Numpy

The main objective of the numpy module is to handle and create single and multi-dimensional array.

Creating an Array using numpy

In [3]:

```
import numpy as np
# for creating an array we simply use syntax (array_name = np.array([initial
array_1 = np.array([2,4,6,8,10]) #This is a single dimensional array
print (array_1.ndim)
array_1

```

```
1
```

Out[3]:

```
array([ 2,  4,  6,  8, 10])
```

In [4]:

```
# Basically we deal with 2 dimensional data having series of values belongin
#In order to create 2-dimensional array:
array_2 = np.array([[1,2,3],[4,5,6]])
print (array_2.ndim)
array_2
#This is a 2-dimensional array having 2 rows and 3 columns
```

```
2
```

Out[4]:

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

In [5]:

```
#Similarly we can also create a 3-dimensional array having n layers stacked
array_3 = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]],[[13,14,15],[16,1
print (array_3.ndim)
array_3
#This is a 3-Dimensional array having 3 layers stacked over one another, and
```

```
3
```

Out[5]:

```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]],

       [[13, 14, 15],
        [16, 17, 18]]])
```

To Create an array completely filled with zeros or ones or to create an identity matrix:

In [9]:

```
np.zeros(5)
```

Out[9]:

```
array([0., 0., 0., 0., 0.])
```

In [17]:

```
a = np.zeros((3,3)) #creates an array of zeros completely filled with zeros
a
```

Out[17]:

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [20]:

```
#Similarly we can create an array completely filled with ones
y = np.ones(5)
print (y)
z = np.ones((3,3))
print (z)
```

```
[1. 1. 1. 1. 1.]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

In [24]:

```
#To create an identity matrix:
Identity_ = np.eye(3)
Identity_
```

Out[24]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [26]:

```
#Linspace gives values which are equally spaced between the start, stop
#Its syntax is np.linspace(start, stop, no. of equally spaced values require
linsp_ = np.linspace(0,5,10)
linsp_
```

Out[26]:

```
array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
       2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])
```

In [28]:

```
#arang_ gives values between start and stop at defined step
#its syntax is as follows np.arange(start,stop,step)
arang_ = np.arange(0,10,2)
arang_
```

Out[28]:

```
array([0, 2, 4, 6, 8])
```

In [30]:

```
#With np.full we can get a matrix completely filled with the given value
#its syntax is np.full(size,element)
ful_ = np.full((3,3),8)
ful_
```

Out[30]:

```
array([[8, 8, 8],
       [8, 8, 8],
       [8, 8, 8]])
```

In [31]:

```
#To generate an array of random values between a given range and of defined
#its syntax is np.random.randint(range of random values,size)
np.random.randint(0,10,(4,4))
```

Out[31]:

```
array([[9, 9, 7, 9],
       [0, 9, 0, 2],
       [6, 5, 2, 3],
       [4, 3, 0, 9]])
```

Accessing elements in array

In [8]:

```
#Above we have created array_1 (1-Dimensional), array_2 (2-dimensional), arr
#In order to access elements in 1-D array
print (array_1[0])
#In order to access elements in 2-D array , you have to pass both the rows a
print (array_2[1][2])
#In order to access elements in 3-D array , you have to pass the layer, row
print (array_3[1][1][2])
```

```
2
6
12
```

Inspecting an array

In [40]:

```
#Replacing a value in array:
#Lets create a 2-D array
arr_ = np.eye(4,5) #gives an identity matrix
arr_[2][1]= np.nan #replaces a value by nan
arr_
```

Out[40]:

```
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0., nan,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.]])
```

In [41]:

```
arr_.shape #use to check the shape of an array
```

Out[41]:

```
(4, 5)
```

In [42]:

```
arr_.size #use to check the size of an array, number of elements it holds
```

Out[42]:

```
20
```

In [46]:

```
arr_.ndim #provides the dimension of the array
```

Out[46]:

```
2
```

In [47]:

```
arr_.dtype #provides the array data-type
```

Out[47]:

```
dtype('float64')
```

In [48]:

```
#converting array of 1 data-type to another (from float to int)
arr_.astype(np.int).dtype  #converts the array data-type from one to other
```

Out[48]:

```
dtype('int32')
```

Statistics of Numpy array:

In [52]:

```
#Lets create a random array:
z = np.random.randint(0,20,(4,4))
z
```

Out[52]:

```
array([[ 5, 10,  3, 12],
       [ 0, 12, 17, 19],
       [19, 13, 16, 16],
       [ 5, 10, 19, 15]])
```

In [60]:

```
print (np.sum(z))  #return sum of all elements of the array
print(np.sum(z,axis=0)) #axis = 0 represents along columns and axis = 1 repr
print (np.sum(z,axis=1))
print (np.mean(z)) #returns mean of all elements of array
print(np.median(z)) #returns median of all elements of array
print (np.cumsum(z)) # returns cumsum of all elements of array
print (np.std(z)) #returns the standard deviation
```

```
191
[29 45 55 62]
[30 48 64 49]
11.9375
12.5
[  5  15  18  30  30  42  59  78  97 110 126 142 147 157 176 191]
5.835974104637545
```

Mathematical operations on array

In [74]:

```
ar_ = np.array([1,2,3,4,5,6])
ar_1 = np.array([11,12,13,14,15,16])
```

In [62]:

```
np.sqrt(ar_) #provides square root of all elements of array
```

Out[62]:

```
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974])
```

In [64]:

```
np.log(ar_) #provides log e of all elements of array
```

Out[64]:

```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947])
```

In [69]:

```
np.log10(ar_) #provides log 10 of all elements of array
```

Out[69]:

```
array([0.        , 0.30103   , 0.47712125, 0.60205999, 0.69897   ,
       0.77815125])
```

In [75]:

```
np.add(ar_,ar_1) #adds two arrays
```

Out[75]:

```
array([12, 14, 16, 18, 20, 22])
```

In [76]:

```
np.subtract(ar_,ar_1) #subtract two arrays
```

Out[76]:

```
array([-10, -10, -10, -10, -10, -10])
```

In [80]:

```
np.multiply(ar_,ar_1) #multiplies each element of ar_ with every element of
```

Out[80]:

```
array([11, 24, 39, 56, 75, 96])
```

In [78]:

```
np.divide(ar_,ar_1) #divide two arrays
```

Out[78]:

```
array([0.09090909, 0.16666667, 0.23076923, 0.28571429, 0.33333333,
       0.375     ])
```

In [79]:

```
np.dot(ar_,ar_1) #the dot product is the actual matrix multiplication
```

Out[79]:

```
301
```

In [81]:

```
#for a dot product of 2-D array
#Number of columns of one matrix should be equal to number of rows of anothe
np.dot(ar_.reshape(2,3),ar_1.reshape(3,2))
```

Out[81]:

```
array([[ 82,  88],
       [199, 214]])
```

Array Manipulation:

In [98]:

```
#Slicing array / Subsetting
a = np.array([10,20,30,40,50,60,70,80])
print (a[0])
print (a[:3])
print (a[a>40])
```

```
10
[10 20 30]
[50 60 70 80]
```

In [99]:

```
np.insert(a, 0, 100) #insert an element in given array, at specified index
```

Out[99]:

```
array([100,  10,  20,  30,  40,  50,  60,  70,  80])
```

In [100]:

```
np.delete(a, [3,4]) #deletes the elements of the given indicess
```

Out[100]:

```
array([10, 20, 30, 60, 70, 80])
```

In [106]:

```
a.resize(2,4)
a #changes the orientation of the array of the given rows*columns
```

Out[106]:

```
array([[10, 20, 30, 40],
       [50, 60, 70, 80]])
```

In [103]:

```
a.reshape(2,4)
```

Out[103]:

```
array([[10, 20, 30, 40],
       [50, 60, 70, 80]])
```

In [104]:

```
a.T #return the transpose of the array, i.e, rows changed to columns and col
```

Out[104]:

```
array([[10, 50],
       [20, 60],
       [30, 70],
       [40, 80]])
```

In [108]:

```
print (a)
np.ravel(a) #flattens a multi-dimensional array.
```

```
[[10 20 30 40]
 [50 60 70 80]]
```

Out[108]:

```
array([10, 20, 30, 40, 50, 60, 70, 80])
```

On two arrays:

In [116]:

```
# Creating few arrays so that we can see the transformations:
x = np.arange(1,11,1).reshape(2,5)
y = np.arange(1,6,1).reshape(1,5)
p = np.arange(1,4,1).reshape(3,1)
print (x)
print (y)
print (p)
```

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

In [114]:

```
#Concatenate two arrays:
z = np.concatenate((x,y),axis=0) #axis = 0 means along column, it merges the
print (z)
```

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

In [120]:

```
new_= np.concatenate((z,p),axis=1) #axis = 1 means along rows, it merges the
new_
```

Out[120]:

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

In [122]:

```
np.vsplit(new_,3) #splits the arrays along vertical, into 3 pieces
```

Out[122]:

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

In [124]:

```
np.hsplit(new_,2) #splits the array along horizontal, into 2 pieces
```

Out[124]:

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

In [ ]:

```

```