**Numpy – My Notes**

* In Numpy an array is call as ‘array’ whereas in Python it is called as ‘List’
* In Numpy entire array elements can be accessible through fixed off-set from the beginning without performing looping operation.
* In Numpy array can be created in different ways, as following:
  + From Python List:
    - A1 = np.array( [1, 2, 3, 4, 5] )
  + A single item Python list to initialize an array of 10 times:
    - A1 = np.array( [0] \* 10 )
  + From Python ‘range’ function:
    - A1 = np.array( range( 10)
  + By using Numpy zeros():
    - A1 = np.zeros(10)
    - A1 = np.zeros(10, dtype = int)
  + By using Numpy ‘arange()’ function:
    - A1 = np.arange(0, 10) – endpoint – 1
    - A1 = np.arange(0, 10, 2) – where ‘2’ is the step
    - A1 = np.arange(10, 0, -1) – count down
  + By using Numpy ‘linspace()’ function
    - A1 = np.linspace( firstnumber, secondnumber, no. of elements in array)
    - A1 = np.linspace( 0, 5, 6)
  + Mathematical operations on Numpy Arrays called **‘vectorization’** i.e., scope is entire array:
    - * A1 = np.arange(0, 5)
      * A1 \* 2 - array( [0, 2, 4, 6, 8] )
    - Adding two arrays of same length:
      * A1 = np.arange(0, 5)
      * A2 = np.arange(5, 10)
      * A1 + A2
      * Array( [ 5, 7, 9, 11, 13] )
    - Subract two arrays of same length:
      * A1 = np.arange(0, 5)
      * A2 = np.arange(5, 10)
      * A1 - A2
    - Multipy two arrays of same length:
      * A1 = np.arange(0, 5)
      * A2 = np.arange(5, 10)
      * A1 \* A2
    - Divide two arrays of same length:
      * A1 = np.arange(0, 5)
      * A2 = np.arange(5, 10)
      * A1 / A2
    - Remainder two arrays of same length:
      * A1 = np.arange(0, 5)
      * A2 = np.arange(5, 10)
      * A1 % A2
  + **Type of Array**
    - A1 = np.arange(0, 5)
    - Type(a1) - numpy.ndarray
  + **How many elements**
    - A1 = np.arange(0, 5)
    - np.size(a1) - 5
  + **Data type of elements**
    - A1 = np.arange(0, 5)
    - A1.dtype - int32
    - A1 = np.arange(0, 5.0)
    - A1.dtype - float64
  + **Two-Dimensional Numpy Array**
    - np.array ( [ [1, 2 ], [3, 4 ] )
    - output => array( [ [1, 2], [3, 4] ] )
  + **Creating Two-Dimensional Numpy Array with ‘reshape()’ function**
    - M = np.arange(0, 20).reshape(5, 4) - reshape(no of rows, no of columns)
    - Np.size(m) – 20
    - Np.size(m, 0) – 5 => finding no. of rows
    - Np.size(m, 1) – 4 => finding no. of columns
  + **Accessig One-Dimensional Numpy Array Element**
    - A1 = np.arange(0, 5)
    - A1[0] => 0
    - A1[2] => 2
  + **Accessig Two-Dimensional Numpy Array Element**
    - A1 = np.arange(0, 9).reshape(3, 3)
    - A1[0] => throw entire first row i.e., array[0, 1, 2]
    - A1[0, 0] => accessing element at 0, 0
    - A1[0, 2] => accessing element at row 0 and element 3
    - A1[1, 1] => accessing element at row 2 and element 2
    - A1[1, ] => entire row
    - A1[ : , 0] => gives entire first column
  + **Logical Operations on Two-Dimensional Array**
    - a1 = np.arange(5)
    - a1 <= 2 => array( [true, true, true, false, false])
    - a1 = np.arange(5) (a1 < 2) | (a1==2) **=>** array([ True, True, True, False, False])
  + **Counting Elements Based on Logical Operation**
    - Np.sum(a < 3) where a = array
  + **Comparing One-Dimensional Arrays**
    - a1 = np.arange(0, 5)
    - a2 = np.arange(5, 0, -1)
    - print(a1)
    - print(a2)
    - a1 < a2 => [] – array ( [true, true, true, false, false] )
  + **Comparing Two-Dimensional Arrays**
    - a1 = np.arange(9).reshape(3, 3)
    - a2 = np.arange(9, 0, -1).reshape(3, 3)
    - print(a1)
    - print(a2)
    - print(a1 < a2)
      * [[0 1 2]
      * [3 4 5]
      * [6 7 8]]
      * [[9 8 7]
      * [6 5 4]
      * [3 2 1]]
      * [[ True True True]
      * [ True True False]
      * [False False False]]

**SLICING ARRAY**

* a1 = np.arange(1, 10)
* a1[3:8] => array([4, 5, 6, 7, 8])

**Slicing with step value:**

* a1 = np.arange(1, 20)
* a1[ : : 2]

**Revesring the Array:**

* a1 = np.arange(1, 20)
* a1[ : : -1]

**Slicing Two-Dimensional Array:**

* A0 = np.arange(9).reshape(3, 3)
* print(A0)
* A0[1:2, 1:2 ] - in order to pick **‘4’** in the matrix of 3x3

[[0 1 2]

[3 4 5]

[6 7 8]]

array([[4]])

* A0 = np.arange(9).reshape(3, 3)
* print(A0)
* A0[2:3, 2:3] - in order to pick ‘8’ in the above matrix
* A0 = np.arange(9).reshape(3, 3)
* print(A0)
* A0[0:2, 1:3] - in order to pick [ [1,2],

[4, 5] ]

* A0 = np.arange(9).reshape(3, 3)
* print(A0)
* A0[2:2+1, 1:1+1] => in order to pick ‘7’ from the above matrix. Note the ‘+’ sign.

**Reshaping Array**

* Changing of array view from one-dimension to two-dimension or vice versa is called ‘reshaping’ of array.
* Reshaping can be performed from ‘.reshape()’, ‘.ravel()’, ‘.flatten()’ methods
* In reshape() method no. of elements should be passed as argument e.g., a.reshape(9) or a.reshape(np.size(a))
* Whereas in .ravel() method no argument should be passed, else it will generate error
* Both reshape and ravel methods creates a View over the original array, however, if we change any element of that View, the original gets changed as well.
* Meaning both reshape and ravel methods follows ‘pass by reference’ NOT ‘pass by value’
* Ravel and Flatten methods are same BUT flatten follows ‘pass by value’ meaning it copies the original array and therefore, changes in the copied array does not impact on the original one.
* .shape property returns a tuple representing the shape of the array.
* .transpose() method, simply changes the rows into columns e.g., m.transpose()
* Transpose can also be achieved through ‘.T’ property e.g., m.T
* .resize() method is similar to the .reshape() method, except while reshaping it returns a new array with data copied into it. e.g., m.resize(1, 9)

**Combining Arrays**

* Process of combining arrays is called staking. Stacking can take various forms including Horizontal, Vertical and Depth-wise stacking. Let’s suppose we have two arrays a & b as following:

**Horizontal Stacking**

a = np.arange(0, 9).reshape(3, 3)

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

b = (a + 1) \* 10

array([[10, 20, 30],

[40, 50, 60],

[70, 80, 90]])

np.hstack((a, b))

array([[ 0, 1, 2, 10, 20, 30],

[ 3, 4, 5, 40, 50, 60],

[ 6, 7, 8, 70, 80, 90]])

* .hstack() is same as np.concatenate() function with **axis = 1**

np.concatenate((a, b), axis = 1)

array([[ 0, 1, 2, 10, 20, 30],

[ 3, 4, 5, 40, 50, 60],

[ 6, 7, 8, 70, 80, 90]])

**Vertical Stacking**

np.vstack((a, b))

array([[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8],

[10, 20, 30],

[40, 50, 60],

[70, 80, 90]])

* Vertical stacking is same as **np.concatenate((a, b), axis = 0)**

**Depth-wise Stacking**

We have two arrays ‘a’ and ‘b’,

where ‘a’ equals:

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

and ‘b’ equals:

array([[10, 20, 30],

[40, 50, 60],

[70, 80, 90]])

np.dstack((a, b))

array([[[ 0, 10],

[ 1, 20],

[ 2, 30]],

[[ 3, 40],

[ 4, 50],

[ 5, 60]],

[[ 6, 70],

[ 7, 80],

[ 8, 90]]])

**Column Stack**

* + Column stacking performs a horizontal stack of two one-dimensional arrays, making each array a column in the resulting array:

one\_d\_a = np.arange(5)

one\_d\_a

array([0, 1, 2, 3, 4])

one\_d\_b = (one\_d\_a + 1) \* 10

one\_d\_b

array([10, 20, 30, 40, 50])

np.column\_stack((one\_d\_a, one\_d\_b))

array([[ 0, 10],

[ 1, 20],

[ 2, 30],

[ 3, 40],

[ 4, 50]])

**Row Stack**

np.row\_stack((one\_d\_a, one\_d\_b))

array([[ 0, 1, 2, 3, 4],

[10, 20, 30, 40, 50]])

**Splitting Arrays Do it again**

Arrays can also be split into multiple arrays along the horizontal, vertical and depth axes using the np.hsplit(), np.vsplit() and np.dsplit() functions.

**Horizontal Split (Infact it splitting Columns)**

m = np.arange(12).reshape(3, 4)

m

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

np.hsplit(m, 4)

[array([[0],

[4],

[8]]),

array([[1],

[5],

[9]]),

array([[ 2],

[ 6],

[10]]),

array([[ 3],

[ 7],

[11]])]

np.hsplit(m, 2)

[array([[0, 1],

[4, 5],

[8, 9]]),

array([[ 2, 3],

[ 6, 7],

[10, 11]])]

**Splitting column 1 & 3**

m

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

**np.hsplit(m,[1, 3])**

[array([[0],

[4],

[8]]),

array([[ 1, 2],

[ 5, 6],

[ 9, 10]]),

array([[ 3],

[ 7],

[11]])]

**np.split(m, 2, axis = 1)**

[array([[0, 1],

[4, 5],

[8, 9]]),

array([[ 2, 3],

[ 6, 7],

[10, 11]])]

**Vertical Split (Infact it Splitting Rows)**

**np.vsplit(m,[1, 3])**

[array([[0, 1, 2, 3]]), array([[ 4, 5, 6, 7],

[ 8, 9, 10, 11]]), array([], shape=(0, 4), dtype=int32)]

**np.vsplit(m, 3)**

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

**Depth Split**

**Splitting – Do it again**

**Useful Numerical Methods of Numpy Arrays**

c = np.arange(10, 19).reshape(3, 3)

print(c)

[[10 11 12]

[13 14 15]

[16 17 18]]

print(c)

print("{0} min of the entire matrix".format(c.min()))

print("{0} max of the entire matrix".format(c.max()))

print("{0} position of the minimum value".format(c.argmin()))

print("{0} position of the maximum value".format(c.argmax()))

print("{0} mins down each column".format(c.min(axis=0)))

print("{0} mins across each row".format(c.min(axis=1)))

print("{0} max down each column".format(c.max(axis=0)))

print("{0} max across each row".format(c.max(axis=1)))

[[10 11 12]

[13 14 15]

[16 17 18]]

10 min of the entire matrix

18 max of the entire matrix

0 position of the minimum value

8 position of the maximum value

[10 11 12] mins down each column

[10 13 16] mins across each row

[16 17 18] max down each column

[12 15 18] max across each row

**Mean, Std. Deviation, Var**

a = np.arange(10)

print(a)

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

a.mean(), a.std(), a.var()

(4.5, 2.8722813232690143, 8.25)

**Sum, Cummulative Sum, Product, Cummulative Product**

b = np.arange(1, 6)

b

array([1, 2, 3, 4, 5])

print(b.sum(), b.prod())

15 120

print(b.cumsum(), b.cumprod())

[ 1 3 6 10 15] [ 1 2 6 24 120]

**Any() and All()**

d = np.arange(2)

(d > 5).any()

False

(d > 5).all()

False

**Size() and ndim()**

e = np.arange(12).reshape(3, 4)

print(e.size)

12

print(e.ndim)

2