Numpy

Numpy is a python library it stands for Numerical python

It provides a high-performance multidimensional array object, and tools for working with these arrays.

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.

In Numpy, number of dimensions of the array is called rank of the array.

A tuple of integers giving the size of the array along each dimension is known as shape of the array.

An array class in Numpy is called as ndarray.

Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.

In [3]:
```
pip install numpy
```
```
Requirement already satisfied: numpy in d:\archana\lib\site-packages (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

In [1]:
```
import numpy as np
```

In [5]:
```
a=np.array([1,2,3])
a
```
Out[5]:
```
array([1, 2, 3])
```

In [6]:
```
a=np.array([1,2,3,'a'])
a
```
Out[6]:
```
array(['1', '2', '3', 'a'], dtype='<U11')
```

In [7]:
```
b=np.array([[1,2,3,4],[1,2,3,4]])
b
```
Out[7]:
```
array([[1, 2, 3, 4],
       [1, 2, 3, 4]])
```

Dimensions in array

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array

In [10]:
```
a=np.array(10)
b=np.array([1,2,3])
c=np.array([[1,2,3],[4,5,6]])
d=np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])
print('a dimension:',a.ndim)
print('b dimension:',b.ndim)
print('c dimension:',c.ndim)
print('d dimension:',d.ndim)
```

```
a dimension: 0
b dimension: 1
c dimension: 2
d dimension: 3
```

shape of an array:

The shape of an array is the number of elements in each dimension.

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

In [12]:
```
a=np.array(10)
b=np.array([1,2,3])
c=np.array([[1,2,3],[4,5,6]])
d=np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])
print(a)
print('shape of a:',a.shape)
print(b)
print('shape of b:',b.shape)
print(c)
print('shape of c:',c.shape)
print(d)
print('shape of d:',d.shape)
```

```
10
shape of a: ()
[1 2 3]
shape of b: (3,)
[[1 2 3]
 [4 5 6]]
shape of c: (2, 3)
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
shape of d: (2, 2, 3)
```

Datatypes in NumPy

The NumPy array object has a property called dtype that returns the data type of the array

In [13]:
```
a=np.array(10)
b=np.array([1,2,3])
print(a.dtype)
print(b.dtype)
```

```
int32
int32
```

In [14]:
```
print(type(a))
print(type(b))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

In [15]:
```
#len of array
len(b)
```

Out[15]:
```
3
```

In [16]:
```
l1=[10,20,30]
l2=[40,50,60]
l3=np.array(l1)
l4=np.array(l2)
print(type(l3))
print(l4)
```

```
<class 'numpy.ndarray'>
[40 50 60]
```

Numpy range

This function returns an ndarray object containing evenly spaced values within a given range.

In [17]:
```
np.arange(1,11)
```

Out[17]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [18]:
```
np.arange(1,11,2)
```

Out[18]:
```
array([1, 3, 5, 7, 9])
```

eye()method

The numpy. eye() method returns an array of shape, R x C, where all elements are equal to zero, except for the kth diagonal, whose values are equal to one.

In [19]:
```
np.eye(10,dtype=int)
```

Out[19]:
```
array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
```

numpy.zeros():

The numpy.zeros() function returns a new array of given shape and type, with zeros.

In [3]:
```
np.zeros((3,5),dtype=int)
```

Out[3]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

In [5]:
```
np.zeros((3,5))# default it takes float
```

Out[5]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

numpy.ones()

The numpy.ones() function returns a new array of given shape and type, with ones.

In [6]:
```
np.ones((3,2))
```

Out[6]:
```
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

numpy.full()

The full() function return a new array of given shape and type, filled with fill_value

In [7]:
```
np.full((3,2),5)
```

Out[7]:
```
array([[5, 5],
       [5, 5],
       [5, 5]])
```

numpy.diag()

we are able to find a diagonal element from a given matrix and gives output as one dimensional matrix.

In [8]:
```
x=[1,2,3,4,5]
np.diag(x)
```

Out[8]:
```
array([[1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0],
       [0, 0, 0, 0, 5]])
```

Numpy Random Numbers

In [9]:
```
np.random.random(1)[0]
```

Out[9]:
```
0.7818367084326971
```

In [10]:
```
np.random.randn(5)
```

Out[10]:
```
array([-0.46529875, -0.2613434 , -0.55276975,  0.95178709, -0.71302441])
```

In [11]:
```
import random #it generates random values every time
a=random.randint(1,100)
a
```

Out[11]:
```
53
```

Numpy Reshape

Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

In [12]:
```
#using arange() to generate numpy array x with numbers between 1 to 16
x=np.arange(1,17)
print(x)
print(x.shape)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
(16,)
```

In [15]:
```
#Reshape x with 2 rows and 8 columns
n=x.reshape((4,4))
print(n)
print(n.shape)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
(4, 4)
```

In [16]:
```
n=x.reshape((4,4),order='c')
print(n)
print(n.shape)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
(4, 4)
```

In [17]:
```
#convert to old dimension
v=n.ravel()
print(v.shape)
print(v)
```

```
(16,)
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
```

Numpy Array indexing

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

By using ndarray.flatten() function we can flatten a matrix to one dimension in python.

In [18]:
```
x.flatten()
```

```
Out[18]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])
```

```
In [19]: #create an array a with all even numbers between 1 to 20
         a=np.arange(2,20,2)
         a
```

```
Out[19]: array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

```
In [20]: #third element in array a
         a[2]
```

```
Out[20]: 6
```

```
In [21]: a[0:7:2]
```

```
Out[21]: array([ 2,  6, 10, 14])
```

Numpy Array Slicing

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step].

```
In [26]: a= np.array([1, 2, 3, 4, 5, 6, 7])

         a[1:5]
```

```
Out[26]: array([2, 3, 4, 5])
```

```
In [27]: a[4:]
```

```
Out[27]: array([5, 6, 7])
```

```
In [28]: a[:4]
```

```
Out[28]: array([1, 2, 3, 4])
```

NumPy Array Copy vs View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

```
In [7]: x1=np.arange(10)
```

```
x1
```

Out[7]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [8]:
```
x2=x1
print(x1)
print(x2)
```

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

In [9]:
```
#changing the first element of x2 as 10
x2[0]=10
print(x1)
print(x2)
```

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

In [10]:
```
print(id(x1))
```

```
2238159592208
```

In [12]:
```
print(np.shares_memory(x1,x2))
print(id(x1))
print(id(x2))
```

```
True
2238159592208
2238159592208
```

In [13]:
```
#create a view of x1 and store it in x3
x3=x1.view()
```

In [14]:
```
#again check memory shares between x1 and x3
np.shares_memory(x1,x3)
```

Out[14]:
```
True
```

In [15]:
```
#change 1st element of x3=100
x3[0]=100
```

In [16]:
```
# print x1  and x3 to check if changes reflected in both
print(x1)
print(x3)
```

```
[100   1   2   3   4   5   6   7   8   9]
[100   1   2   3   4   5   6   7   8   9]
```

In [17]:
```
#now create a array x4 which is copy of x1
x4=np.copy(x1)
```

In [18]:
```
#change last element of x4 as 900
x4[-1]=900
```

In [19]:
```
# print x1  and x4 to check if changes reflected in both
print(x1)
print(x4)
```
```
[100   1   2   3   4   5   6   7   8   9]
[100   1   2   3   4   5   6   7   8 900]
```

In [21]:
```
#check memory shares between x1 and x4
print(id(x1))
print(id(x4))
print(np.shares_memory(x1,x4))
```
```
2238159592208
2238159591536
False
```

more operation on numpy

1.Apply condition

In [24]:
```
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
a
```
Out[24]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [25]:
```
#print the elements grater than 3
a>3
```
Out[25]:
```
array([[False, False, False],
       [ True,  True,  True],
       [ True,  True,  True]])
```

In [26]:
```
a[a>3]
```
Out[26]:
```
array([4, 5, 6, 7, 8, 9])
```

In [27]:
```
a[(a>3)&(a<6)]
```
Out[27]:
```
array([4, 5])
```

In [28]:
```
x1==x4
```
Out[28]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
       False])
```

2.Transpose Array

In [29]:
```
#print transpose of array 'a'
print(a.transpose())
#print array 'a'
print('----------')
print(a)
```
```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
----------
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

3.hstack vs vstack function

Stacking is same as concatenation,the only difference is that stacking is done along with the given arrays

numpy has two arrays

hstack() vstack()

In [31]:
```
np.hstack((x1,x4))
```

Out[31]:
```
array([100,   1,   2,   3,   4,   5,   6,   7,   8,   9, 100,   1,   2,
         3,   4,   5,   6,   7,   8, 900])
```

In [32]:
```
print(x1)
print(x4)
np.vstack((x1,x4))
```

```
[100   1   2   3   4   5   6   7   8   9]
[100   1   2   3   4   5   6   7   8 900]
```

Out[32]:
```
array([[100,   1,   2,   3,   4,   5,   6,   7,   8,   9],
       [100,   1,   2,   3,   4,   5,   6,   7,   8, 900]])
```

Adding insert,delete operations

In [33]:
```
print(x1)
print(x4)
np.insert(x1,4,x4)
```

```
[100   1   2   3   4   5   6   7   8   9]
[100   1   2   3   4   5   6   7   8 900]
```

Out[33]:
```
array([100,   1,   2,   3, 100,   1,   2,   3,   4,   5,   6,   7,   8,
       900,   4,   5,   6,   7,   8,   9])
```

In [34]:
```
print(x2)
np.delete(x2,0)
```

```
[100   1   2   3   4   5   6   7   8   9]
```

Out[34]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Mathematical operations on numpy

In [35]:
```
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
np.sin(a)
```

Out[35]:
```
array([[ 0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ],
       [ 0.6569866 ,  0.98935825,  0.41211849]])
```

In [36]:
```
(np.sin(a))/(np.cos(a))
```

Out[36]:
```
array([[ 1.55740772, -2.18503986, -0.14254654],
       [ 1.15782128, -3.38051501, -0.29100619],
       [ 0.87144798, -6.79971146, -0.45231566]])
```

In [37]:
```
np.exp(a)
```
Out[37]:
```
array([[2.71828183e+00, 7.38905610e+00, 2.00855369e+01],
       [5.45981500e+01, 1.48413159e+02, 4.03428793e+02],
       [1.09663316e+03, 2.98095799e+03, 8.10308393e+03]])
```

In [38]:
```
np.sum(a)
```
Out[38]:
```
45
```

In [39]:
```
np.sum(a,axis=1)
```
Out[39]:
```
array([ 6, 15, 24])
```

In [40]:
```
np.sum(a,axis=0)
```
Out[40]:
```
array([12, 15, 18])
```

In [41]:
```
np.median(a)
```
Out[41]:
```
5.0
```

In [42]:
```
np.mean(a)
```
Out[42]:
```
5.0
```

In [43]:
```
np.std(a)
```
Out[43]:
```
2.581988897471611
```

In [44]:
```
np.sort(x4)
```
Out[44]:
```
array([  1,   2,   3,   4,   5,   6,   7,   8, 100, 900])
```

In [46]:
```
m=np.where(x1==x4)
m
```
Out[46]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8], dtype=int64),)
```

In [47]:
```
np.where(x1%x2==0)
```
Out[47]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int64),)
```

In [48]:
```
np.where(x1>5,x1,0)
```
Out[48]:
```
array([100,   0,   0,   0,   0,   0,   6,   7,   8,   9])
```

In [ ]: