NumPy Tutorials (CWH)

Video link: https://youtu.be/Rbh1rieb3zc (https://youtu.be/Rbh1rieb3zc)

Array creation in numpy

(source - google) There are 6 general mechanisms for creating arrays:

1. Conversion from other Python structures (i.e. lists and tuples)
2. Intrinsic NumPy array creation functions (e.g. arange, ones, zeros, etc.)
3. Replicating, joining, or mutating existing arrays
4. Reading arrays from disk, either from standard or custom formats
5. Creating arrays from raw bytes through the use of strings or buffers
6. Use of special library functions (e.g., random)

1. Conversion from other Python structures (i.e. lists and tuples)

In [1]:
```
import numpy as np
```

In [2]:
```
print('new in Jupyeter')
```

```
new in Jupyeter
```

In [3]:
```
myarr1= np.array([[4,5,8],[6,4,15]], np.int16)
```

In [4]:
```
 myarr1
```

Out[4]:
```
array([[ 4,  5,  8],
       [ 6,  4, 15]], dtype=int16)
```

In [5]:
```
myarr1[1,0]
```

Out[5]:
```
6
```

In [6]:
```
myarr1.dtype
```

Out[6]:
```
dtype('int16')
```

changing an element

In [7]:
```
myarr1[1,2]= 12
```

In [8]:
```
myarr1[1,2]
```

Out[8]:
```
12
```

changing memory from int16 to float64

In [9]:
```
myarr2= np.array([[8,47,51],[5445,48,85.01],[1.250,12,56]], np.float64)
```

In [10]:
```
myarr2
```

Out[10]:
```
array([[8.000e+00, 4.700e+01, 5.100e+01],
       [5.445e+03, 4.800e+01, 8.501e+01],
       [1.250e+00, 1.200e+01, 5.600e+01]])
```

In [11]:
```
myarr2.shape
```

Out[11]:
```
(3, 3)
```

In [12]:
```
myarr2.size
```

Out[12]:
```
9
```

In [13]:
```
myarr2.dtype
```

Out[13]:
```
dtype('float64')
```

In [ ]:

In [14]:
```
myarr3= np.array([[756466836,4,7842],[5.5,487,84],[9.4,18.6,43]])
```

In [15]:
```
myarr3
```

Out[15]:
```
array([[7.56466836e+08, 4.00000000e+00, 7.84200000e+03],
       [5.50000000e+00, 4.87000000e+02, 8.40000000e+01],
       [9.40000000e+00, 1.86000000e+01, 4.30000000e+01]])
```

In [16]:
```
myarr3.dtype
```

Out[16]:
```
dtype('float64')
```

To know more, google *"numpy types reference"*.

In [17]:
```
np.array({45,655,55})
```

Out[17]:
```
array({655, 45, 55}, dtype=object)
```

In [ ]:

2. Intrinsic NumPy array creation functions (e.g. arange, ones, zeros, etc.)

zeros - makes an array filled with zeros for a given shape

In [18]:
```
zero = np.zeros((3,4))
```

In [19]:
```
zero
```

Out[19]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [20]:
```
zero.shape
```

Out[20]:
```
(3, 4)
```

In [21]:
```
zero.size
```

Out[21]:
```
12
```

In [22]:
```
zero.dtype
```

Out[22]:
```
dtype('float64')
```

range - makes a numpy array

In [23]:
```
rng= np.arange(12)
```

In [24]:
```
rng
```

Out[24]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

linspace(a,b,c) - gives c equidistant numbers between a and b

In [25]:
```
lsp= np.linspace(1,100,10)
```

In [26]:
```
lsp
```

Out[26]:
```
array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.])
```

empty - gives random filled array for a given shape

In [27]:
```
emp= np.empty((2,4))
```

In [28]:
```
emp
```

Out[28]:
```
array([[6.23042070e-307, 4.67296746e-307, 1.69121096e-306,
        1.29062229e-306],
       [1.89146896e-307, 7.56571288e-307, 3.11525958e-307,
        1.24610723e-306]])
```

In [29]:
```
np.empty_like(lsp)
```

Out[29]:
```
array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.])
```

identity - gives identity matrixx of given order

In [30]:
```
I6= np.identity(6)
```

In [31]:
```
I6
```

Out[31]:
```
array([[1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

In [32]:
```
I6.shape
```

Out[32]:
```
(6, 6)
```

some more array functions: reshape(change order), ravel(convert to 1D araay)

In [33]:
```
arr1= np.array([[5,6,8,7],[4,5,5,4.7],[5,6,10,1.99]])
```

In [34]:
```
arr1.reshape(6,2)
```

Out[34]:
```
array([[ 5.  ,  6.  ],
       [ 8.  ,  7.  ],
       [ 4.  ,  5.  ],
       [ 5.  ,  4.7 ],
       [ 5.  ,  6.  ],
       [10.  ,  1.99]])
```

In [35]:
```
arr1
```

Out[35]:
```
array([[ 5.  ,  6.  ,  8.  ,  7.  ],
       [ 4.  ,  5.  ,  5.  ,  4.7 ],
       [ 5.  ,  6.  , 10.  ,  1.99]])
```

In [36]:
```
arr1= arr1.reshape(2,6)
```

In [37]:
```
arr1
```

Out[37]:
```
array([[ 5.  ,  6.  ,  8.  ,  7.  ,  4.  ,  5.  ],
       [ 5.  ,  4.7 ,  5.  ,  6.  , 10.  ,  1.99]])
```

In [38]:
```
arr1.ravel()
```

Out[38]:
```
array([ 5.  ,  6.  ,  8.  ,  7.  ,  4.  ,  5.  ,  5.  ,  4.7 ,  5.  ,
        6.  , 10.  ,  1.99])
```

In [39]:
```
arr1.shape
```

Out[39]:
```
(2, 6)
```

In [40]:
```
arr1= arr1.ravel()
```

In [41]:
```
arr1.shape
```

Out[41]:
```
(12,)
```

numpy axis

1D axis - 1 axis - axis0

2D axis - 2 axes - rows(axis0) and columns(axis1)

we can do operations on the axes

In [42]:
```
A= np.array([[6,9,4],[2,1.5,11.4],[5,5,1]])
```

In [43]:
```
A
```

Out[43]:
```
array([[ 6. ,  9. ,  4. ],
       [ 2. ,  1.5, 11.4],
       [ 5. ,  5. ,  1. ]])
```

In [44]:
```
A.ndim
```

Out[44]:
```
2
```

In [45]:
```
A.nbytes
```

Out[45]:
```
72
```

In [46]:
```
A.sum(axis=0)
```

Out[46]:
```
array([13. , 15.5, 16.4])
```

In [47]:
```
A.sum(axis=1)
```

Out[47]:
```
array([19. , 14.9, 11. ])
```

In [48]:
```
A.sum()
```

Out[48]:
```
44.9
```

In [49]:
```
A.max()
```

Out[49]:
```
11.4
```

In [50]:
```
A.min()
```

Out[50]:
```
1.0
```

Transpose of matrix A

In [51]:
```
A.T
```

Out[51]:
```
array([[ 6. ,  2. ,  5. ],
       [ 9. ,  1.5,  5. ],
       [ 4. , 11.4,  1. ]])
```

In [52]:
```
A.flat
```

Out[52]:
```
<numpy.flatiter at 0x2230ab56a00>
```

In [53]:
```
for item in A:
    print(item)
```

```
[6. 9. 4.]
[ 2.   1.5 11.4]
[5. 5. 1.]
```

In [54]:
```
for item in A.flat:
    print(item)
```

```
6.0
9.0
4.0
2.0
1.5
11.4
5.0
5.0
1.0
```

argmax or argmin - first converts the matrix to 1D and then gives the maximum or minimum positions.

In [55]:
```
A.argmax()
```

Out[55]:
```
5
```

In [56]:
```
A.argmin()
```

Out[56]:
```
8
```

In [57]:
```
A.argmin(axis=1)
```

Out[57]:
```
array([2, 1, 2], dtype=int64)
```

In [58]:
```
A.argmax(axis=0)
```

Out[58]:
```
array([0, 0, 1], dtype=int64)
```

In [59]:
```
A.argsort()
```

Out[59]:
```
array([[2, 0, 1],
       [1, 0, 2],
       [2, 0, 1]], dtype=int64)
```

In [60]:
```
A.argsort(axis=0)
```

Out[60]:
```
array([[1, 1, 2],
       [2, 2, 0],
       [0, 0, 1]], dtype=int64)
```

In [61]:
```
A.argsort(axis=1)
```

Out[61]:
```
array([[2, 0, 1],
       [1, 0, 2],
       [2, 0, 1]], dtype=int64)
```

NOT UNDERSTOOD - argsort in 2D and argmax,argmin with axis in 2D addition and subtraction of matrices

In [62]:
```
A
```

Out[62]:
```
array([[ 6. ,  9. ,  4. ],
       [ 2. ,  1.5, 11.4],
       [ 5. ,  5. ,  1. ]])
```

In [63]:
```
B= np.array([[3,9,5],[5,2,0],[2,5,3]])
```

In [64]:
```
A+B
```

Out[64]:
```
array([[ 9. , 18. ,  9. ],
       [ 7. ,  3.5, 11.4],
       [ 7. , 10. ,  4. ]])
```

In [65]:
```
A-B
```

Out[65]:
```
array([[ 3. ,  0. , -1. ],
       [-3. , -0.5, 11.4],
       [ 3. ,  0. , -2. ]])
```

In [66]:
```
np.sqrt(A+B)
```

Out[66]:
```
array([[3.        , 4.24264069, 3.        ],
       [2.64575131, 1.87082869, 3.3763886 ],
       [2.64575131, 3.16227766, 2.        ]])
```

In [67]:
```
(A+B)**1.5
```

Out[67]:
```
array([[27.        , 76.36753237, 27.        ],
       [18.52025918,  6.54790043, 38.49083008],
       [18.52025918, 31.6227766 ,  8.        ]])
```

In [68]:
```
np.where((A-B)<0)
```

Out[68]:
```
(array([0, 1, 1, 2], dtype=int64), array([2, 0, 1, 2], dtype=int64))
```

write the matrix. the pairs will be in columns

In [69]:
```
np.nonzero(A-B)
```

Out[69]:
```
(array([0, 0, 1, 1, 1, 2, 2], dtype=int64),
 array([0, 2, 0, 1, 2, 0, 2], dtype=int64))
```

In [70]:
```
np.count_nonzero(A-B)
```

Out[70]:
```
7
```

In [71]:
```
import sys
```

In [72]:
```
py_C= [5,1,9,7]
```

In [73]:
```
np_C= np.array(py_C)
```

In [74]:
```
sys.getsizeof(1)*len(py_C)
```

Out[74]:
```
112
```

In [75]:
```
np_C.itemsize * np_C.size
```

Out[75]:
```
16
```

Hence, numpy array takes 16 bytes only, i.e. lesser space

To know more about all these, google **"numpy array methods and attributes"** and go to the website of docs.scipy.org

In [ ]:
```

```