ASSIGNMENT/ TASK 4

1. Import the numpy package under the name np and Print the numpy version and the configuration

```
import numpy as np
print(np.__version__)
print(np.show_config())
```

```
1.19.5
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
lapack_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/usr/local/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
None
```

2. Create a null vector of size 10

```
a=np.zeros(10)
```

3. Create Simple 1-D array and check type and check data types in array

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

```
print(a)
print(type(a))
print(a.dtype)
```

```
[1 2 3 4]
<class 'numpy.ndarray'>
int64
```

4. How to find number of dimensions, bytes per element and bytes of memory used?

```
print(a.ndim)
print(a.itemsize)
print(a.nbytes)
```

```
1
8
32
```

5. Create a null vector of size 10 but the fifth value which is 1

```
b=np.zeros(10)
b[4]=1
print(b)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

6. Create a vector with values ranging from 10 to 49

```
c=np.arange(10,49)
print(c)
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48]
```

7. Reverse a vector (first element becomes last)

```
print("Original->",c)
c[::-1]
```

```
Original-> [10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48]
array([48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32,
       31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15,
       14, 13, 12, 11, 10])
```

8. Create a 3x3 matrix with values ranging from 0 to 8

```
arr=np.arange(0,9)
print(arr)
arr.reshape(3,3)
```

```
[0 1 2 3 4 5 6 7 8]
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

9. Find indices of non-zero elements from [1,2,0,0,4,0]

```
a=np.array([1,2,0,0,4,0])
np.nonzero(a)
```

```
(array([0, 1, 4]),)
```

10. Create a 3x3 identity matrix

```
a=np.identity(3)
a
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

11. Create a 3x3x3 array with random values

```
a=np.random.random((3,3,3))
a
```

```
array([[[0.03019695, 0.91400571, 0.85368996],
        [0.83954276, 0.44034404, 0.01851509],
        [0.68570311, 0.528917  , 0.85456716]],

       [[0.96066752, 0.15352242, 0.50021987],
        [0.90181506, 0.71646244, 0.05555609],
        [0.9910813 , 0.2022268 , 0.29804974]],

       [[0.56388204, 0.48622515, 0.65567053],
        [0.01529096, 0.75307591, 0.2881835 ],
        [0.39952213, 0.36204728, 0.74873543]]])
```

12. Create a 10x10 array with random values and find the minimum and maximum values

```
a=np.random.random((10,10))
print(a)
print("mAximumu element-> ", np.max(a))
print("Minimum element-> ",np.min(a))
```

```
[[0.00643712 0.73682595 0.2216662  0.19750697 0.62802164 0.34202566
  0.01951598 0.30366503 0.88952348 0.60163877]
 [0.16960594 0.41165511 0.23009272 0.02976681 0.97495688 0.6974669
  0.4540809  0.52674064 0.03054289 0.63699754]
 [0.13837255 0.51760352 0.68053096 0.75424185 0.40306351 0.7409419
  0.2226207  0.01777046 0.61520726 0.62070694]
 [0.36777728 0.42404731 0.72649194 0.68588509 0.60058119 0.78010059
  0.3275787  0.91188882 0.6060301  0.33968914]
 [0.24996983 0.24721898 0.96374606 0.13402655 0.83667601 0.03275482
  0.88368912 0.46602069 0.24603813 0.82926483]
 [0.14309023 0.179693   0.93165595 0.18467138 0.62734918 0.17929439
  0.51881163 0.96470121 0.93790613 0.19377841]
 [0.08729371 0.35276851 0.91102087 0.2711576  0.25603401 0.17797015
  0.7168976  0.55926625 0.7188628  0.52836311]
 [0.55998689 0.66542068 0.79717329 0.40068699 0.88031327 0.55870131
  0.60396141 0.21542522 0.72611825 0.63343404]
 [0.11332174 0.191647   0.89358955 0.06752915 0.23915356 0.72399521
  0.15278568 0.38743446 0.6592267  0.42600497]
 [0.12261879 0.78859465 0.16191478 0.5680062  0.63847046 0.69187345
  0.11336696 0.61086851 0.30865166 0.82064132]]
mAximumu element->  0.9749568783165975
Minimum element->  0.006437115846314789
```

13. Create a random vector of size 30 and find the mean value

```
a=np.random.random((30))
print(a)
print("Mean Value->",np.mean(a))
```

```
[5.02404551e-02 1.49385603e-01 9.91509557e-01 5.00196279e-02
 5.15169300e-01 1.81439126e-01 1.53010370e-04 3.99773216e-01
 2.81156927e-01 7.08193333e-01 4.05197986e-01 8.85826418e-01
 8.25841207e-01 7.04547914e-01 4.73475226e-02 9.37346746e-01
 5.26728595e-01 2.54635762e-01 6.02730254e-01 2.16653810e-01
 5.10119206e-01 4.52451259e-02 5.07967495e-01 7.67874750e-01
 2.77441758e-01 8.02957937e-01 9.91469478e-01 8.95163525e-02
 1.48035870e-01 5.24809512e-02]
Mean Value-> 0.4309001764650699
```

14. Create a 2d array with 1 on the border and 0 inside

```
arra=np.zeros((4,4))
print("Original array with all zeros\n",arra)
```

```
arra[0::3,:]=1
arra[1:3,0::3]=1
print("\nNew array with 1 on the border and 0 inside\n",arra)
```

```
Original array with all zeros
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

New array with 1 on the border and 0 inside
 [[1. 1. 1. 1.]
 [1. 0. 0. 1.]
 [1. 0. 0. 1.]
 [1. 1. 1. 1.]]
```

15. How to add a border (filled with 0's) around an existing array?

```
a=np.arange(0,9)
a=a.reshape(3,3)
print("Existing array\n",a)
a[0::2,:]=0
a[1:2,0::2]=0
print("\nNew array\n",a)
```

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

New array
 [[0 0 0]
 [0 4 0]
 [0 0 0]]
```

16. How to Accessing/Changing specific elements, rows, columns, etc in Numpy array? Example - [[1 2 3 4 5 6 7] [8 9 10 11 12 13 14]]

Get 13, get first row only, get 3rd column only, get [2, 4, 6], replace 13 by 20

```
a=np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
print(a)
#to get 13
print("\n",a[1,5])
#to get first row only
print("\n",a[0:1,:])
#to get third column only
print("\n",a[:,3:4])
#get [2,4,6]
print("\n",a[0:1,1::2])
```

```
#replace 13 by 20
a[1,5]=20
```

```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]]

 13

 [[1 2 3 4 5 6 7]]

 [[ 4]
 [11]]

 [[2 4 6]]
```

17. How to Convert a 1D array to a 2D array with 2 rows

```
a=np.array([1,2,3,4,5,6,7,8,9,10])
a=a.reshape(2,5)
print(a)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
```

18. Create the following pattern without hardcoding. Use only numpy functions and the below input array a.

Input:

a = np.array([1,2,3])` Desired Output: array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])

```
a = np.array([1, 2, 3])
np.append(np.repeat(a,3), np.tile(a,3))
```

```
array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

19. Write a program to show how Numpy taking less memory compared to Python List?

```
l=range(1000)
import sys
a=10
#print(sys.getsizeof(a)) #memory allocated to integer a
print(sys.getsizeof(a)*len(l))
a1=np.arange(1000)
print(a1.size*a1.itemsize)
```

```
28000
```

```
8000
```

20. Write a program to show how Numpy taking less time compared to Python List?

```
import time
import sys
size=1000000
#for lists
l1= range(size)
l2=range(size)
start=time.time()
result=[(x+y) for x,y in zip(l1,l2)]
print((time.time()-start)*1000)
#for numpy array
n1=np.arange(size)
n2=np.arange(size)
start=time.time()
result1=n1+n2
print((time.time()-start)*1000)
```

```
95.87478637695312
2.170085906982422
```

✓ 0s completed at 09:25