Numpy

In [95]:
```
pip install numpy
```

```
Requirement already satisfied: numpy in d:\anacoda\lib\site-packages (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

In [96]:
```
import numpy as np
a= np.array ([1,2,2])
print (a)
b=np.zeros(3)
print (b)

c=np.ones(3)
print ("c",c)

d=np.empty(2)

print (d)
```

```
[1 2 2]
[0. 0. 0.]
c [1. 1. 1.]
[4.24399158e-314 1.18831764e-312]
```

1-D Array and 2-D array

In [97]:
```
#one dimentional ::
a= np.array ([1,2,3,4,5])
print("a:\n",a)
#two dimentional::
b = np.array([[1,2,3,4,5],[2,3,4,5,6],[2,4,6,8,0]])
print("b:\n",b)
print ("a = ",type(a),'\n',"b = ",type(b))
```

```
a:
 [1 2 3 4 5]
b:
 [[1 2 3 4 5]
 [2 3 4 5 6]
 [2 4 6 8 0]]
a =  <class 'numpy.ndarray'>
 b =  <class 'numpy.ndarray'>
```

How to Create an Array 1-D ?

In [98]:
```
# with range function
e=np.arange(6)
print ("e : \n",e)
```

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

In [99]:
```
# with range of specific element
f=np.arange(2,10)
print ("f : \n",f)
```

```
f :
 [2 3 4 5 6 7 8 9]
```

In [100…
```
# with range of specific element and gap of 2
g=np.arange(2,10,2)
print ("f : \n",g)
```

```
f :
 [2 4 6 8]
```

In [101…
```
# with linear spaced arrays:: it will give us 7 those elements that have same differenc
h=np.linspace(0,10,num=7)
print ("f : \n",h)
```

```
f :
 [ 0.          1.66666667  3.33333333  5.          6.66666667  8.33333333
 10.        ]
```

In [102…
```
# specific data types in array:: you can have specific type elements as below
i=np.ones(5,dtype=np.int8)
print ("i: \n",i,"\ntype :",type(i))
```

```
i:
 [1 1 1 1 1]
type : <class 'numpy.ndarray'>
```

In [103…
```
# specitic data type of float
j = np.ones (3,dtype=np.float64)
print ("j: \n",j,"\ntype :",type(j))
```

```
j:
 [1. 1. 1.]
type : <class 'numpy.ndarray'>
```

How to Create an Array 2-D ?

In [104…
```
# 2-D array of zeros.

k = np.zeros((3,4))

print ("k: \n",k,"\ntype :",type(k))
```

```
k:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
type : <class 'numpy.ndarray'>
```

In [105…
```
# 2-D array of ones.
l = np.ones((3,4))
```

```
print ("l: \n",l,"\ntype :",type(l))
```

```
l:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
type : <class 'numpy.ndarray'>
```

In [106…

```
# 2-D array of ones.
n = np.empty((3,4))

print ("n: \n",n,"\ntype :",type(n))
```

```
n:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
type : <class 'numpy.ndarray'>
```

How to Create an Array 3-D ?

- arange will give the range of the elements.
- reshape will tell you the shape of 3 D array.

In [107…

```
# making and reshaping 3D arrays
o=np.arange(24).reshape(2,3,4)
print ("o: \n",o,"\ntype :",type(o))
```

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

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
type : <class 'numpy.ndarray'>
```

In [108…

```
len (o)
```

Out[108… 2

Empty function is assingment

In [109…

```
## Its not empty matrix ::The function empty creates an array whose initial content is
pp= np.empty((3))
print (pp)
```

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

Array Functions

In [110…
```
# make an array

a= np.array ([2.3,.4,.5,.6,9,1])
a
```

Out[110…
```
array([2.3, 0.4, 0.5, 0.6, 9. , 1. ])
```

In [111…
```
# sort the elements..
a.sort()
a
```

Out[111…
```
array([0.4, 0.5, 0.6, 1. , 2.3, 9. ])
```

In [112…
```
# concatenation ::
b = np.array ([1,2,4,56])
c = np.concatenate((a,b))
c.sort ()
c
```

Out[112…
```
array([ 0.4,  0.5,  0.6,  1. ,  1. ,  2. ,  2.3,  4. ,  9. , 56. ])
```

2-D array

In [113…
```
x=np.array([[1,2],[4,6]])
x
```

Out[113…
```
array([[1, 2],
       [4, 6]])
```

In [114…
```
y=np.array([[4,8],[3,6]])
y
```

Out[114…
```
array([[4, 8],
       [3, 6]])
```

In [115…
```
# concatinate the array's
z=np.concatenate((x,y),axis=0)
z
```

Out[115…
```
array([[1, 2],
       [4, 6],
       [4, 8],
       [3, 6]])
```

In [116…
```
# concatinate the array's
z=np.concatenate((x,y),axis=1)
z
```

Out[116…
```
array([[1, 2, 4, 8],
       [4, 6, 3, 6]])
```

3-D array !!

In [117…
```
# make 2-D array inside and array you will get 3-D array


d=np.array([[[1,2,3,4],[4,3,2,1]],[[2,4,6,8],[8,6,4,2]],[[3,6,7,5],[1,2,4,2]]])

d
```

Out[117…
```
array([[[1, 2, 3, 4],
        [4, 3, 2, 1]],

       [[2, 4, 6, 8],
        [8, 6, 4, 2]],

       [[3, 6, 7, 5],
        [1, 2, 4, 2]]])
```

In [118…
```
# find the dimension of the array.


d.ndim
```

Out[118…
```
3
```

In [119…
```
# find the number of elements
d.size
```

Out[119…
```
24
```

In [120…
```
# find the shape 3 is no. of dimension 2 is rows 4 is columns

d.shape
```

Out[120…
```
(3, 2, 4)
```

In [121…
```
# Reshape the 1-D

a= np.arange(9)
a
```

Out[121…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [122…
```
# Reshape the 1-D to 2-D

a.reshape(3,3)
```

Out[122…
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [123…
```
# check the dimensions
a.ndim
```

Out[123…
```
1
```

In [124…
```
# Convert 1-D to 2-D Row wise
c= np.arange(9)
b =c[np.newaxis,:]
b.ndim
```

Out[124…
```
2
```

In [125…
```
#check the shape
b.shape
```

Out[125…
```
(1, 9)
```

In [126…
```
# column wise 2-D conversion
d= np.arange(9)
j=d[: ,np.newaxis]
j
```

Out[126…
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8]])
```

In [127…
```
# find the shape and dimension
print ("shape: ",j.shape,"\n")
print ("DImension : ",j.ndim)
```

```
shape:  (9, 1)

DImension :  2
```

In [128…
```
## sum

a=np.arange (10)

print ("array ::\n",a)
print ("dimension ::\n",a.ndim)
print  ("sum :" , a.sum())
```

```
array ::
 [0 1 2 3 4 5 6 7 8 9]
dimension ::
 1
sum : 45
```

In [129…
```
## Assigments of all function remaining
```

In [130…
```
# eye function :: the diagonal values are 1 n rest are 0
eyes=np.eye(6,6)
print(eyes)
```

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

In [131…
```
# random function :: the values are between 0 and 1 and the matrix is 4*2(4 rows and 2
rand=np.random.random((4,2))

print (rand)
```

```
[[0.71096683 0.3675308 ]
 [0.63608981 0.40354381]
 [0.62624268 0.40885925]
 [0.62481784 0.59837012]]
```

In [132…
```
# slicing
a= np.arange(1,10)

a[3:8]
```

Out[132…
```
array([4, 5, 6, 7, 8])
```

In [133…
```
# mx,min,mean

print("max ::",a.max(),"\n",
"min ::",a.min(),"\n"
"sum ::",a.sum(),"\n"
"max ::",a.mean())
```

```
max :: 9
 min :: 1
sum :: 45
max :: 5.0
```