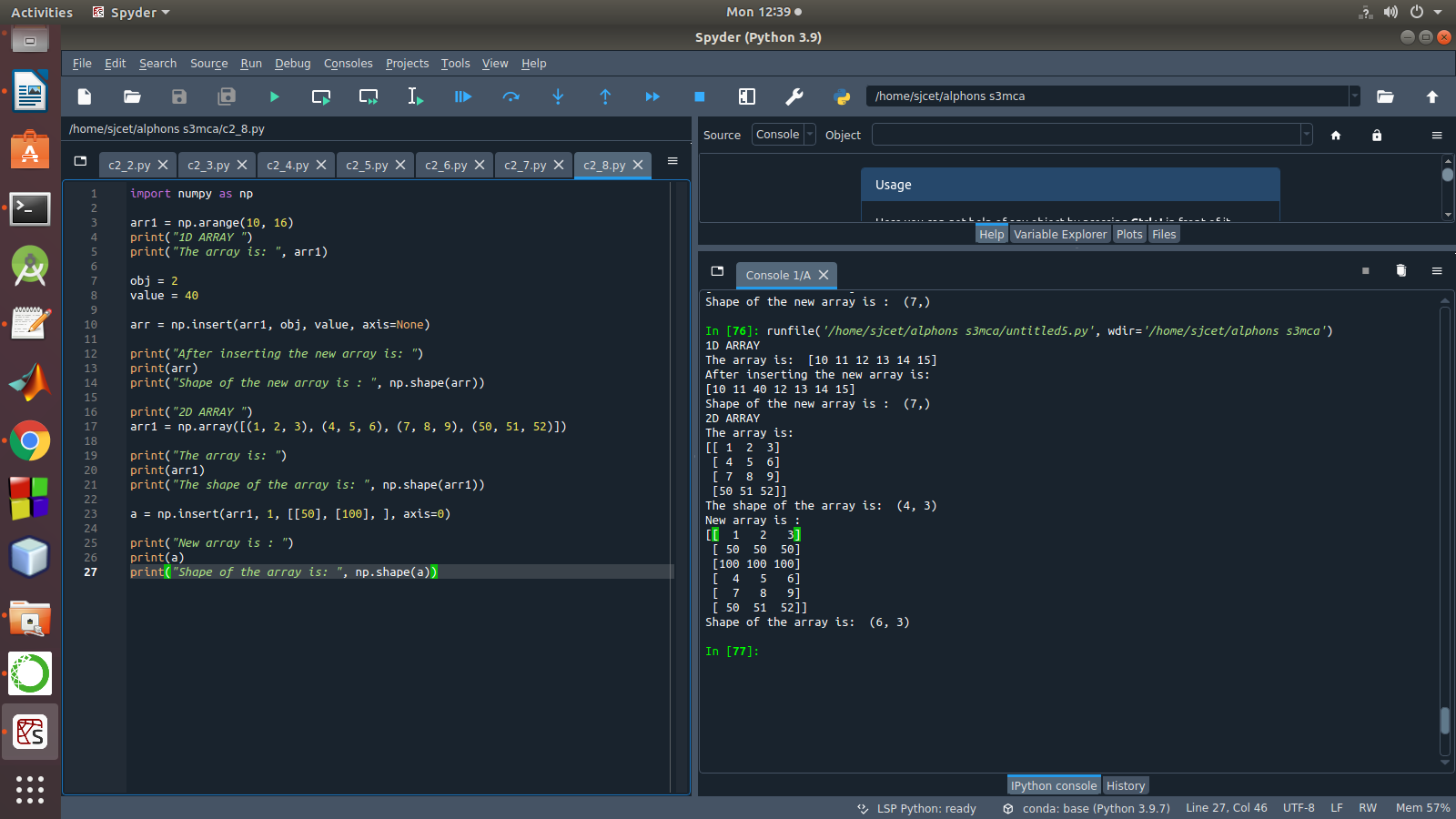
a = np.insert(arr1, 1, [[50], [100], ], axis=0)

print("New array is : ")

print(a)

print("Shape of the array is: ", np.shape(a))

**Output**



9. Demonstrate the use of diag() function in 1D and 2D array.

**Program**

import numpy as np

a= np.array([[3, 6,7,8]])

b=np.array([[3, 6,8,7], [4, 2,1,0],[3,1,3,3],[1,1,2,2]])

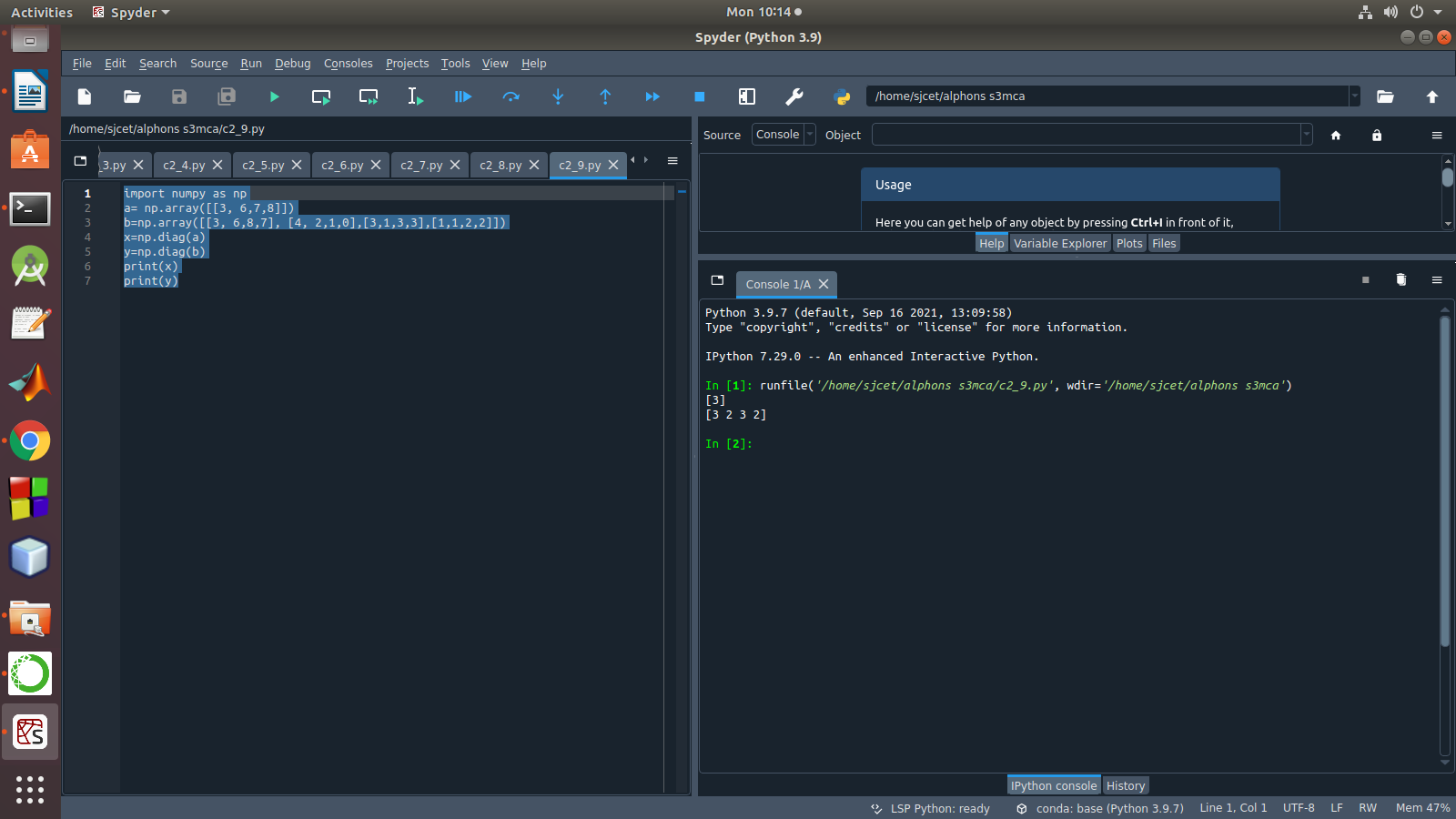
x=np.diag(a)

y=np.diag(b)

print(x)

print(y)

**Output**



10. Demonstarte the use of append() function in 1D and 2D

array.

**Program**

import numpy as np

a = np.array([[1,2,3],[4,5,6]])

b=np.array([1,2,3])

print("First array:")

print (a)

print("Second array")

print(b)

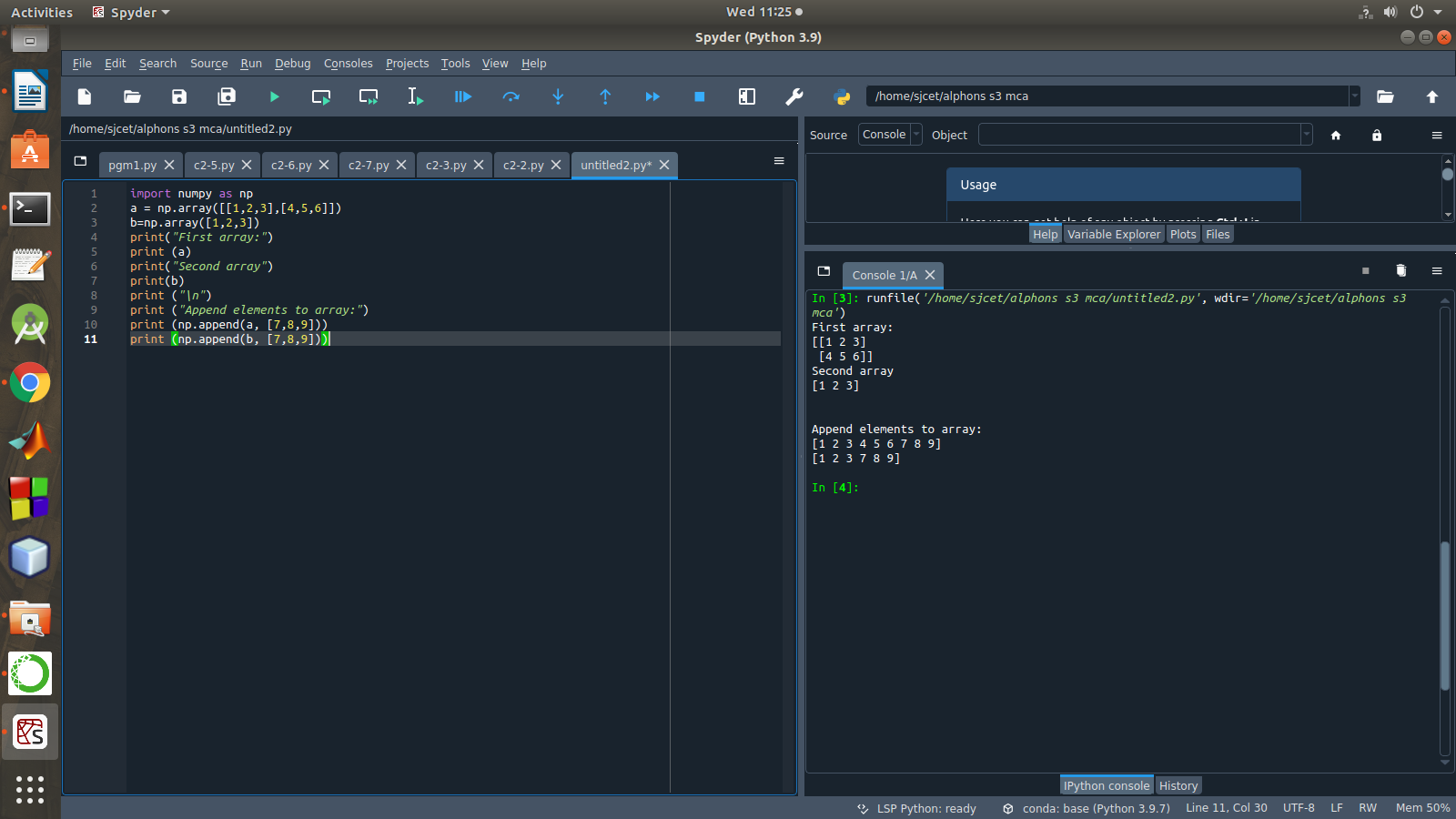
print ("\n")

print ("Append elements to array:")

print (np.append(a, [7,8,9]))

print (np.append(b, [7,8,9]))

**Output**



11. Demonstarte the use of sum() function in 1D and 2D array.

**Program**

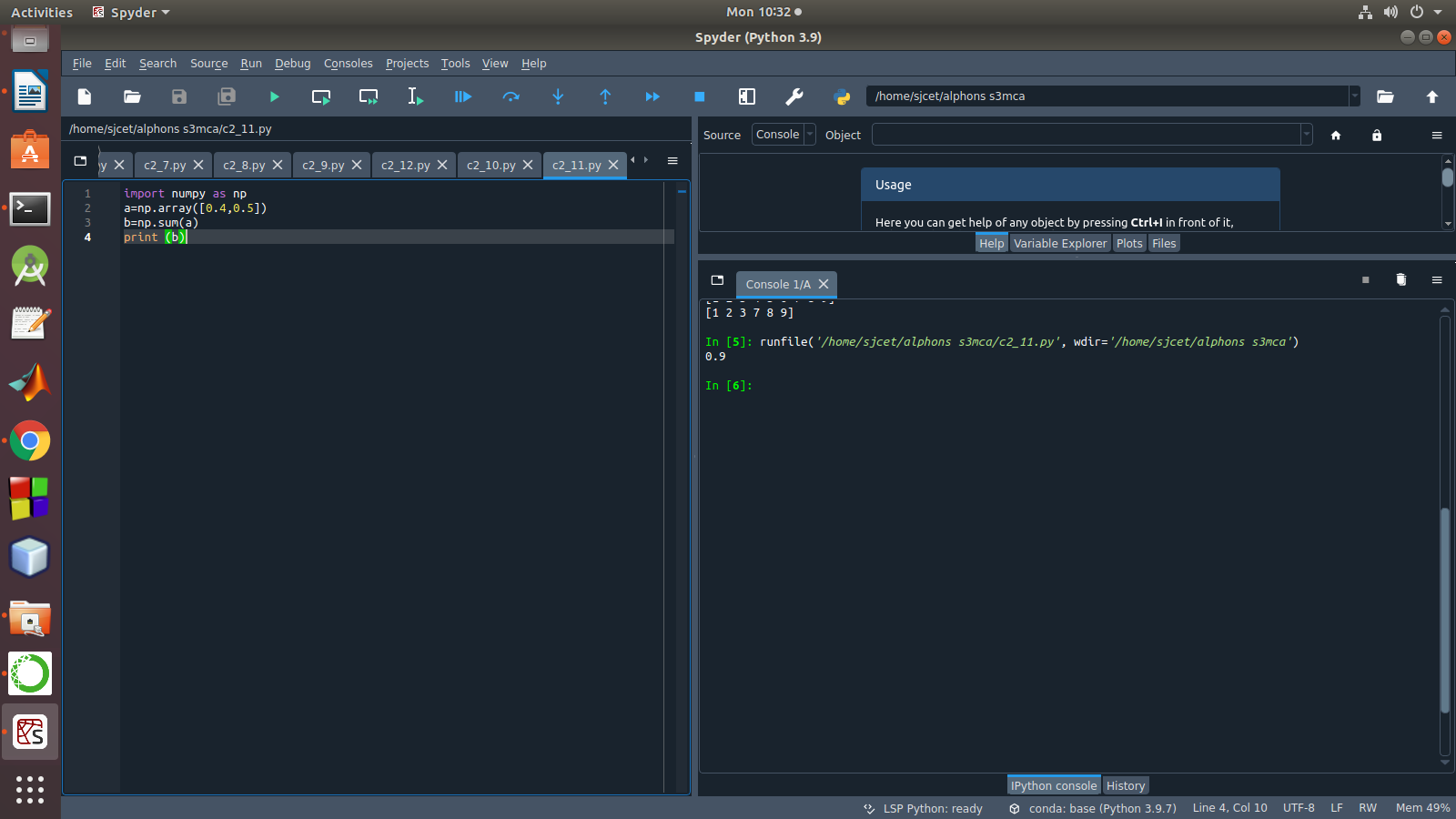
import numpy as np

a=np.array([0.4,0.5])

b=np.sum(a)

print (b)

**Output**



12.Create a 1 Dimensional array .Display the elements from indices 4 to 10 in descending order

**Program**

import numpy as np

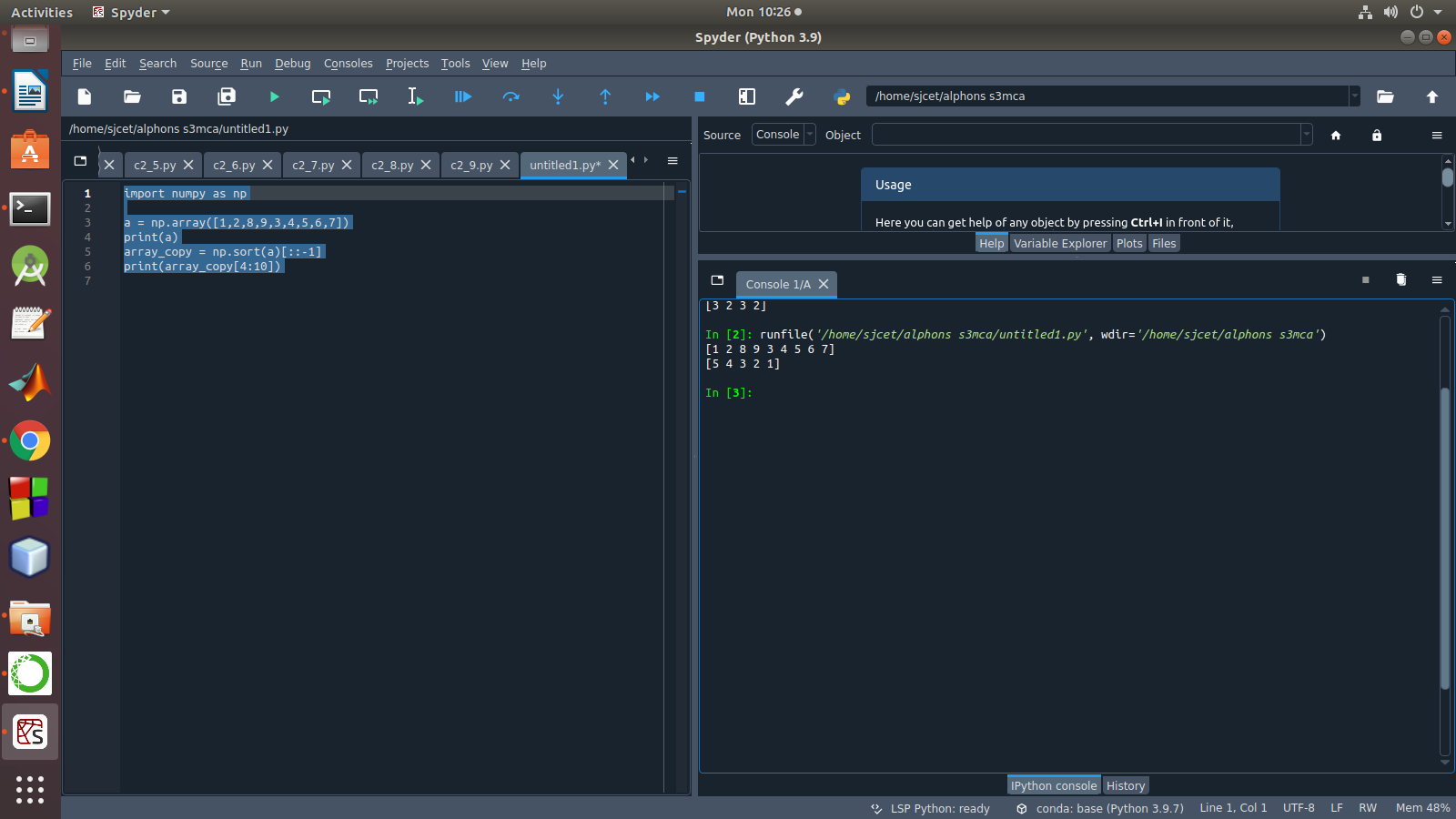
a = np.array([1,2,8,9,3,4,5,6,7])

print(a)

array\_copy = np.sort(a)[::-1]

print(array\_copy[4:10])

Output



**LAB CYCLE 2-PART 2**

1.Matrix decomposition, also kn Create a square matrix with random integer values(use randint()) and use appropriate functions to find:

i) inverse

ii) rank of matrix

iii) Determinant

iv) transform matrix into 1D array

v) eigen values and vectors

**Program**

import numpy as np

import numpy as nf

from numpy.linalg import eig

mat = np.random.randint(10, size=(3, 3))

array = nf.random.randint(10, size=(3, 3))

print(mat)

M\_inverse = np.linalg.inv(mat)

print("inverse of the array")

print(M\_inverse)

rank = np.linalg.matrix\_rank(mat)

print("Rank of the given Matrix ")

print(rank)

det= np.linalg.det(mat)

print("determinant of the given Matrix ")

print(det)

arr=mat.flatten()

print("transform matrix to array ")

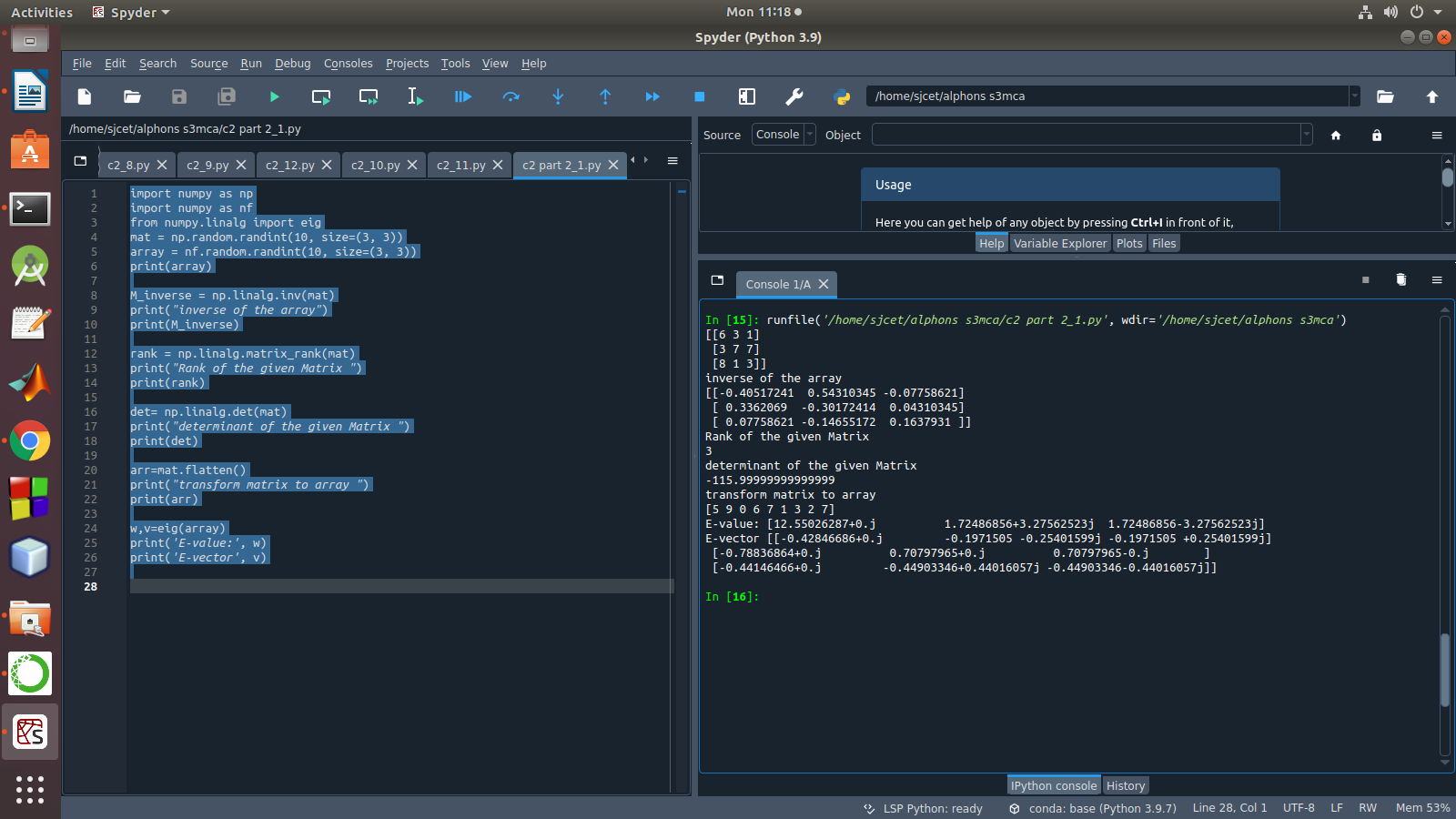
print(arr)

w,v=eig(array)

print('E-value:', w)

print('E-vector', v)

**Output**



2. Create a matrix X with suitable rows and columns

i) Display the cube of each element of the matrix using different methods(use multiply(), \*, power(),\*\*)

ii) Display identity matrix of the given square matrix.

iii) Display each element of the matrix to different powers.

iv) Create a matrix Y with same dimension as X and perform the operation X2+2Y

**Program**

import numpy as np

matrix=np.random.randint(0,10,4).reshape(2,2)

print("Display the cube of each element of the matrix using different methods (use multiply(),\*, power(),\*\*)")

x=np.power(matrix,3)

print("power()",x)

y=np.multiply(matrix,(matrix\*matrix))

print("multiply()")

print(y)

z=matrix\*matrix\*matrix

print("\*\*")

print(z)

cube=matrix\*3

print("\*")

print(cube)

print("Display identity matrix of the given square matrix.")

identity=np.identity(2,dtype=int)

print(identity)

print("Display each element of the matrix to different powers.")

dpow=np.power(matrix,matrix)

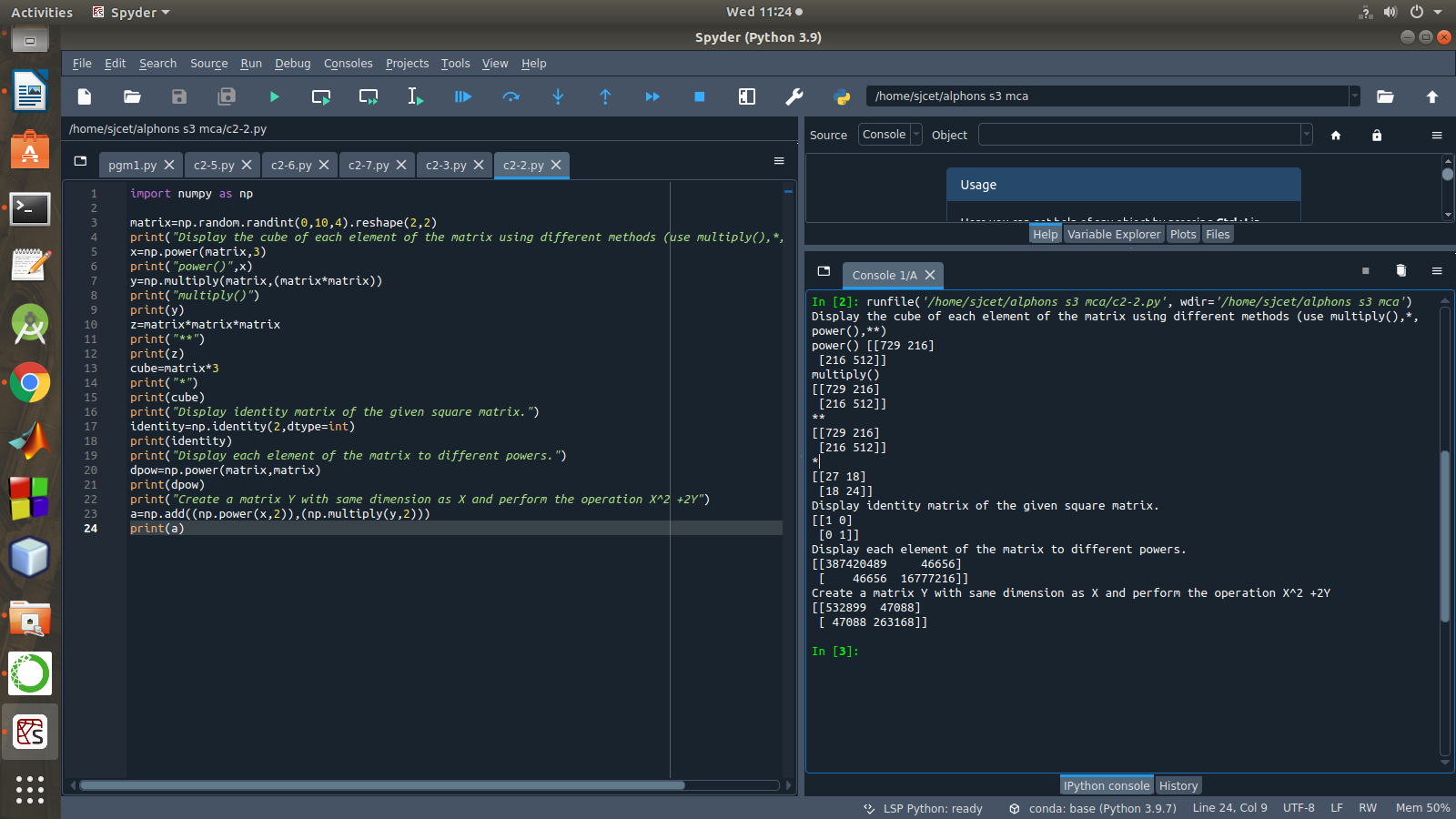
print(dpow)

print("Create a matrix Y with same dimension as X and perform the operation X^2 +2Y")

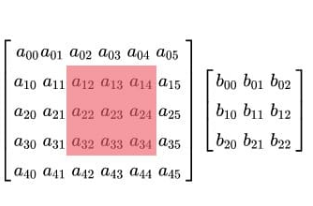
a=np.add((np.power(x,2)),(np.multiply(y,2)))

print(a)

**Output**



3. Multiply a matrix with a submatrix of another matrix and replace the same in larger matrix.



Program

import numpy as np

A = np.array([[6, 1, 1,6,3],

[4, -2, 5,1,3],

[2, 8, 7,7,8],

[6, 1, 1,6,3],

[2, 8, 7,7,8]])

B=np.array([[2, 1, -2],

[3, 0, 1],

[1, 1, -1]])

print("Mat A=\n",A)

print("Mat B=\n",B)

C=A[:3, :3]

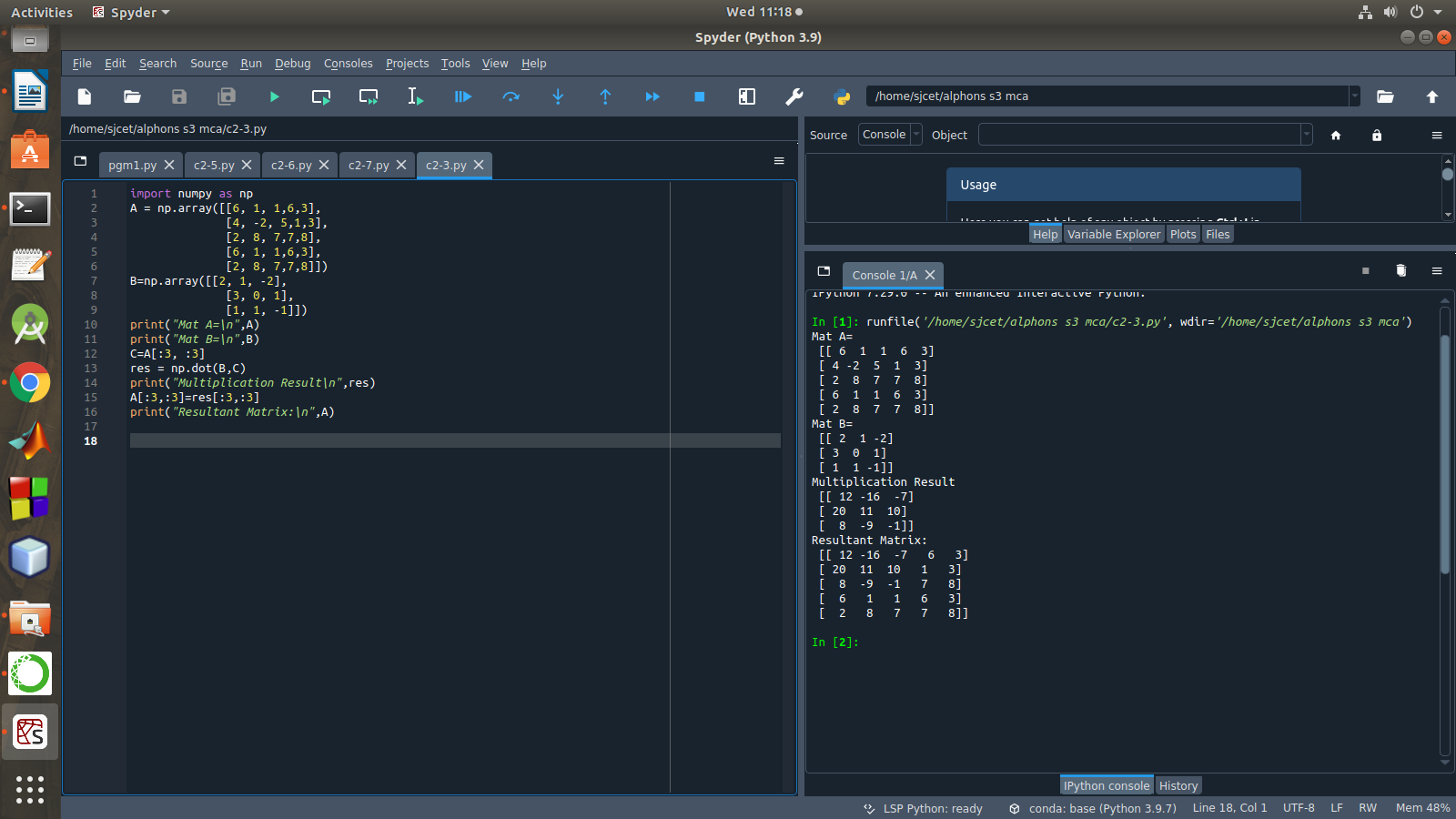
res = np.dot(B,C)

print("Multiplication Result\n",res)

A[:3,:3]=res[:3,:3]

print("Resultant Matrix:\n",A)

Output



4.Given 3 Matrices A, B and C. Write a program to perform matrix multiplication of the 3 matrices.

**Program**

import numpy as np

m1 = np.random.randint(20, size=(2, 2))

print(m1)

m2 = np.random.randint(20, size=(2, 2))

print(m2)

m3 = np.random.randint(20, size=(2, 2))

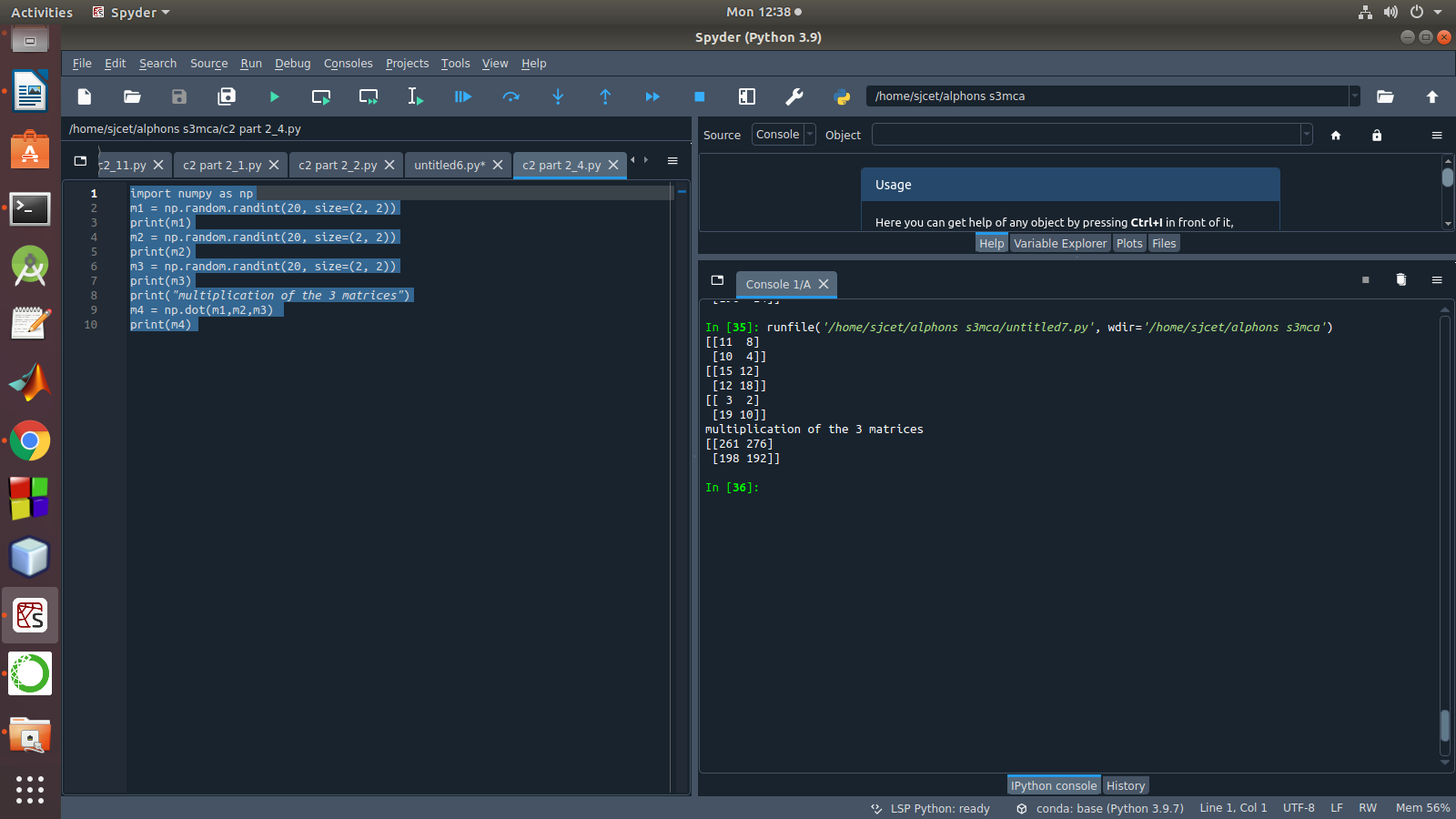
print(m3)

print("multiplication of the 3 matrices")

m4 = np.dot(m1,m2,m3)

print(m4)

**output**



5. Write a program to check whether given matrix is symmetric or Skew Symmetric.

**Program**

import numpy as np

A = np.array([[6, 1, 1],

[4, -2, 5],

[2, 8, 7]])

inv=np.transpose(A)

print (inv)

neg=np.negative(A)

comparison = A == inv

comparison1 = inv== neg

equal\_arrays = comparison.all()

skew=comparison1.all()

if equal\_arrays :

print("Symmetric")

else:

print("not Symmetric")

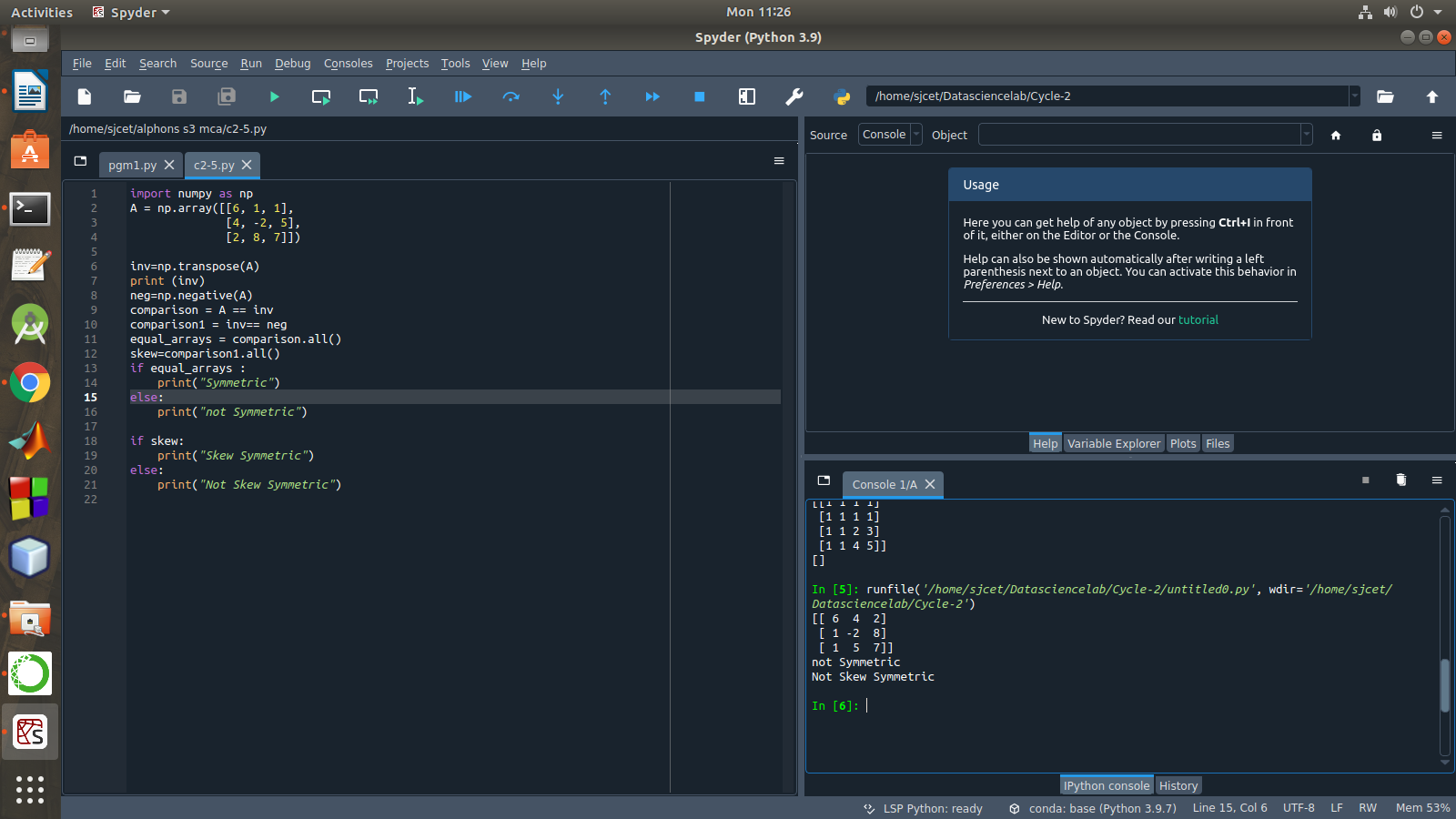
if skew:

print("Skew Symmetric")

else:

print("Not Skew Symmetric")

**output**

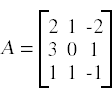


Solving systems of equations with numpy

One of the more common problems in linear algebra is solving a matrix-vector equation.

Here is an example. We seek the vector x that solves the equation

A X = b

Where  

And X=A-1 b.

Numpy provides a function called solve for solving such eauations.

6. Write a program to find out the value of X using **solve(),** given **A** and **b** as above own as matrix factorization, involves describing a given matrix using its constituent elements.

**Program**

import numpy as np

A = np.array([[2, 1, -2],

[3, 0, 1],

[1, 1, -1]])

b=np.array([[3],

[5],

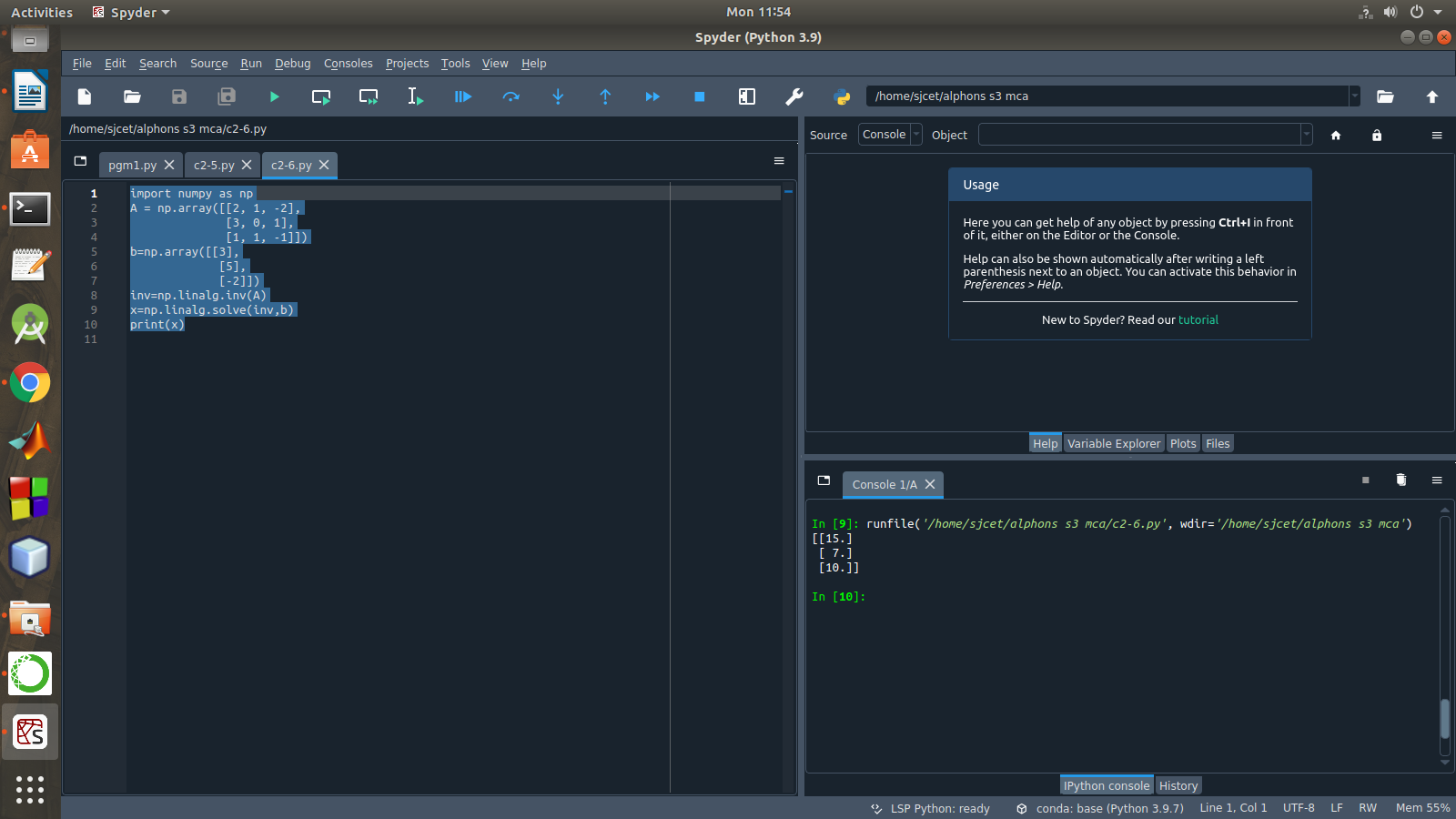
[-2]])

inv=np.linalg.inv(A)

x=np.linalg.solve(inv,b)

print(x)

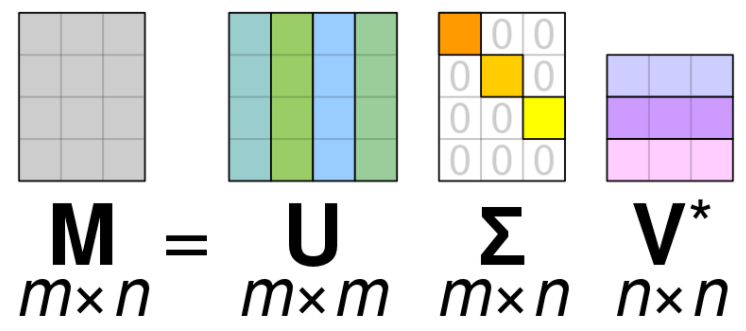
**Output**



Singular value Decomposition

The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler. This approach is commonly used in reducing the no: of attributes in the given data set.

**M= U ∑V^T**



* **M**-is original matrix we want to decompose
* **U**-is left singular matrix (columns are left singular vectors). **U** columns contain eigenvectors of matrix **MM**ᵗ
* **Σ**-is a diagonal matrix containing singular (eigen) values.
* **V**-is right singular matrix (columns are right singular vectors). **V** columns contain eigenvectors of matrix **M**ᵗ**M**

**Numpy** provides a function for performing svd, which decomposes the given matrix into 3 matrices.

7 . Write a program to perform the SVD of a given matrix. Also reconstruct the given matrix from the 3 matrices obtained after performing SVD.

**Program**

from numpy import array

from scipy.linalg import svd

from numpy import diag

from numpy import dot

from numpy import zeros

# define a matrix

A = array([[1, 2], [3, 4], [5, 6]])

print(A)

# SVD

U, s, VT = svd(A)

print("first" ,U)

print("second",s)

print("3rd" ,VT)

Sigma = zeros((A.shape[0], A.shape[1]))

# populate Sigma with n x n diagonal matrix

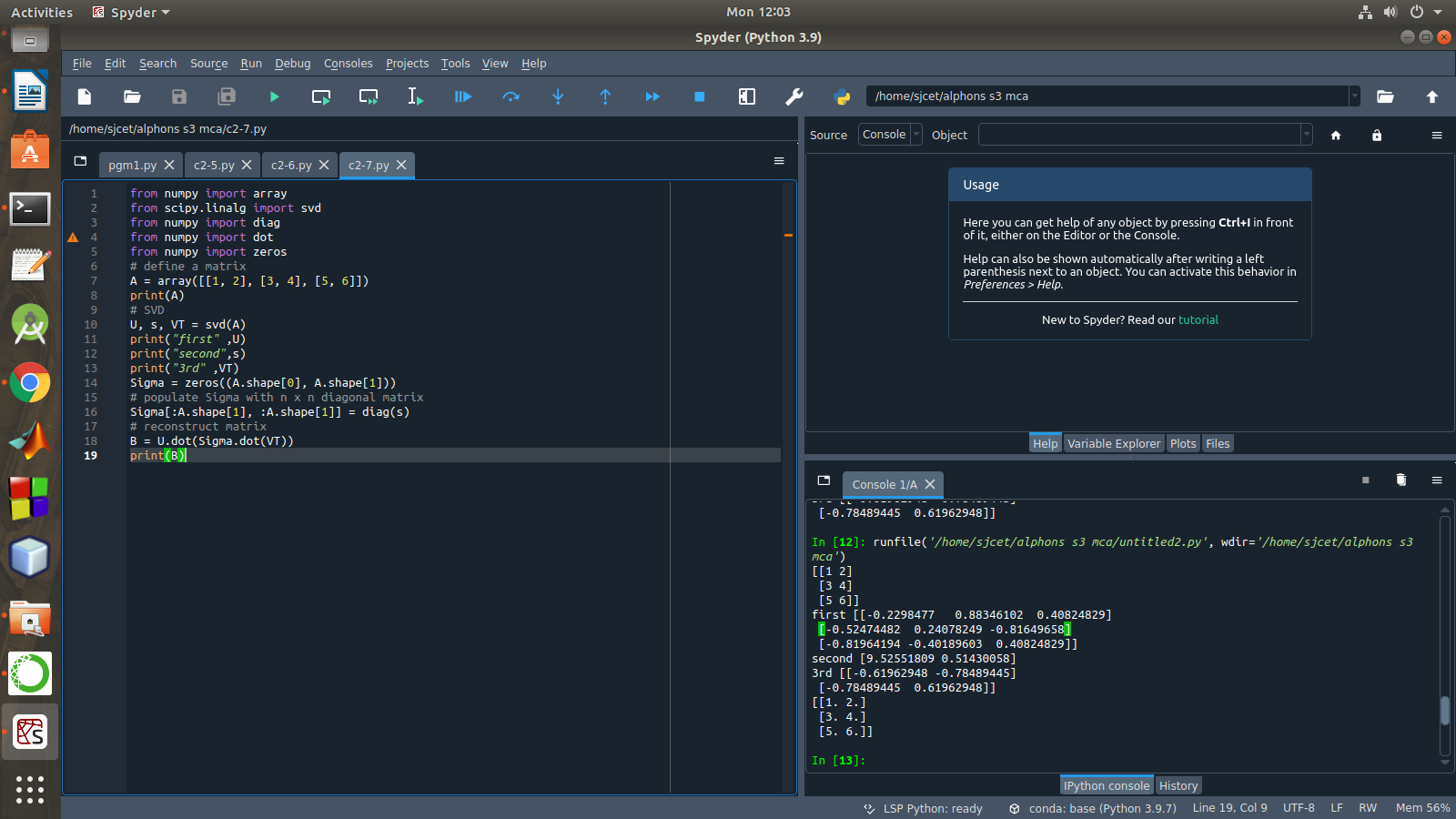
Sigma[:A.shape[1], :A.shape[1]] = diag(s)

# reconstruct matrix

B = U.dot(Sigma.dot(VT))

print(B)

**Output**

****