Tool & Technique Laboratory (T&T Lab.) [CS-3096]

Individual Work

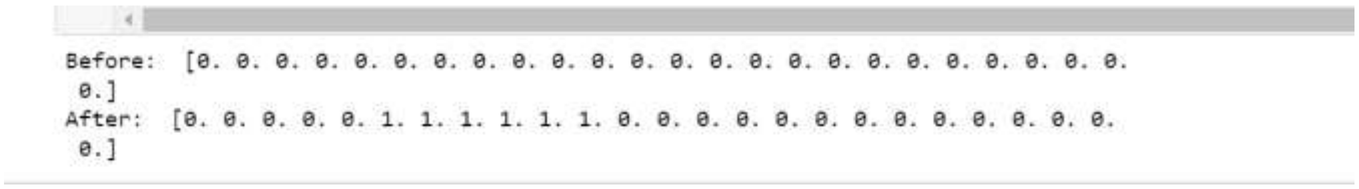
Lab. No:5 , Date: 14/02/2023 , Day: 5

Topic: Python basics

Roll Number:	**20051939**	**Branch/Section:**	**CSE-17**
Name in Capital:	**Shashikant shah**		

Program No: (5.1)

Program Title:
Write a program to create a null vector of size 25 but the values from fifth to tenth elements are all 1.
Input/Output Screenshots:
RUN-1:



```
Before:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]
After:  [0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]
```

Source code

```
import numpy as np
x = np.zeros(25)
print("Before operation: ",x)
for i in range(5,11):
    x[i] = 1
print("After operation: ",x)
```

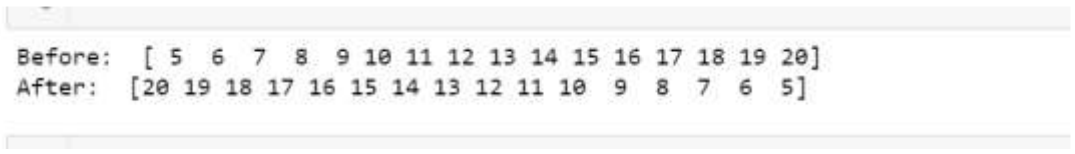
Conclusion/Observation

We have successfully created a null vector of size 25.

Program No: (5.2)

Program Title:
Write a program to create a vector with values ranging from 5 to 20 & reverse it.

Input/Output Screenshots:
RUN-1:



```
Before:  [ 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
After:  [20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5]
```

Source code

```
import numpy as np
v = np.arange(5,20)
print("Before Reversing: ",v)
res = np.flip(v)
print("AFter reversing: ",res)
```

Conclusion/Observation

We have successfully created a array of values ranging from 5 to 20 and reversed it.

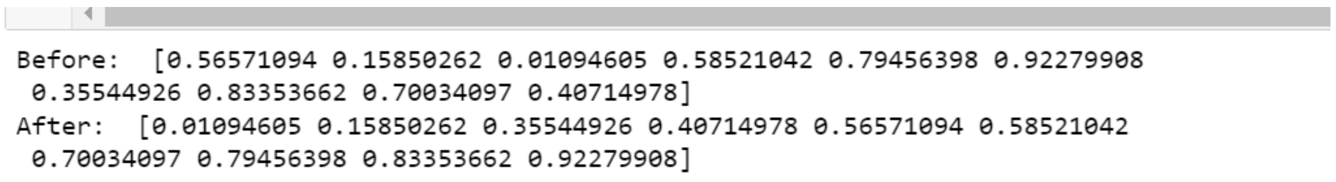
Program No: **(**5.3)

Program Title:

Write a program to create a vector with 10 random values & sort first half ascending & second half descending.

Input/Output Screenshots:

RUN-1:



```
Before:  [0.56571094 0.15850262 0.01094605 0.58521042 0.79456398 0.92279908
 0.35544926 0.83353662 0.70034097 0.40714978]
After:  [0.01094605 0.15850262 0.35544926 0.40714978 0.56571094 0.58521042
 0.70034097 0.79456398 0.83353662 0.92279908]
```

Source code

```
import numpy as np
arr=np.random.rand(10)
print("Before sorting: ",arr)
arr[:10].sort()
arr[10:].sort()
arr[10:]=arr[10:][::-1]
print("After sorting: ",arr)
```

Conclusion/Observation

We have successfully sorted in the required order.

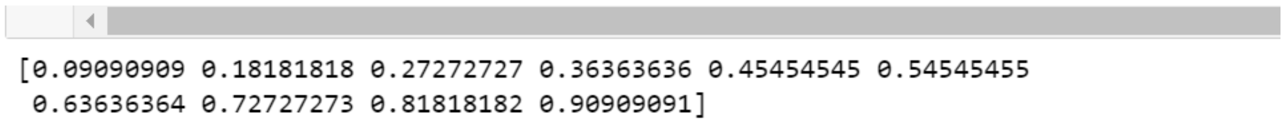
Program No: (5.4)

Program Title:

Write a program to create a vector of size 10 with values ranging from 0 to 1, both excluded.

Input/Output Screenshots:

RUN-1:



```
[0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
 0.63636364 0.72727273 0.81818182 0.90909091]
```

Source code

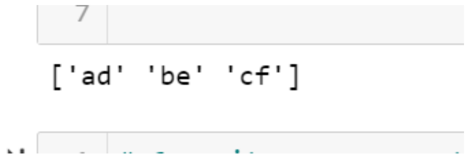
```
import numpy as np
x = np.linspace(0,1,12,endpoint=True)[1:-1]
print(x)
```

Conclusion/Observation

We have successfully created a vector having random values ranging from 0 to 1.

Program No: (5.5)

Program Title:
Write a program to concatenate element-wise two arrays of string.
Input/Output Screenshots:
RUN-1:



```
['ad' 'be' 'cf']
```

Source code

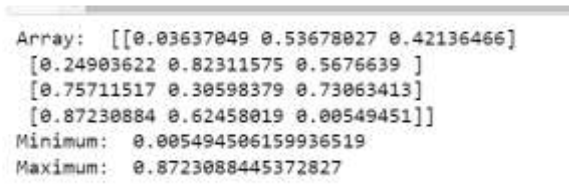
```
import numpy as np
x1 = np.array(['Hello', 'are'], dtype=np.str)
x2 = np.array([' how', ' you'], dtype=np.str)
print("Array1:")
print(x1)
print("Array2:")
print(x2)
new_array = np.char.add(x1, x2)
print("new array:")
print(new_array)
```

Conclusion/Observation

Succefully concatenated two arrays element wise.

Program No: (5.6)

Program Title:
Write a program to create a 4x3 array with random values & find out the minimum & maximum values.
Input/Output Screenshots:
RUN-1:



```
Array:  [[0.03637049 0.53678027 0.42136466]
 [0.24903622 0.82311575 0.5676639 ]
 [0.75711517 0.30598379 0.73063413]
 [0.87230884 0.62458019 0.00549451]]
Minimum:  0.005494506159936519
Maximum:  0.8723088445372827
```

Source code

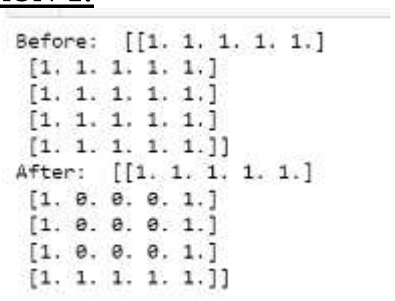
```
import numpy as np
x = np.random.random((4,3))
print("Original Array:")
print(x)
xmin, xmax = x.min(), x.max()
print("Minimum and Maximum Values:")
print(xmin, xmax)
```

Conclusion/Observation

We have successfully calculated min and max value in the 4*3 matrix.

Program No: (5.7)

Program Title:
Write a program to create a 2D array with 1 on the border and 0 inside.
Input/Output Screenshots:
RUN-1:



```
Before:  [[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
After:  [[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

Source code

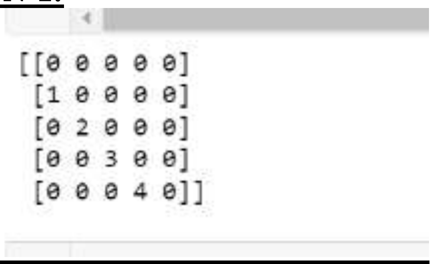
```
import numpy as np
x = np.ones((5,5))
print("Original array:")
print(x)
print("1 on the border and 0 inside in the array")
x[1:-1,1:-1] = 0
print(x)
```

Conclusion/Observation

We have successfully created a 2d array with 1 on the border and 0 inside.

Program No: (5.8)

Program Title:
Write a program to create a 4x4 matrix with values 1,2,3,4 just below & above the diagonal, rest zeros.
Input/Output Screenshots:
RUN-1:



```
[[0 0 0 0 0]
 [1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]]
```

Source code

```
import numpy as np
x = np.zeros((4, 4))
x[::2, 1::2] = 1
x[1::2, ::2] = 2
```

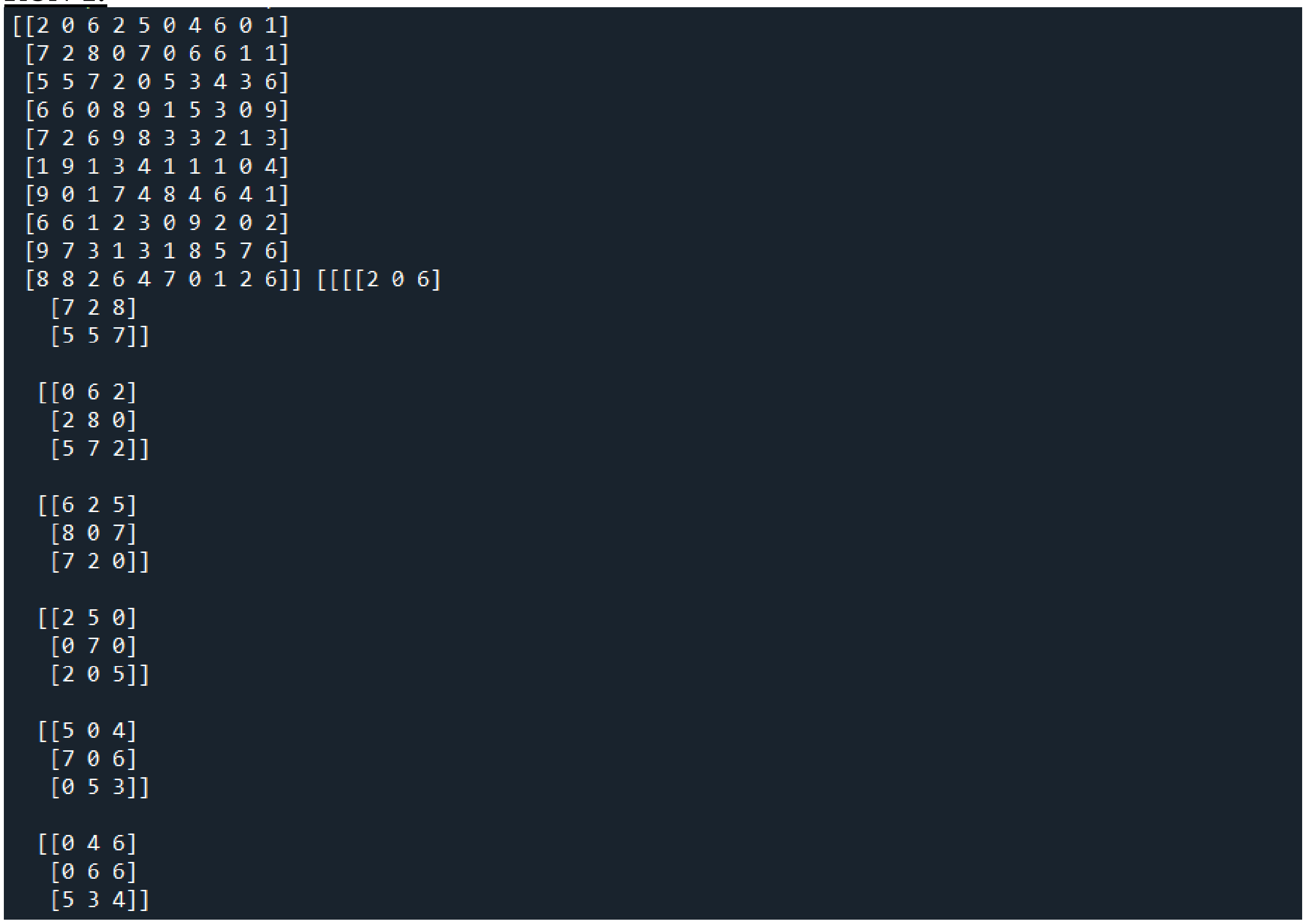
```
x[2::3, ::3] = 3
x[3::4, ::4] = 4
print(x)
```

Conclusion/Observation

We have successfully filled the matrix with the required values.

Program No: (5.9)

Program Title:
Write a program to extract all the contiguous 3x3 blocks from a random 10x10 matrix.
Input/Output Screenshots:
RUN-1:



```
[[2 0 6 2 5 0 4 6 0 1]
 [7 2 8 0 7 0 6 6 1 1]
 [5 5 7 2 0 5 3 4 3 6]
 [6 6 0 8 9 1 5 3 0 9]
 [7 2 6 9 8 3 3 2 1 3]
 [1 9 1 3 4 1 1 1 0 4]
 [9 0 1 7 4 8 4 6 4 1]
 [6 6 1 2 3 0 9 2 0 2]
 [9 7 3 1 3 1 8 5 7 6]
 [8 8 2 6 4 7 0 1 2 6]] [[[[2 0 6]
   [7 2 8]
   [5 5 7]]

  [[0 6 2]
   [2 8 0]
   [5 7 2]]

  [[6 2 5]
   [8 0 7]
   [7 2 0]]

  [[2 5 0]
   [0 7 0]
   [2 0 5]]

  [[5 0 4]
   [7 0 6]
   [0 5 3]]

  [[0 4 6]
   [0 6 6]
   [5 3 4]]
```

Source code

```
import numpy as np
arr = np.random.randint(10, size=(10, 10))
blocks = np.lib.stride_tricks.as_strided(arr, shape=(8, 8, 3, 3), strides=arr.strides*2)
print(arr,blocks)
```

Program No: (5.10)

Program Title:

A magic square is a matrix all of whose row sums, column sums and the sums of the twodiagonalsare the same. (One diagonal of a matrix goes from the top left to the bottom right, the other diagonalgoesfrom top right to bottom left.) Show by direct computation that if the matrix Ais given byA=np.array([[17, 24, 1, 8, 15],

[23, 5, 7, 14, 16],

[4, 6, 13, 20, 22],

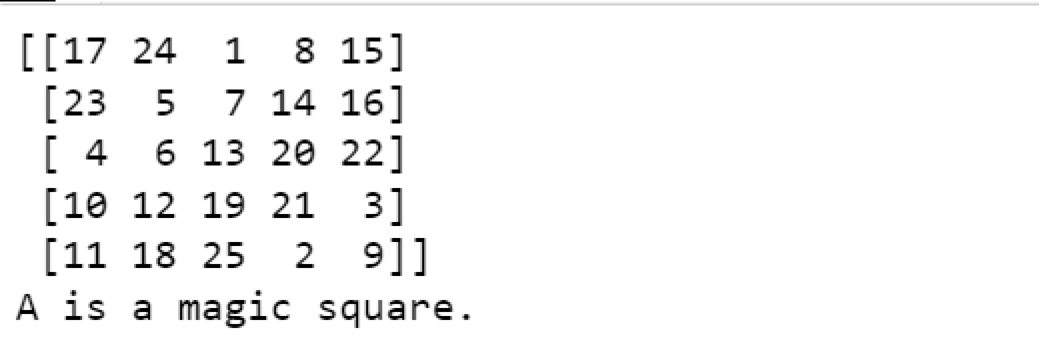
[10, 12, 19, 21, 3],

[11, 18, 25, 2, 9]])

The matrix A has 5 row sums (one for each row), 5 column sums (one for each column) andtwodiagonal sums. These 12 sums should all be exactly the same, and you could verify that theyarethesame by printing them and "seeing" that they are the same. It is easy to miss small differencesamongsomany numbers, though. Instead, verify that A is a magic square by constructing the 5 columnsumsandcomputing the maximum and minimum values of the column sums. Do the same for the5rowsums,and compute the two diagonal sums. Check that these six values are the same. If the maximumandminimum values are the same, the flyswatter principle says that all values are the same.

Input/Output Screenshots:

RUN-1:



```
[[17 24  1  8 15]
 [23  5  7 14 16]
 [ 4  6 13 20 22]
 [10 12 19 21  3]
 [11 18 25  2  9]]
A is a magic square.
```

Source code

```
import numpy as np
A = np.array([[17, 24, 1, 8, 15],
              [23, 5, 7, 14, 16],
              [4, 6, 13, 20, 22],
              [10, 12, 19, 21, 3],
              [11, 18, 25, 2, 9]])
print(A)
row_sums = A.sum(axis=1)
col_sums = A.sum(axis=0)
diag1_sum = np.diag(A).sum()
diag2_sum = np.diag(np.fliplr(A)).sum()
sums = np.concatenate((row_sums, col_sums, [diag1_sum], [diag2_sum]))
if np.ptp(sums) == 0:
    print("A is a magic square.")
```

```
else:
    print("A is not a magic square.")
```