Exercise3 NumPy

April 15, 2018

1 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Library documentation: http://www.numpy.org/

```
In [2]: from numpy import *
        import numpy as np
```

2 Task 1: declare a vector using a list as the argument

```
In [3]: #TASK 1
        vec1 = np.array([1,5,2])
        vec2 = np.array([7,4,8])
        print type(vec1)
```

```
<type 'numpy.ndarray'>
```

3 Task 2: declare a matrix using a nested list as the argument

```
In [180]: #TASK 2
          matr = np.array([[1,0,2],
                           [2,5,3],
                           [7,8,6]])
          print matr
```

```
[[1 0 2]
 [2 5 3]
 [7 8 6]]
```

4 Task 3: initialize x or x and y using the following functions: arange, linspace, logspace, mgrid

```
In [42]: #arange
         x = np.arange(3,7,2)
         print type(x)
         print "X = ", x

         #linespace
         y = np.linspace(1.0, 2.0, num=5)
         print "Y = ", y

         #logspace
         x1 = np.logspace(1, 2, 5, endpoint=True)
         print "Log scale, X1:", x1
         x2 = np.logspace(1, 2, 5, endpoint=False)
         print "Log scale, X2:", x2

         #mgrid
         M_ = np.mgrid[-1:4, -1:4]
         print "Grid:", M_
```

```
<type 'numpy.ndarray'>
X =  [3 5]
Y =  [1.   1.25 1.5  1.75 2.  ]
Log scale, X1: [ 10.          17.7827941   31.6227766   56.23413252 100.        ]
Log scale, X2: [10.         15.84893192 25.11886432 39.81071706 63.09573445]
Grid: [[[-1 -1 -1 -1 -1]
  [ 0  0  0  0  0]
  [ 1  1  1  1  1]
  [ 2  2  2  2  2]
  [ 3  3  3  3  3]]

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

```
In [36]: from numpy import random
```

5 Task 4: what is difference between random.rand and random.randn

The main difference between "numpy.random.rand" and "numpy.random.randn" is,

numpy.random.rand$(d_0, d_1....d_n)$ creates an array of the given shape and populates it with random samples from a uniform distribution over [0, 1).

numpy.random.randn$(d_0, d_1....d_n)$ creates an array of the given shape and populates with samples from standard normal distribution.

citation: 1) https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randn.html

2) https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.rand.html

6 Task 5: what are the functions diag, itemsize, nbytes and ndim about?

1. *numpy.diag* : This function returns array of diagnol elements from a matrix.
2. *numpy.ndarray.itemsize* : This function returns the size of the numpy array.
3. *numpy.ndarray.nbytes* : The function returns total bytes consumed by the array object.
4. *numpy.ndarray.ndim* : The function returns number of array dimensions.

citation: https://docs.scipy.org/doc/numpy-1.14.0/reference/index.html

In [41]:
```
#diag
print np.diag(matrix)

#itemsize
print x.itemsize

#nbytes
print y.nbytes

#ndim
print matrix.ndim
```

```
[1 5 6]
8
40
2
```

In [171]:
```
# assign new value
M[0,0] = 7
```

In [181]:
```
M[0,:] = 0
```

In [173]:
```
# slicing works just like with lists
A = array([1,2,3,4,5])
A[1:3]
```

Out[173]:
```
array([2, 3])
```

7 Task 6: Using list comprehensions create the following matrix

array([[0, 1, 2, 3, 4], [10, 11, 12, 13, 14], [20, 21, 22, 23, 24], [30, 31, 32, 33, 34], [40, 41, 42, 43, 44]])

```
In [195]: indx = array([0,1,2,3,4])
          s = array([q for r in range(6) for q in range(r+10, 50)])
          s1 = array([q for r in range(5) for q in range(s[r+10], 50)])
          s2 = array([q for r in range(5) for q in range(s1[r+10], 50)])
          s3 = array([q for r in range(5) for q in range(s2[r+10], 50)])
          A = array([indx, s[indx], s1[indx], s2[indx], s3[indx]])
          print type(A)
          print A
```

```
<type 'numpy.ndarray'>
[[ 0  1  2  3  4]
 [10 11 12 13 14]
 [20 21 22 23 24]
 [30 31 32 33 34]
 [40 41 42 43 44]]
```

```
In [163]: # index masking
          B = array([n for n in range(5)])
          row_mask = array([True, False, True, False, False])
          B[row_mask]
```

```
Out[163]: array([0, 2])
```

7.0.1 Linear Algebra

```
In [198]: v1 = arange(0, 5)

In [199]: v1 + 2

Out[199]: array([2, 3, 4, 5, 6])

In [200]: v1 * 2

Out[200]: array([0, 2, 4, 6, 8])

In [201]: v1 * v1

Out[201]: array([ 0,  1,  4,  9, 16])

In [202]: dot(v1, v1)

Out[202]: 30

In [203]: dot(A, v1)

Out[203]: array([ 30, 130, 230, 330, 430])
```

```
In [204]: # cast changes behavior of + - * etc. to use matrix algebra
          M = np.matrix(A)
          M * M
```

```
Out[204]: matrix([[ 300,  310,  320,  330,  340],
                  [1300, 1360, 1420, 1480, 1540],
                  [2300, 2410, 2520, 2630, 2740],
                  [3300, 3460, 3620, 3780, 3940],
                  [4300, 4510, 4720, 4930, 5140]])
```

```
In [205]: # inner product
          v1.T * v1
```

```
Out[205]: array([ 0,  1,  4,  9, 16])
```

```
In [208]: C = np.matrix([[1j, 2j], [3j, 4j]])
          print C
```

```
[[0.+1.j 0.+2.j]
 [0.+3.j 0.+4.j]]
```

```
In [209]: conjugate(C)
```

```
Out[209]: matrix([[0.-1.j, 0.-2.j],
                  [0.-3.j, 0.-4.j]])
```

```
In [210]: # inverse
          C.I
```

```
Out[210]: matrix([[0.+2.j , 0.-1.j ],
                  [0.-1.5j, 0.+0.5j]])
```

7.0.2 Statistics

```
In [211]: mean(A[:,3])
```

```
Out[211]: 23.0
```

```
In [212]: std(A[:,3]), var(A[:,3])
```

```
Out[212]: (14.142135623730951, 200.0)
```

```
In [213]: A[:,3].min(), A[:,3].max()
```

```
Out[213]: (3, 43)
```

```
In [214]: d = arange(1, 10)
          sum(d), prod(d)
```

```
Out[214]: (45, 362880)
```

```
In [215]: cumsum(d)

Out[215]: array([ 1,  3,  6, 10, 15, 21, 28, 36, 45])

In [216]: cumprod(d)

Out[216]: array([     1,      2,      6,     24,    120,    720,   5040,  40320,
                 362880])

In [217]: # sum of diagonal
          trace(A)

Out[217]: 110

In [218]: m = random.rand(3, 3)

In [219]: # use axis parameter to specify how function behaves
          m.max(), m.max(axis=0)

Out[219]: (0.7516638099866687, array([0.51815456, 0.46494903, 0.75166381]))

In [220]: # reshape without copying underlying data
          n, m = A.shape
          B = A.reshape((1,n*m))

In [221]: # modify the array
          B[0,0:5] = 5

In [222]: # also changed
          A

Out[222]: array([[ 5,  5,  5,  5,  5],
                 [10, 11, 12, 13, 14],
                 [20, 21, 22, 23, 24],
                 [30, 31, 32, 33, 34],
                 [40, 41, 42, 43, 44]])

In [231]: # creates a copy
          B = A.flatten()
          print B

[ 5  5  5  5  5 10 11 12 13 14 20 21 22 23 24 30 31 32 33 34 40 41 42 43
 44]


In [224]: # can insert a dimension in an array
          v = array([1,2,3])
          v[:, newaxis], v[:,newaxis].shape, v[newaxis,:].shape

Out[224]: (array([[1],
                  [2],
                  [3]]), (3, 1), (1, 3))
```

```
In [225]: repeat(v, 3)

Out[225]: array([1, 1, 1, 2, 2, 2, 3, 3, 3])

In [226]: tile(v, 3)

Out[226]: array([1, 2, 3, 1, 2, 3, 1, 2, 3])

In [232]: w = array([5, 6])
          print w

[5 6]


In [228]: concatenate((v, w), axis=0)

Out[228]: array([1, 2, 3, 5, 6])

In [230]: # deep copy
          B = copy(A)
          print B

[[ 5  5  5  5  5]
 [10 11 12 13 14]
 [20 21 22 23 24]
 [30 31 32 33 34]
 [40 41 42 43 44]]
```