Numpy in python

NumPy is a Python library used for working with arrays and primarily for numerical and scientific computing. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Why Use NumPy?

1. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
2. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

Which Language is NumPy written in?

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

In [1]:
```
pip install numpy
```
```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (1.19.
5)
Note: you may need to restart the kernel to use updated packages.
```

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

# 1d array

arr=[1,2,3,4,5]

np.array(arr)
```
Out[1]:
```
array([1, 2, 3, 4, 5])
```

In [3]:
```
# 2d array

my_mat=[[1,2,3],[4,5,6],[7,8,9]]

np.array(my_mat)
```
Out[3]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Below is some numpy methods for arrays

1. arange() - to generate array just like range function

In [29]:
```
np.arange(0,11)
```
Out[29]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [30]:
```
np.arange(0,11,2) # adding step size
```
Out[30]:
```
array([ 0,  2,  4,  6,  8, 10])
```

2. zeros() - to generate array with zeros

In [31]:
```
np.zeros(5)
```
Out[31]:
```
array([0., 0., 0., 0., 0.])
```

In [32]:
```
np.zeros((5,5)) # zeroes 2-d matrix
```
Out[32]:
```
array([[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.]])
```

3. ones() - to generate array with ones

In [33]:
```
np.ones(5)
```
Out[33]:
```
array([1., 1., 1., 1., 1.])
```

4. eye() method - to make Identity matrix.

In [16]:
```
np.eye(4)
```
Out[16]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

5. linspace() - to generate evenly spaced numbers over a specified interval.

In [15]:
```
np.linspace(0,1,100)
```
Out[15]:
```
array([0.        , 0.01010101, 0.02020202, 0.03030303, 0.04040404,
       0.05050505, 0.06060606, 0.07070707, 0.08080808, 0.09090909,
       0.1010101 , 0.11111111, 0.12121212, 0.13131313, 0.14141414,
       0.15151515, 0.16161616, 0.17171717, 0.18181818, 0.19191919,
       0.2020202 , 0.21212121, 0.22222222, 0.23232323, 0.24242424,
       0.25252525, 0.26262626, 0.27272727, 0.28282828, 0.29292929,
       0.3030303 , 0.31313131, 0.32323232, 0.33333333, 0.34343434,
       0.35353535, 0.36363636, 0.37373737, 0.38383838, 0.39393939,
       0.4040404 , 0.41414141, 0.42424242, 0.43434343, 0.44444444,
       0.45454545, 0.46464646, 0.47474747, 0.48484848, 0.49494949,
       0.50505051, 0.51515152, 0.52525253, 0.53535354, 0.54545455,
       0.55555556, 0.56565657, 0.57575758, 0.58585859, 0.5959596 ,
       0.60606061, 0.61616162, 0.62626263, 0.63636364, 0.64646465,
       0.65656566, 0.66666667, 0.67676768, 0.68686869, 0.6969697 ,
       0.70707071, 0.71717172, 0.72727273, 0.73737374, 0.74747475,
       0.75757576, 0.76767677, 0.77777778, 0.78787879, 0.7979798 ,
       0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485,
       0.85858586, 0.86868687, 0.87878788, 0.88888889, 0.8989899 ,
       0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495,
       0.95959596, 0.96969697, 0.97979798, 0.98989899, 1.        ])
```

6. reshape() - It changes the dimension of array

In [23]:
```
arr= np.arange(1,16)
arr
```
Out[23]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

In [24]:
```
arr.reshape(3,5)
```
Out[24]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

7. min() and max() methods on arrays

In [25]:
```
arr.max()
```
Out[25]:
```
15
```

In [27]:
```
arr.min()
```
Out[27]:
```
1
```

In [28]:
```
arr.argmax() # it returns index value of max element
```
Out[28]:
```
14
```

8. dtype() - to get datatype of array

In [34]:
```
arr.dtype
```
Out[34]:
```
dtype('int64')
```

9. random module

In [17]:
```
np.random.rand(5)
```
Out[17]:
```
array([0.99073827, 0.96626483, 0.91757493, 0.92252976, 0.62171637])
```

In [18]:
```
np.random.rand(5,5)
```
Out[18]:
```
array([[0.19707334, 0.38562581, 0.67980184, 0.78631147, 0.43902611],
       [0.46415045, 0.00664001, 0.80238642, 0.16118304, 0.4207219 ],
       [0.68200941, 0.43007264, 0.03703333, 0.0462983 , 0.43096494],
       [0.47119373, 0.8451655 , 0.20813501, 0.85848234, 0.31707059],
       [0.82420619, 0.69601511, 0.37139566, 0.2097775 , 0.09517707]])
```

In [19]:
```
np.random.randn(4,4)
```
Out[19]:
```
array([[ 0.34584626,  0.41696322, -0.35319161, -0.09439203],
       [ 0.65520298,  0.89900956,  0.12269491,  0.53173262],
       [ 0.14812314,  0.47682511, -0.89518899,  0.63055104],
       [ 0.47227976,  0.59834106,  1.26414908,  0.46707973]])
```

In [21]:
```
np.random.randint(1,100,10)
```
Out[21]:
```
array([64, 78, 46, 15, 89, 47,  4, 86, 42, 14])
```

Numpy Indexing & Selection

In [36]:
```
arr=np.arange(1,11)
arr
```
Out[36]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [37]:
```
arr[0] # indexing with place
```
Out[37]:
```
1
```

In [38]:
```
arr[0:5] # indexing with start, stop
```
Out[38]:
```
array([1, 2, 3, 4, 5])
```

In [39]:
```
arr[:] # indexing from star to end
```
Out[39]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [40]:
```
arr[5::2] # indexing from 5 to end with step=2
```
Out[40]:
```
array([ 6,  8, 10])
```

In [43]:
```
arr[0:5]=100 # broadcasting on arrays
arr
```
Out[43]:
```
array([100, 100, 100, 100, 100,   6,   7,   8,   9,  10])
```

Slice affects actual array can be seen below -

Reason: numpy do this is, because of memmory issues with very large arrays it don't make copies of array automatically.

In [46]:
```
slice_of_arr=arr[0:6]
slice_of_arr
```
Out[46]:
```
array([100, 100, 100, 100, 100,   6])
```

In [47]:
```
arr # actual arr
```
Out[47]:
```
array([100, 100, 100, 100, 100,   6,   7,   8,   9,  10])
```

In [48]:
```
slice_of_arr[:]=99
```

In [49]:
```
arr #arr got changed as seen
```
Out[49]:
```
array([99, 99, 99, 99, 99, 99,  7,  8,  9, 10])
```

To avoid that we use copy() method specifically.

In [50]:
```
arr_copy= arr.copy()
arr_copy
```
Out[50]:
```
array([99, 99, 99, 99, 99, 99,  7,  8,  9, 10])
```

In [51]:
```
arr_copy[:]=101
arr_copy
```
Out[51]:
```
array([101, 101, 101, 101, 101, 101, 101, 101, 101, 101])
```

In [52]:
```
arr # actual array didn't changed
```
Out[52]:
```
array([99, 99, 99, 99, 99, 99,  7,  8,  9, 10])
```

Indexing of a 2-D array.

In [63]:
```
arr_2d=np.arange(1,13).reshape(3,4)
arr_2d
```
Out[63]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [65]:
```
arr_2d[0][1] # array indexing using dounle braces.
```
Out[65]:
```
2
```

In [64]:
```
arr_2d[0,1] # array indexing using single braces.
```
Out[64]:
```
2
```

Important slicing to grab sub-section of 2-D matrix.

In [68]:
```
arr_2d[:2,2:4]
```
Out[68]:
```
array([[3, 4],
       [7, 8]])
```

In [71]:
```
arr_2d[0:1]  # 1st row
```
Out[71]:
```
array([[1, 2, 3, 4]])
```

In [73]:
```
arr_2d[:,0:1] # 1st column
```
Out[73]:
```
array([[1],
       [5],
       [9]])
```

Conditional selection or slicing

In [74]:
```
arr=np.arange(0,15)
arr
```
Out[74]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [76]:
```
arr>5 # gives a bool array
```
Out[76]:
```
array([False, False, False, False, False, False,  True,  True,  True,
        True,  True,  True,  True,  True,  True])
```

In [79]:
```
exp=arr>5
```

In [80]:
```
arr[exp] # filters array element
```
Out[80]:
```
array([ 6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [81]:
```
arr
```
Out[81]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [82]:
```
arr[arr>3] # better way to write without variable
```
Out[82]:
```
array([ 4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

Numpy Operations

- **Array with Array**
- **Array with scalar**
- **Universal array functions. (ufunc)**

Array with Array

In [86]:
```
arr=np.arange(1,11)
```
Out[86]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [87]:
```
arr+arr
```
Out[87]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [88]:
```
arr*arr
```
Out[88]:
```
array([  1,   4,   9,  16,  25,  36,  49,  64,  81, 100])
```

In [89]:
```
arr-arr
```
Out[89]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [91]:
```
arr+100 #array with scalar
```
Out[91]:
```
array([101, 102, 103, 104, 105, 106, 107, 108, 109, 110])
```

In [92]:
```
arr/arr
```
Out[92]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [93]:
```
arr/0
```
```
/home/clecotech/.local/lib/python3.6/site-packages/ipykernel_launcher.py:1: RuntimeWar
ning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```
Out[93]:
```
array([inf, inf, inf, inf, inf, inf, inf, inf, inf, inf])
```

Error: In general, anything divided by 0 will give pyhton error. However, numpy will not give error instead will give a warning and execute further as above and "inf" means infinity.

In [95]:
```
np.sqrt(arr)  #universal array functions
```
Out[95]:
```
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974, 2.64575131, 2.82842712, 3.        , 3.16227766])
```

In [96]:
```
np.exp(arr) # calculate exponential
```
Out[96]:
```
array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
       1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
       8.10308393e+03, 2.20264658e+04])
```

In [98]:
```
np.max(arr) # to get max element in array
```
Out[98]:
```
10
```

@numpy notes by rajveer mehta

In [ ]:
```

```