numpy

February 9, 2018

1 Numpy

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

```
In [2]: # Array Creation

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

```
[2 3 4]
```

```
Out[2]: dtype('int64')
```

```
In [3]: a = np.array([2.1,3,4])
        print(a)
        a.dtype
```

```
[ 2.1  3.   4. ]
```

```
Out[3]: dtype('float64')
```

```
In [4]: a = np.array([[2,3,4], [5,6,7]]) # sequence of sequences; 2D array
        print(a)
```

```
[[2 3 4]
 [5 6 7]]
```

```
In [5]: a = np.array([2,3,4], dtype = float)
        print(a)
```

```
[ 2.  3.  4.]
```

```
In [6]: # create a 3x4 matrix of zeros
        np.zeros([3,4], dtype = int)
```

```
Out[6]: array([[0, 0, 0, 0],
               [0, 0, 0, 0],
               [0, 0, 0, 0]])
```

```
In [7]: np.ones([2,3,4])
```

```
Out[7]: array([[[ 1.,  1.,  1.,  1.],
                [ 1.,  1.,  1.,  1.],
                [ 1.,  1.,  1.,  1.]],

               [[ 1.,  1.,  1.,  1.],
                [ 1.,  1.,  1.,  1.],
                [ 1.,  1.,  1.,  1.]]])
```

```
In [8]: #initialize an empty arrary
        np.empty([2,3])
```

```
Out[8]: array([[  9.88131292e-324,   1.48219694e-323,   1.97626258e-323],
               [  2.47032823e-323,   2.96439388e-323,   3.45845952e-323]])
```

```
In [9]: np.arange(10,20,5) #  returns an array from 10 to 20 with steps of 5, excluding 20
```

```
Out[9]: array([10, 15])
```

```
In [10]: np.arange(0,2,0.2) #steps of 0.2
```

```
Out[10]: array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ,  1.2,  1.4,  1.6,  1.8])
```

```
In [11]: np.linspace(0,2,9) # 9 numbers from 0 to 2.. instead of step
```

```
Out[11]: array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])
```

```
In [12]: from numpy import pi
         x = np.linspace(0, 2*pi, 100)
         x
```

```
Out[12]: array([ 0.        ,  0.06346652,  0.12693304,  0.19039955,  0.25386607,
                 0.31733259,  0.38079911,  0.44426563,  0.50773215,  0.57119866,
                 0.63466518,  0.6981317 ,  0.76159822,  0.82506474,  0.88853126,
                 0.95199777,  1.01546429,  1.07893081,  1.14239733,  1.20586385,
                 1.26933037,  1.33279688,  1.3962634 ,  1.45972992,  1.52319644,
                 1.58666296,  1.65012947,  1.71359599,  1.77706251,  1.84052903,
                 1.90399555,  1.96746207,  2.03092858,  2.0943951 ,  2.15786162,
                 2.22132814,  2.28479466,  2.34826118,  2.41172769,  2.47519421,
                 2.53866073,  2.60212725,  2.66559377,  2.72906028,  2.7925268 ,
                 2.85599332,  2.91945984,  2.98292636,  3.04639288,  3.10985939,
                 3.17332591,  3.23679243,  3.30025895,  3.36372547,  3.42719199,
                 3.4906585 ,  3.55412502,  3.61759154,  3.68105806,  3.74452458,
                 3.8079911 ,  3.87145761,  3.93492413,  3.99839065,  4.06185717,
                 4.12532369,  4.1887902 ,  4.25225672,  4.31572324,  4.37918976,
                 4.44265628,  4.5061228 ,  4.56958931,  4.63305583,  4.69652235,
                 4.75998887,  4.82345539,  4.88692191,  4.95038842,  5.01385494,
                 5.07732146,  5.14078798,  5.2042545 ,  5.26772102,  5.33118753,
                 5.39465405,  5.45812057,  5.52158709,  5.58505361,  5.64852012,
                 5.71198664,  5.77545316,  5.83891968,  5.9023862 ,  5.96585272,
                 6.02931923,  6.09278575,  6.15625227,  6.21971879,  6.28318531])
```

```
In [13]: a= np.arange(6)
         print(a)
```

```
[0 1 2 3 4 5]
```

```
In [14]: b = np.arange(12).reshape(4,3)
         print(b)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

1.0.1 Basic Operations

```
In [15]: a = np.array([20,30,40,50])
         b = np.arange(4)
         print(a,b)
```

```
[20 30 40 50] [0 1 2 3]
```

```
In [16]: print(a-b)
```

```
[20 29 38 47]
```

```
In [17]: print(b**2)
```

```
[0 1 4 9]
```

```
In [18]: print(a<35)
```

```
[ True  True False False]
```

```
In [19]: A = np.array([[1,1], [0,1]])
         print(A)
```

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

```
In [20]: B = np.array([(2,0), (3,4)])
         print(B)

[[2 0]
 [3 4]]


In [21]: # elementwise product
         A*B

Out[21]: array([[2, 0],
                [0, 4]])

In [22]: # Matrix product
         A.dot(B)

Out[22]: array([[5, 4],
                [3, 4]])

In [23]: # or
         np.dot(A,B)

Out[23]: array([[5, 4],
                [3, 4]])

In [24]: a = np.ones([2,3])
         b = np.random.random([2,3])
         print(a)
         print(b)

[[ 1.  1.  1.]
 [ 1.  1.  1.]]
[[ 0.33957925  0.86965007  0.02455916]
 [ 0.75069577  0.64871184  0.41545429]]


In [25]: a *=3
         print(a)

[[ 3.  3.  3.]
 [ 3.  3.  3.]]


In [26]: b +=a
         print(b)

[[ 3.33957925  3.86965007  3.02455916]
 [ 3.75069577  3.64871184  3.41545429]]
```

```
In [27]: b.sum()

Out[27]: 21.048650392728344

In [28]: b.min()

Out[28]: 3.0245591627309651

In [29]: b.max()

Out[29]: 3.869650070235461

In [30]: np.random.seed(1000)
         a = np.random.randint(1,100, 12).reshape(3,4)
         print(a)

[[52 88 72 65]
 [95 93  2 62]
 [ 1 90 46 41]]


In [31]: b.sum()

Out[31]: 21.048650392728344

In [32]: b.sum(axis = 0) # adding elements in each column

Out[32]: array([ 7.09027502,  7.51836191,  6.44001346])

In [33]: b.min(axis = 1) # finding min in each row

Out[33]: array([ 3.02455916,  3.41545429])

In [34]: b.cumsum(axis = 1)

Out[34]: array([[  3.33957925,   7.20922932,  10.23378848],
                [  3.75069577,   7.39940762,  10.81486191]])
```

1.1 Indexing, Slicing, and Iterating

```
In [35]: a = np.arange(10)**2
         print(a)

[ 0  1  4  9 16 25 36 49 64 81]


In [36]: a[2:5]

Out[36]: array([ 4,  9, 16])

In [37]: a[0:6:2] # start at 0 end at 6, steps = 2
```

```
Out[37]: array([ 0,  4, 16])

In [38]: a[:6:2] # similar to the previous command

Out[38]: array([ 0,  4, 16])

In [39]: a[:6:2] = 1000
         print(a)

[1000    1 1000    9 1000   25   36   49   64   81]


In [40]: a[::-1] # reversed a

Out[40]: array([  81,   64,   49,   36,   25, 1000,    9, 1000,    1, 1000])

In [41]: for i in a:
             print(i**(1/3))

10.0
1.0
10.0
2.08008382305
10.0
2.92401773821
3.30192724889
3.65930571002
4.0
4.32674871092


In [42]: a = np.arange(30, 50).reshape(4,5)
         print(a)

[[30 31 32 33 34]
 [35 36 37 38 39]
 [40 41 42 43 44]
 [45 46 47 48 49]]


In [43]: #traverse the array using for loops

         for i in range(4):
             for j in range(5):
                 print(a[i,j])

30
31
32
33
```

```
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

```
In [44]: a = np.arange(0, 100, 5).reshape(5,4)
         print(a)


         for i in range(a.shape[0]):
             for j in range(a.shape[1]):
                 if(a[i,j] % 10 !=0):
                     print(a[i,j])
```

```
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]
 [60 65 70 75]
 [80 85 90 95]]
5
15
25
35
45
55
65
75
85
95
```

```
In [45]: num_list = [1, 2, 3]
         alpha_list = ['a', 'b', 'c']

         for number in num_list:
             print(number)
             for letter in alpha_list:
                 print(letter)
```

```
1
a
b
c
2
a
b
c
3
a
b
c
```

```
In [46]: list_of_lists = [['apple', 'orange', 'grape'],[0, 1, 2],[9.9, 8.8, 7.7]]

         for list in list_of_lists:
             for item in list:
                 print(item)
```

```
apple
orange
grape
0
1
2
9.9
8.8
7.7
```

```
In [47]: a = np.arange(0, 100, 5).reshape(5,4)
         print(a)
```

```
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]
 [60 65 70 75]
 [80 85 90 95]]
```

```
In [48]: a[2,3]
```

```
Out[48]: 55
```

```
In [49]: a[0:5, 1] # each row in the second column
```

```
Out[49]: array([ 5, 25, 45, 65, 85])

In [50]: a[:,1] #each row in the second column

Out[50]: array([ 5, 25, 45, 65, 85])

In [51]: a[-1:] # last row

Out[51]: array([[80, 85, 90, 95]])

In [52]: a[-1,] # last row

Out[52]: array([80, 85, 90, 95])

In [53]: a[-1] # last row

Out[53]: array([80, 85, 90, 95])

In [54]: a[:,-1] #last column

Out[54]: array([15, 35, 55, 75, 95])
```

1.1.1 Array Reshaping

```
In [55]: a = np.floor(10*np.random.random([3,4]))
         a

Out[55]: array([[ 3.,  1.,  8.,  5.],
                [ 7.,  9.,  0.,  0.],
                [ 9.,  8.,  0.,  8.]])

In [56]: a.shape

Out[56]: (3, 4)

In [57]: a.ravel() # flattens the array.. the original is not changed

Out[57]: array([ 3.,  1.,  8.,  5.,  7.,  9.,  0.,  0.,  9.,  8.,  0.,  8.])

In [58]: a.reshape(6,2)

Out[58]: array([[ 3.,  1.],
                [ 8.,  5.],
                [ 7.,  9.],
                [ 0.,  0.],
                [ 9.,  8.],
                [ 0.,  8.]])

In [59]: a.T #transposed
```

```
Out[59]: array([[ 3.,  7.,  9.],
                [ 1.,  9.,  8.],
                [ 8.,  0.,  0.],
                [ 5.,  0.,  8.]])
```

```
In [60]: a.T.shape
```

```
Out[60]: (4, 3)
```

```
In [61]: a.shape
```

```
Out[61]: (3, 4)
```

1.1.2 Stacking Arrays

```
In [62]: a = np.arange(4).reshape(2,2)
         print(a)
```

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

```
In [63]: b = np.arange(4,8).reshape(2,2)
         print(b)
```

```
[[4 5]
 [6 7]]
```

```
In [64]: np.vstack([a,b])
```

```
Out[64]: array([[0, 1],
                [2, 3],
                [4, 5],
                [6, 7]])
```

```
In [65]: np.hstack([a,b])
```

```
Out[65]: array([[0, 1, 4, 5],
                [2, 3, 6, 7]])
```

1.1.3 Fancy Indexing

```
In [66]: a = np.arange(12)**2 # the first 12 square numbers
         i = np.array([2,2,3,8,5]) #the array of indicies
```

```
In [67]: print(a)
```

```
[  0   1   4   9  16  25  36  49  64  81 100 121]
```

```
In [68]: a[i]

Out[68]: array([ 4,  4,  9, 64, 25])

In [69]: j = np.array([3,4,9,7]).reshape(2,2)
         a[j]

Out[69]: array([[ 9, 16],
                [81, 49]])

In [70]: # update elements at indicies 3,5,6 to -1, -5 and -7 respectively
         a[[3,5,6]] = [-1,-5,-7]
         print(a)

[  0   1   4  -1  16  -5  -7  49  64  81 100 121]
```

1.1.4 Boolean Indexing

```
In [71]: a >0

Out[71]: array([False,  True,  True, False,  True, False, False,  True,  True,
                 True,  True,  True], dtype=bool)

In [72]: b = a[a>0]
         print(b)

[  1   4  16  49  64  81 100 121]
```