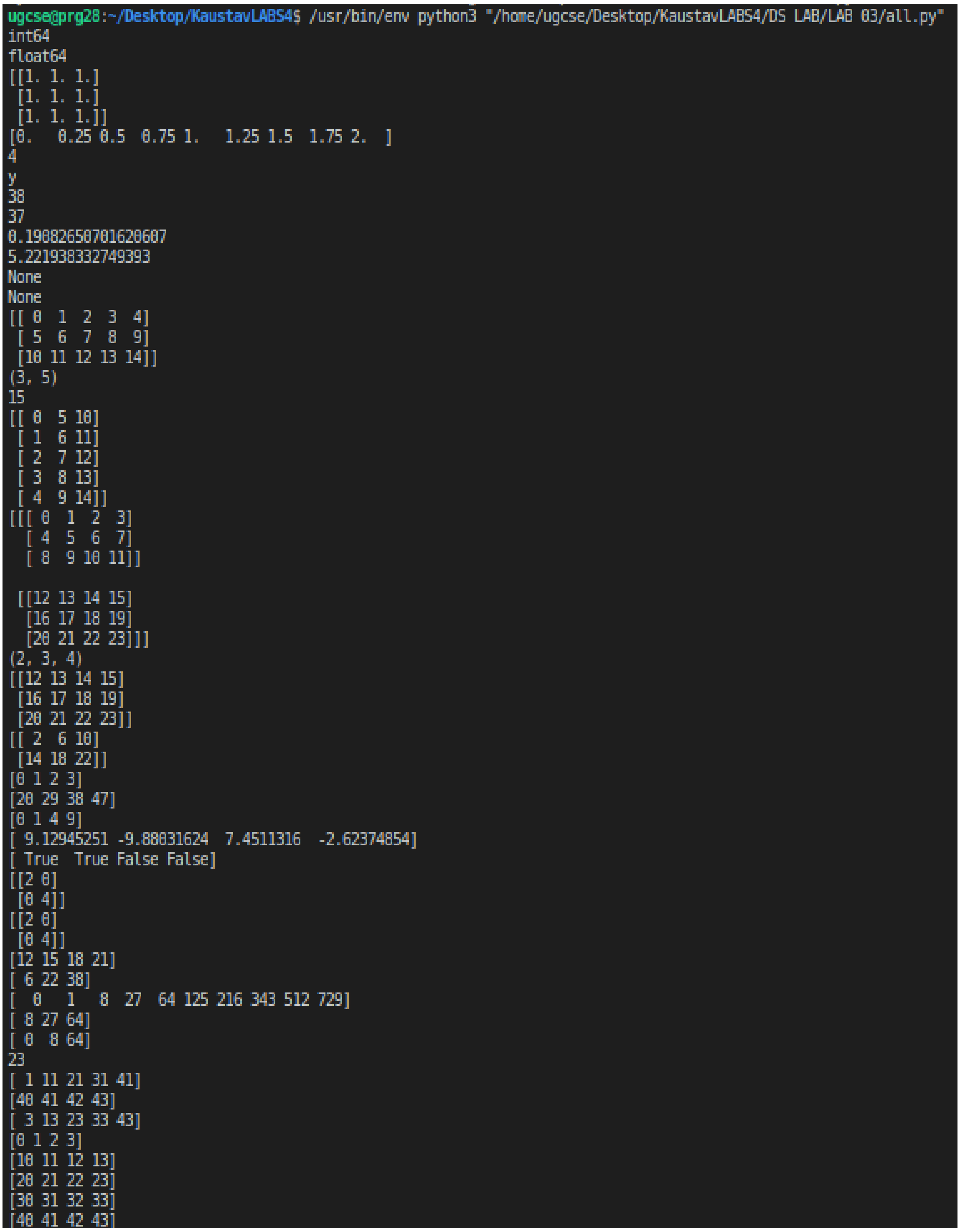
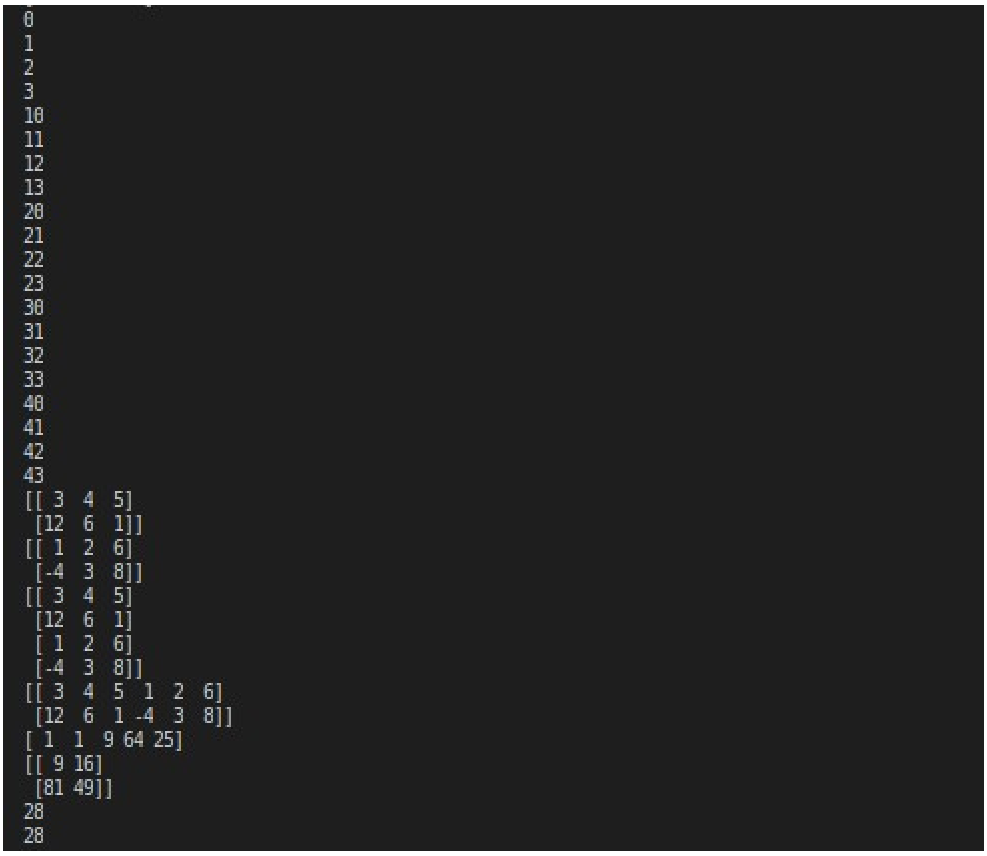
```
#!/usr/bin/env python3
import numpy as np
import random

# Usage of Numpy Data Structure
# 1. Array creation
A = np.array([2, 5, 10])
print(A.dtype)  # Will show, int64 data type
B = np.array([2.4, 10.6, 5.2])
print(B.dtype)  # Will show, float64 data type

# Creating sequence of sequence will create 2-dimensional array.
A = np.array([(3, 4, 5), (12, 6, 1)])
Z = np.zeros((2, 4))  # will create zero matrix of dimension 2x4

# Similarly,
print(np.ones((3, 3)))  # will create one’s matrix of dimension 3x3

# To create a sequence of data,
# Print(S) will give(10, 15, 20, 25, 30), with step size of 5
S = np.arange(10, 30, 5)
np.arange(0, 2, 0.3)  # it accepts float arguments

# Instead of step-size, we can specify total number of elements in the array
using
# produce 9 numbers starting 0 & ends with 2array([ 0. , 0.25, 0.5 , 0.75, 1. ,
1.25, 1.5 , 1.75, 2. ])
S1 = np.linspace(0, 2, 9)
print(S1)

# usage of Random Number functions
# this will pick one number from the list randomly
print(random.choice([1, 2, 3, 4, 5]))
# will pick one character from the string randomly
print(random.choice("python"))
print(random.randrange(25, 50))  # will pick one integer between 25 to 50

# will pick one integer between 25 to 50 with step size of 2
print(random.randrange(25, 50, 2))

print(random.random())  # will pick a random number between 0 to 1
# will pick a floating point number between 5 to 10
print(random.uniform(5, 10))
print(random.shuffle([1, 2, 3, 4, 5]))  # will shuffle the list elements
print(random.seed(10))  # to get same random value during every execution

# 2-Dimensional array(Matrix)
a = np.arange(15).reshape(3, 5)
print(a)

# to check the dimension
print(a.shape)
print(a.size)  # will return total elements in matrix (here 15)
# to transpose a matrix
print(a.T)  # transposed to 5x3 matrix


# 3-Dimensional array
# 1 st value indicates (no of planes) (3,4) is the dimensionprint(c)
c = np.arange(24).reshape(2, 3, 4)
print(c)
print(c.shape)  # will return (2, 3, 4)
print(c[1, ...])  # is equal to c[1, :, :]  # will fetch all elements of 2 nd
plane
```

```
print(c[..., 2])  # is equal to c[:, :, 2][[3, 7, 11], [15, 19, 23]]

# Array operations
a = np.array([20, 30, 40, 50])
b = np.arange(4)
print(b)

c = a-b
print(c)

print(b**2)
print(10*np.sin(a))
print(a < 35)

# Matrix operations
A = np.array([[1, 1], [0, 1]])
B = np.array([[2, 0], [3, 4]])
print(A*B)

# elementwise product

print(A * B)

# matrix product

np.dot(A, B)

# another matrix product
b = np.arange(12).reshape(3, 4)

print(b.sum(axis=0))
print(b.sum(axis=1))

# Indexing, Slicing & Iterating Array
a = np.arange(10)**3
print(a)
print(a[2:5])
print(a[0:6:2])

# Let ▫b▫, is an input matrix of size 5x4
b = np.array([[0, 1, 2, 3],
              [10, 11, 12, 13],
              [20, 21, 22, 23],
              [30, 31, 32, 33],
              [40, 41, 42, 43]])
print(b[2, 3])  # will fetch 23
print(b[0:5, 1])  # or b[:5, 1] or b[:, 1]  # will fetch [1,11,21,31,41]
print(b[-1, :])  # will fetch last row
print(b[:, -1])  # will fetch last col
for row in b:
    print(row)  # will print every rowfor element in b.flat:

for element in b.flat:
    print(element)  # will show all elements of b in 1-D array

# Changing the shape of a matrix
b.ravel()  # returns the array flattened to (1x 20)
# Later, we can convert 5x4 matrix into 4x 5 matrix using
B1 = b.reshape(4, 5)
# Stacking together different arrays
A1 = np.array([(3, 4, 5), (12, 6, 1)])
A2 = np.array([(1, 2, 6), (-4, 3, 8)])
D1 = np.vstack((A1, A2))
D2 = np.hstack((A1, A2))
```

```
print(A1)
print(A2)
print(D1)
print(D2)

# Stacking 1-D array into 2-D array(column wise)
a = np.array([4., 2.])
b = np.array([3., 8.])
np.column_stack((a, b))

# returns a 2D array
np.hstack((a, b))
# the result is different
# np.hstack((a[:, newaxis], b[:, newaxis]))  # the result is the same


# Indexing with array of indices
a = np.arange(12)**2
i = np.array([1, 1, 3, 8, 5])
print(a[i])
# the first 12 square numbers
# an array of indices
# the elements of a at the positions i
j = np.array([[3, 4], [9, 7]])  # a bidimensional array of indices
print(a[j])  # the same shape as j

# Usage of for-loop(Mapping by Value)
# Calculate sum of all the elements in a 2D Numpy Array(iterate over elements)
a = np.array([(3, 2, 9), (1, 6, 7)])
s1 = 0

for row in a:
    for col in row:
        s1 += col
print(s1)

# Usage of for-loop(Mapping by Index)
# Calculate sum of all the elements in a 2D Numpy Array(iterate over range)
a = np.array([(3, 2, 9), (1, 6, 7)])
s = 0
for i in range(a.shape[0]):
    for j in range(a.shape[1]):
        s += a[i, j]
print(s)
```



```
ugcse@prg28:~/Desktop/KaustavLABS4$ /usr/bin/env python3 "/home/ugcse/Desktop/KaustavLABS4/DS LAB/LAB 03/all.py"
int64
float64
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[0.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]
4
y
38
37
0.19082650701620607
5.221938332749393
None
None
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
(3, 5)
15
[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
(2, 3, 4)
[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
[[ 2  6 10]
 [14 18 22]]
[0 1 2 3]
[20 29 38 47]
[0 1 4 9]
[ 9.12945251 -9.88031624  7.4511316  -2.62374854]
[ True  True False False]
[[2 0]
 [0 4]]
[[2 0]
 [0 4]]
[12 15 18 21]
[ 6 22 38]
[  0   1   8  27  64 125 216 343 512 729]
[ 8 27 64]
[ 0  8 64]
23
[ 1 11 21 31 41]
[40 41 42 43]
[ 3 13 23 33 43]
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```



```
0
1
2
3
10
11
12
13
20
21
22
23
30
31
32
33
40
41
42
43
[[ 3  4  5]
 [12  6  1]]
[[ 1  2  6]
 [-4  3  8]]
[[ 3  4  5]
 [12  6  1]
 [ 1  2  6]
 [-4  3  8]]
[[ 3  4  5  1  2  6]
 [12  6  1 -4  3  8]]
[ 1  1  9 64 25]
[[ 9 16]
 [81 49]]
28
28
```