20191206 - Friday

100_Numpy_Exercises

Q1. Find indices of Non_Zero elements from [1,2,0,0,4,0]

(Hint : use np.nonzero())

In [1]:

```
import numpy as np
```

In [2]:

```
a1 = [1,2,0,0,4,0]
np.nonzero(a1)
# it gives the output only indices number not the value
```

Out[2]:

```
(array([0, 1, 4], dtype=int64),)
```

In [3]:

```
b1 = ['x',0,1,'y',0]
print(np.nonzero(b1))
```

```
(array([0, 1, 2, 3, 4], dtype=int64),)
```

Note : Here it takes only the one data type even if you give input with diferent data types

In [4]:

```
print(np.__version__)
print(np.show_config())
```

```
1.16.5
mkl_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Users/Rudra/Anaconda3\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'C:/Users/Rudra/Anaconda3\\Library\\include']
blas_mkl_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Users/Rudra/Anaconda3\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'C:/Users/Rudra/Anaconda3\\Library\\include']
blas_opt_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Users/Rudra/Anaconda3\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'C:/Users/Rudra/Anaconda3\\Library\\include']
lapack_mkl_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Users/Rudra/Anaconda3\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'C:/Users/Rudra/Anaconda3\\Library\\include']
lapack_opt_info:
    libraries = ['mkl_rt']
    library_dirs = ['C:/Users/Rudra/Anaconda3\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:\\Program Files (x86)\\IntelSWTools\\compilers_and_
libraries_2019.0.117\\windows\\mkl', 'C:\\Program Files (x86)\\IntelSWTool
s\\compilers_and_libraries_2019.0.117\\windows\\mkl\\include', 'C:\\Progra
m Files (x86)\\IntelSWTools\\compilers_and_libraries_2019.0.117\\windows
\\mkl\\lib', 'C:/Users/Rudra/Anaconda3\\Library\\include']
None
```

Q2. Create a Null vector of size 10.?

In [4]:

```
print(np.zeros(10))
np.ones(10)
```

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

Out[4]:

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

4. How to get the documentation of the numpy add function from the command line?

python -c "import numpy; numpy.info(numpy.add)"

In [5]:

```
np.add(1,5)
```

Out[5]:

```
6
```

In [6]:

```
np.arange(9)
```

Out[6]:

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

In [8]:

```
x1 =  np.arange(9).reshape((3,3))
x1
```

Out[8]:

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

In [9]:

```
x2=np.arange(3)
x2
```

Out[9]:

```
array([0, 1, 2])
```

In [10]:

```
np.add(x1,x2)
```

Out[10]:

```
array([[ 0,  2,  4],
       [ 3,  5,  7],
       [ 6,  8, 10]])
```

Q5. Create a null vector of size 10 but the fifth value which is 1

In [11]:

```
x3 = np.zeros(10)
x3[4]=1
print(x3)
print(np.nonzero(x3))
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
(array([4], dtype=int64),)
```

Q6. Create a vector with values ranging from 10 to 49.?

In [13]:

```
np.arange(50,10)
```

Out[13]:

```
array([], dtype=int32)
```

Q7. Reverse a vector (first element becomes last)

In [14]:

```
np.arange(10,50)[::-1]
```

Out[14]:

```
array([49, 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])
```

Q8. Create a 3x3 matrix with values ranging from 0 to 8

In [15]:

```
np.arange(9).reshape((3,3))
```

Out[15]:

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

Q10. Create a 3x3 identity matrix

In [17]:

```
np.eye(4)
```

Out[17]:

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

In [29]:

```
a1=np.ones(9).reshape(3,3)
np.diag((1,1,1,1,1))
```

Out[29]:

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

Q11. Create a 3x3x3 array with random values

In [34]:

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

Out[34]:

```
array([[0.70680514, 0.21229302, 0.71990848],
       [0.41449314, 0.28809472, 0.62909245],
       [0.88147506, 0.74775064, 0.37482543]])
```

In [37]:

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

Out[37]:

```
array([[[0.81853415, 0.0478464 , 0.24026772],
        [0.76147244, 0.10327512, 0.85040397],
        [0.06282564, 0.47623315, 0.64232894]],

       [[0.91738724, 0.60493773, 0.47883706],
        [0.72429023, 0.5466045 , 0.87021666],
        [0.98826211, 0.72296235, 0.46830551]],

       [[0.59309136, 0.09938519, 0.63950272],
        [0.7781824 , 0.6364872 , 0.63353376],
        [0.12375442, 0.49918306, 0.55228553]]])
```

In [40]:

```
np.random.random((3,2))
```

Out[40]:

```
array([[0.79191157, 0.00376662],
       [0.65685693, 0.49726218],
       [0.19045165, 0.28764732]])
```

In [ ]:

```

```

In [43]:

```
np.random.random((3))
```

Out[43]:

```
array([0.54552694, 0.80825141, 0.78323331])
```

In [66]:

```
a11=np.random.random((3,2))
print(a11)
print(a11.ndim)
a12=np.random.random((3,2,1))
print(a12)
a12.ndim
```

```
[[0.14346402 0.58720547]
 [0.21359144 0.55446625]
 [0.15925889 0.41812058]]
2
[[[0.48859958]
  [0.7918154 ]]

 [[0.18369857]
  [0.70199138]]

 [[0.29332649]
  [0.81585046]]]
```

Out[66]:

```
3
```

In [67]:

```
print(np.arange(9).reshape(3,3).ndim)
np.arange(9).reshape(3,3,1).ndim
```

```
2
```

Out[67]:

```
3
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Q12. Create a 10x10 array with random values and find the minimum and maximum value

In [32]:

```
x4=np.random.random((10,10))
x4
```

Out[32]:

```
array([[0.08388711, 0.07081671, 0.11230704, 0.82203335, 0.08775375,
        0.75358797, 0.30271349, 0.62643365, 0.70860097, 0.81844678],
       [0.43907259, 0.05578271, 0.45583803, 0.95514474, 0.95989984,
        0.78165138, 0.79803467, 0.83116733, 0.10261892, 0.93714844],
       [0.52552262, 0.11854952, 0.31061741, 0.62164663, 0.34121513,
        0.66985829, 0.49818426, 0.51412965, 0.35018502, 0.06156678],
       [0.66936324, 0.3794674 , 0.50209511, 0.23416598, 0.37051114,
        0.82797794, 0.8302702 , 0.68619308, 0.39376845, 0.13189691],
       [0.74776065, 0.73536189, 0.65175856, 0.99011478, 0.67424581,
        0.99345106, 0.9432167 , 0.8649774 , 0.82468744, 0.45985762],
       [0.18370811, 0.59956615, 0.92771921, 0.9636544 , 0.43768155,
        0.25586438, 0.15568027, 0.76500513, 0.53259754, 0.45775518],
       [0.44504695, 0.64703326, 0.86420889, 0.46498339, 0.20566366,
        0.16487747, 0.93342874, 0.55364532, 0.8860904 , 0.68557739],
       [0.54388519, 0.32496292, 0.14312694, 0.64115758, 0.44075271,
        0.46279334, 0.51144943, 0.7740021 , 0.5172623 , 0.46008862],
       [0.10832386, 0.1216372 , 0.89102304, 0.98210192, 0.20412897,
        0.4248664 , 0.65286541, 0.18797564, 0.94866476, 0.05702854],
       [0.38919373, 0.20314928, 0.66255884, 0.62179986, 0.08192801,
        0.17756985, 0.32198508, 0.85706139, 0.20045462, 0.14262334]])
```

In [33]:

```
x4.min(),x4.max(),x4.mean()
```

Out[33]:

```
(0.05578271197536133, 0.993451057120584, 0.5181176440884474)
```

In [50]:

```
min(x4[2])#min(x4[1:-1,1:-1])
```

Out[50]:

```
0.06156678466009846
```

Q13. Create a random vector of size 30 and find the mean value

In [45]:

```
print(np.random.random(10).mean())
print(np.arange(10).mean())
```

```
0.37828985220529315
4.5
```

Q14. Create a 2d array with 1 on the border and 0 inside

In [48]:

```
x5=np.ones((5,5))[1:-1]=0
x5
```

Out[48]:

```
0
```

In [51]:

```
x6=np.ones((5,5))
#x6
x6[1:-1,1:-1]=0
x6
```

Out[51]:

```
array([[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.]])
```

In [52]:

```
x6[1:-1]=0
x6
```

Out[52]:

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

Q15. What is the result of the following expression?

In [68]:

```
0 * np.nan
np.nan == np.nan
np.inf > np.nan
np.nan - np.nan
0.3 == 3 * 0.1
```

Out[68]:

```
False
```

In [83]:

```
b12=np.nan
print(type(b12))
print(5.0 * np.nan)
print(np.nan == np.nan)
a='b'
print(10*a)
c='b'
if 'b'=='b':
    print("yes")
print(np.inf > np.nan)
print(np.nan - np.nan)
print(0.3 == (3 * 0.1))
print(3*0.1)
if (0.3==(1*0.1)):
    print("yesssssssssssss")
```

```
<class 'float'>
nan
False
bbbbbbbbbb
yes
False
nan
False
0.30000000000000004
```

In [70]:

```
print(np.arange(10)*0.1)
l1=[1,2,3,4,5,6,7,8,9]
for i in l1:
    print(i*0.1,end=" ")
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
0.1 0.2 0.30000000000000004 0.4 0.5 0.6000000000000001 0.7000000000000001
0.8 0.9
```

In [ ]:

```

```

In [84]:

```
print(2*'a')
```

```
aa
```

In [73]:

```
0.3 == (3*(0.1))
```

Out[73]:

```
0.30000000000000004
```

In [86]:

```
(3*(0.1))==0.3
```

Out[86]:

```
False
```

Q16. Create a 5x5 matrix with values 1,2,3,4 just below the diagonal (★☆☆)

In [84]:

```
x7 = np.diag(1+np.arange(4),k=-1)
print(x7)
```

```
[[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]]
```

In [90]:

```
np.diag(np.arange(1,4))
```

Out[90]:

```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [91]:

```
np.diag(np.arange(4))
```

Out[91]:

```
array([[0, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 2, 0],
       [0, 0, 0, 3]])
```

In [92]:

```
np.diag(1+np.arange(4))
```

Out[92]:

```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

In [93]:

```
np.diag(1+np.arange(4),k=-1)
```

Out[93]:

```
array([[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]])
```

In [96]:

```
np.diag(1+np.arange(4),k=-2)
```

Out[96]:

```
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0],
       [0, 2, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 0],
       [0, 0, 0, 4, 0, 0]])
```

In [97]:

```
np.diag(1+np.arange(4),k=-3)
```

Out[97]:

```
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0],
       [0, 2, 0, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 0, 0],
       [0, 0, 0, 4, 0, 0, 0]])
```

In [98]:

```
np.diag(1+np.arange(4),k=1)
```

Out[98]:

```
array([[0, 1, 0, 0, 0],
       [0, 0, 2, 0, 0],
       [0, 0, 0, 3, 0],
       [0, 0, 0, 0, 4],
       [0, 0, 0, 0, 0]])
```

In [100]:

```
np.diag(1+np.arange(4),j=2)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-100-04e50cf9eb03> in <module>
----> 1 np.diag(1+np.arange(4),j=2)

TypeError: diag() got an unexpected keyword argument 'j'
```

In [ ]:

```

```

Q17. Create a 8x8 matrix and fill it with a checkerboard pattern (★☆☆)

In [91]:

```
x8 = np.zeros((8,8),dtype=int)
x8[1::2,::2] = 1
x8[::2,1::2] = 1
print(x8)
```

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

In [116]:

```
x9 = np.zeros((8,8),dtype=int)
x9[1::2,::2] = 1
x9[::2,1::2] = 2
x9
```

Out[116]:

```
array([[0, 2, 0, 2, 0, 2, 0, 2],
       [1, 0, 1, 0, 1, 0, 1, 0],
       [0, 2, 0, 2, 0, 2, 0, 2],
       [1, 0, 1, 0, 1, 0, 1, 0],
       [0, 2, 0, 2, 0, 2, 0, 2],
       [1, 0, 1, 0, 1, 0, 1, 0],
       [0, 2, 0, 2, 0, 2, 0, 2],
       [1, 0, 1, 0, 1, 0, 1, 0]])
```

re-started @ 1:50am on 20191208

Q18)Consider a (6,7,8) shape array, what is the index (x,y,z) of the 100th element?

In [117]:

```
print(np.unravel_index(100,(6,7,8)))
```

```
(1, 5, 4)
```

In [128]:

```
np.unravel_index(100,(2,4,6))
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-128-fc37330879b4> in <module>
----> 1 np.unravel_index(100,(2,4,6))

<__array_function__ internals> in unravel_index(*args, **kwargs)

ValueError: index 100 is out of bounds for array with size 48
```

19. Create a checkerboard 8x8 matrix using the tile function (★☆☆)

In [129]:

```
Z = np.tile( np.array([[0,1],[1,0]]), (4,4))
print(Z)
```

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

In [ ]:

In [ ]:

In [ ]:

q20. Normalize a 5x5 random matrix (★☆☆)

In [130]:

```
Z = np.random.random((5,5))
Zmax, Zmin = Z.max(), Z.min()
Z = (Z - Zmin)/(Zmax - Zmin)
print(Z)
```

```
[[0.03310105 0.33664747 0.18561869 0.21653684 1.        ]
 [0.05727113 0.90587698 0.24684716 0.57808044 0.73032725]
 [0.74345048 0.26722843 0.85711439 0.49021902 0.56656787]
 [0.58866396 0.86108448 0.99186767 0.61933051 0.        ]
 [0.91259549 0.80871758 0.44894368 0.77824663 0.39074435]]
```

In [ ]:

In [ ]:

In [ ]: