Usage of Numpy Data Structure

Import numpy as np

1. Array creation

A = np.array ([2,5,10])

A.dtype

Will show, int64 data type

B=np.array ([2.4,10.6,5.2])

B.dtype

Will show, float64 data type

Creating sequence of sequence will create 2-dimensional array.

A=np.array([(3,4,5),(12,6,1)])

Z=np.zeros((2,4)) # will create zero matrix of dimension 2x4

Similarly, np.ones((3,3)) # will create one's matrix of dimension 3x3

To create a sequence of data,

S=np.arange(10,30,5)

Print (S) will give (10,15,20,25,30), with step size of 5

np.arange(0, 2, 0.3) # it accepts float arguments

array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])

#Instead of step-size, we can specify total number of elements in the array using

S1=np.linspace(0,2,9) # produce 9 numbers starting 0 & ends with 2

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

#usage of Random Number functions

import random

random.choice([1,2,3,4,5]), this will pick one number from the list randomly

random.choice('python'), will pick one character from the string randomly

random.randrange(25,50), will pick one integer between 25 to 50

random.randrange(25,50,2), will pick one integer between 25 to 50 with step size of 2

random.random(), will pick a random number between 0 to 1

random.uniform(5,10), will pick a floating point number between 5 to 10

random.shuffle([1,2,3,4,5]), will shuffle the list elements

random.seed(10), to get same random value during every execution

2-Dimensional array (Matrix)

a = np.arange(15).reshape(3, 5)

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

#to check the dimension

a.shape
(3,5)
a.size # will return total elements in matrix (here 15)

to transpose a matrix

a.T # transposed to 5x3 matrix

3-Dimensional array

c = np.arange(24).reshape(2,3,4) # 1st value indicates (no of planes) (3,4) is the dimension

```
print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

c.shape will return (2,3,4)
c[1,...] is equal to c[1,:,:] # will fetch all elements of 2nd plane
c[...,2] is equal to c[:,:,2] [[3,7,11],[15,19,23]]

Array operations

```
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
b
array([0, 1, 2, 3])
c = a-b
c
array([20, 29, 38, 47])
b**2
array([0, 1, 4, 9])
10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
a<35
array([ True, True, False, False], dtype=bool)
```

Matrix operations

```
A = np.array( [[1,1],[0,1]] )
B = np.array( [[2,0],[3,4]] )
A*B                   # elementwise product
array([[2, 0],
      [0, 4]])
A.dot(B)              # matrix product
array([[5, 4],
      [3, 4]])
```

(OR)

```
np.dot(A, B)            # another matrix product
array([[5, 4],
      [3, 4]])
```

```
b = np.arange(12).reshape(3,4)
b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

b.sum(axis=0)                    # sum of each column
array([12, 15, 18, 21])

b.sum(axis=1)                    # sum of each row
array([6, 22, 38])
```

Indexing, Slicing & Iterating Array

```
a = np.arange(10)**3
a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])

a[2:5]
array([ 8, 27, 64])

a[0:6:2]

array([0,8,64,216])
```

Let '**b**', is an input matrix of size 5x4

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

b[2,3] #will fetch 23

b[0:5,1] or b[:5,1]  or b[:,1] #will fetch [1,11,21,31,41]

b[-1,:] # will fetch last row
b[:,-1] # will fetch last col


for row in b:
    print (row)  # will print every row
```

```
for element in b.flat:
   print (element) # will show all elements of b in 1-D array
```

Changing the shape of a matrix

```
b.ravel() # returns the array flattened to (1x 20)
```
Later, we can convert 5x4 matrix into 4x 5 matrix using

```
B1=b.reshape(4,5)
```

Stacking together different arrays

```
A1=np.array([(3,4,5),(12,6,1)])

3   4  5
12  6  1

A2=np.array([(1,2,6),(-4,3,8)])

 1   2  6
-4   3  8

D1=np.vstack((A1,A2))

3   4   5
12  6  1
1   2  6
-4   3  8

D2=np.hstack((A1,A2))

3   4   5  1  2  6
12  6  1 -4  3  8
```

Stacking 1-D array into 2-D array (column wise)

```
a = np.array([4.,2.])
b = np.array([3.,8.])

np.column_stack((a,b))     # returns a 2D array

array([[ 4., 3.],
       [ 2., 8.]])
```

```
np.hstack((a,b))          # the result is different
array([ 4., 2., 3., 8.])

np.hstack((a[:,newaxis],b[:,newaxis]))   # the result is the same
array([[ 4.,  3.],
       [ 2.,  8.]])
```

Indexing with array of indices

```
a = np.arange(12)**2                  # the first 12 square numbers
i = np.array( [ 1,1,3,8,5 ] )         # an array of indices
a[i]                                  # the elements of a at the positions i
array([ 1,  1,  9, 64, 25])

j = np.array( [ [ 3, 4], [ 9, 7 ] ] )     # a bidimensional array of indices
a[j]                             # the same shape as j

array([[ 9, 16],
       [81, 49]])
```

Usage of for-loop (Mapping by Value)

Calculate sum of all the elements in a 2D Numpy Array (iterate over elements)

```
import numpy as np
a=np.array([(3,2,9),(1,6,7)])

s1=0
for row in a:
   for col in row:
      s1+=col
print(s1)
```

Usage of for-loop (Mapping by Index)

Calculate sum of all the elements in a 2D Numpy Array (iterate over range)

```
import numpy as np
a=np.array([(3,2,9),(1,6,7)])
s=0
for i in range(a.shape[0]):
   for j in range(a.shape[1]):
      s+=a[i,j]
print(s)
```
