NUMPY

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. array vs list array are collection of items that are stored at contiguous memory locations, should be same data type/uses less memory to store data/for creation of n-dimensional array/ finding elemnts in numpy is easy list cannot directly handle mathematical operations while array can array is faster than list

In [2]:
```
import numpy as np
```

In [3]:
```
import numpy as np
np.__version__
```

Out[3]: '2.1.3'

In [4]:
```
my_list = [0,1,2,3,4,5]
print(my_list)
type(my_list)
```

```
[0, 1, 2, 3, 4, 5]
```

Out[4]: list

In [5]:
```
my_list1 = [0,1,2,3,4,5]
my_list2 = [4]
print(my_list1+my_list2)
```

```
[0, 1, 2, 3, 4, 5, 4]
```

In [6]:
```
arr1=np.array([0,1,2,3,4,5])
arr2=np.array([4])            #performs mathametical calculations
print(arr1+arr2)
```

```
[4 5 6 7 8 9]
```

In [7]:
```
#converting list to array
arr = np.array(my_list)
print(type(arr))
arr
```

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

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

In [8]:
```
a=np.array([1,2,3])
a
```

Out[8]: array([1, 2, 3])

In [9]:
```
my_list = [[0,1,2],[3,4,5,6,7]]
my_list
```

Out[9]: [[0, 1, 2], [3, 4, 5, 6, 7]]

In [10]:
```
my_list =np.array([[0,1,2],[3,4,5,6,7]])    # cannot create of different length
my_list
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 1
----> 1 my_list =np.array([[0,1,2],[3,4,5,6,7]])    # cannot create of different
length
      2 my_list

ValueError: setting an array element with a sequence. The requested array has an
inhomogeneous shape after 1 dimensions. The detected shape was (2,) + inhomogeneo
us part.
```

In [11]:
```
ar1d=np.array([1,2,3,4])
print(ar1d)
print(type(ar1d))
np.ndim(ar1d)
```

```
[1 2 3 4]
<class 'numpy.ndarray'>
```

Out[11]: 1

In [12]:
```
ar2d=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(ar2d)
print(type(ar2d))
np.ndim(ar2d)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
<class 'numpy.ndarray'>
```

Out[12]: 2

In [13]:
```
ar3d=np.array([[[1,2],[3,4],[5,6]]])
print(ar3d)
np.ndim(ar3d)
```

```
[[[1 2]
  [3 4]
  [5 6]]]
```

Out[13]: 3

slicing

In [14]:
```
arr=np.array([10,20,30,40,50])
arr[0:2]
```

Out[14]: array([10, 20])

In [15]:
```
arr=np.array([10,20,30,40,50])
arr[2:]
```

Out[15]: array([30, 40, 50])

In [16]:
```
arr=np.array([10,20,30,40,50])
arr[:3]
```

Out[16]: array([10, 20, 30])

In [17]:
```
arr=np.array([10,20,30,40,50])
arr[2:5]
```

Out[17]:
```
array([30, 40, 50])
```

In [18]:
```
arr=np.array([10,20,30,40,50])
arr[:]
```

Out[18]:
```
array([10, 20, 30, 40, 50])
```

In [19]:
```
arr=np.array([10,20,30,40,50])
arr[::2]
```

Out[19]:
```
array([10, 30, 50])
```

In [20]:
```
arr=np.array([10,20,30,40,50])
arr[::-1]
```

Out[20]:
```
array([50, 40, 30, 20, 10])
```

In [21]:
```
arr=np.array([[10,20,30,40],[50,60,70,80]])
arr[0:2,1:3]
```

Out[21]:
```
array([[20, 30],
       [60, 70]])
```

In [22]:
```
arr=np.array([[10,20,30,40],[50,60,70,80]])
arr[0,1:3]
```

Out[22]:
```
array([20, 30])
```

In [23]:
```
arr=np.array([[10,20,30,40],[50,60,70,80]])
arr[1,3]
```

Out[23]:
```
np.int64(80)
```

In [24]:
```
arr=np.array([[10,20,30,40],[50,60,70,80]])
arr[2:3,2:3]
```

Out[24]:
```
array([], shape=(0, 1), dtype=int64)
```

In [25]:
```
b=np.random.randint(10,20,(5,4))       #slicing in matrix
b
```

Out[25]:
```
array([[12, 19, 19, 10],
       [15, 11, 14, 13],
       [18, 10, 12, 17],
       [13, 19, 10, 16],
       [12, 16, 11, 15]], dtype=int32)
```

In [26]:
```
b[:]
```

Out[26]:
```
array([[12, 19, 19, 10],
       [15, 11, 14, 13],
       [18, 10, 12, 17],
       [13, 19, 10, 16],
       [12, 16, 11, 15]], dtype=int32)
```

In [27]:
```
b[2:4]                    # : displays entire  row
```

Out[27]:
```
array([[18, 10, 12, 17],
       [13, 19, 10, 16]], dtype=int32)
```

In [28]:
```
b[1,3]                    # , displays a specific value
```

Out[28]:
```
np.int32(13)
```

In [29]:
```
b[0,-3]
```

Out[29]:
```
np.int32(19)
```

In [30]:
```
b[3]
```

Out[30]:
```
array([13, 19, 10, 16], dtype=int32)
```

In [ ]:
```

```

there are diff ways to create array

In [31]:
```
# np.array([])              #using a array
arr1=np.array([1,2,3,4])
arr1
```

Out[31]:
```
array([1, 2, 3, 4])
```

In [32]:
```
b=np.arange(10,100,10)   #using arange
b                         #creates array from 10 to 100 with step 10
```

Out[32]:
```
array([10, 20, 30, 40, 50, 60, 70, 80, 90])
```

In [33]:
```
b=np.arange(10,20)
b
```

Out[33]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [34]:
```
b=np.arange(-10,20,3)
b
```

Out[34]:
```
array([-10,  -7,  -4,  -1,   2,   5,   8,  11,  14,  17])
```

In [35]:
```
np.arange(20,10,-1)
```

Out[35]:
```
array([20, 19, 18, 17, 16, 15, 14, 13, 12, 11])
```

In [36]:
```
np.arange(3.0)
```

Out[36]:
```
array([0., 1., 2.])
```

In [37]:
```
c=np.linspace(0,10,5) #using linspace
c                     #5 values eveenly spaced b/w 0 to 10
```

Out[37]:
```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

In [38]:
```
d=np.logspace(0,10,5)  #5 log values b/w 0 to 10
d
```

Out[38]:
```
array([1.00000000e+00, 3.16227766e+02, 1.00000000e+05, 3.16227766e+07,
       1.00000000e+10])
```

In [ ]:
```

```

In [39]:
```
np.zeros(4)              #using zeros
```

Out[39]:
```
array([0., 0., 0., 0.])
```

In [40]:
```
e=np.zeros(4,dtype=int)
e
```

Out[40]:
```
array([0, 0, 0, 0])
```

In [41]:
```
e=np.zeros(4,int)
e
```

Out[41]:
```
array([0, 0, 0, 0])
```

In [42]:
```
e=np.zeros((10,5))
e
```

Out[42]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [43]:
```
e=np.zeros((10,5),int)
type(e)
e
```

Out[43]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

In [44]:
```
np.ones(3)
```

Out[44]:
```
array([1., 1., 1.])
```

In [45]:
```
np.ones(3,int)
```

Out[45]:
```
array([1, 1, 1])
```

In [46]:
```
f=np.ones(5,dtype=int)            #using ones
f
```

Out[46]:
```
array([1, 1, 1, 1, 1])
```

In [47]:
```
f=np.ones((3,5))
f
```

Out[47]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

In [48]:
```
f*25
```

Out[48]:
```
array([[25., 25., 25., 25., 25.],
       [25., 25., 25., 25., 25.],
       [25., 25., 25., 25., 25.]])
```

In [49]:
```
g=np.eye(4)   #prints 4x4 identity matrix
g
```

Out[49]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [50]:
```
full_array=np.full((3,3),7)      #prints the complete array
full_array
```

Out[50]:
```
array([[7, 7, 7],
       [7, 7, 7],
       [7, 7, 7]])
```

In [51]:
```
#rand randint

rand(3,2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[51], line 3
      1 #rand randint
----> 3 rand(3,2)

NameError: name 'rand' is not defined
```

In [52]:
```
rand(3,2)
random.rand(5)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[52], line 1
----> 1 rand(3,2)
      2 random.rand(5)

NameError: name 'rand' is not defined
```

In [53]:
```
np.random.rand(5)
```

Out[53]:
```
array([0.36013203, 0.66822321, 0.86059033, 0.55044198, 0.66727531])
```

In [54]:
```
np.random.rand(3,5)    #prints 3*5 matrix
```

Out[54]:
```
array([[0.85852401, 0.60621348, 0.9578482 , 0.77412245, 0.72143314],
       [0.24863571, 0.46185399, 0.15820703, 0.27314259, 0.16878062],
       [0.85667683, 0.60022196, 0.10491915, 0.62178335, 0.80759876]])
```

In [55]:
```
np.random.randint(2,20)     # random int b/w 2 to 20
```

Out[55]:
```
2
```

In [56]:
```
r=np.random.randint(2,15,4)      # 4   random integrers b/w 2 to 15
print(r)
type(r)
```

```
[13  4  4  3]
```

Out[56]:
```
numpy.ndarray
```

In [57]:
```
np.random.randint(10,40,(10,10))   #generates the element b/w 10 to 40 10*10 mat
```

Out[57]:
```
array([[22, 18, 37, 12, 21, 23, 29, 23, 18, 26],
       [27, 25, 12, 35, 12, 14, 16, 13, 18, 38],
       [36, 29, 22, 33, 24, 37, 24, 17, 17, 33],
       [26, 15, 38, 19, 31, 18, 23, 30, 25, 23],
       [33, 20, 23, 16, 20, 16, 22, 31, 32, 21],
       [23, 20, 14, 35, 24, 36, 28, 33, 12, 33],
       [12, 32, 21, 38, 37, 29, 39, 15, 35, 24],
       [33, 37, 26, 34, 19, 18, 28, 30, 26, 28],
       [15, 22, 27, 14, 35, 20, 20, 19, 27, 20],
       [15, 15, 25, 35, 39, 22, 21, 37, 20, 14]], dtype=int32)
```

masking

In [58]:
```
print(b)
id(b)
```

```
[-10  -7  -4  -1   2   5   8  11  14  17]
```

Out[58]:
```
2346787552784
```

In [59]:
```
b<15
```

Out[59]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
       False])
```

In [60]:
```
b[b<15]
```

Out[60]:
```
array([-10,  -7,  -4,  -1,   2,   5,   8,  11,  14])
```

In [61]:
```
b[b>=15]
```

Out[61]:
```
array([17])
```

In [62]:
```
#masking or filtering
```

properties-shape,reshape,size,type,ndim,astype

In [63]:
```
np.arange(1,13)
```

Out[63]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [64]:
```
np.arange(1,13).reshape(3,4)           #should match  in both same no of element
```

Out[64]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [65]:
```
np.arange(1,13).reshape(6,2)      #the reshape should match the no of elements
```

Out[65]:
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])
```

In [84]:
```
arr=np.arange(48)
print(arr)                    #1d tp 3d
newarr=arr.reshape(2,4,6)
newarr
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47]
```

Out[84]:
```
array([[[ 0,  1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10, 11],
        [12, 13, 14, 15, 16, 17],
        [18, 19, 20, 21, 22, 23]],

       [[24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35],
        [36, 37, 38, 39, 40, 41],
        [42, 43, 44, 45, 46, 47]]])
```

#reshaping 3 formats order-c order-a order-f

In [66]:
```
arr2=np.array([[1,2,3],[4,5,6]])          #shape
print(arr2)
arr2.shape
```

```
[[1 2 3]
 [4 5 6]]
```

Out[66]:
```
(2, 3)
```

In [67]:
```
print(np.shape(arr2))
```

```
(2, 3)
```

In [68]:
```
arr2=np.array([[1,2,3],[4,5,6]])          #size        Total no of elemnts
print(arr2)
```

```
arr2.size
```

```
[[1 2 3]
 [4 5 6]]
```

Out[68]: 6

In [91]:
```
ravel()   - view
flatten() - copy

arr2=np.array([[1,2,3],[4,5,6]])
print(arr2.ravel())
print(arr2.flatten())
```

```
[1 2 3 4 5 6]
[1 2 3 4 5 6]
```

In [69]:
```
print(np.size(arr2))
```

```
6
```

In [30]:
```
arr2=np.array([[1,2,3],[4,5,6]])            #number of dimensions
print(arr2)
np.ndim(arr2)
```

```
[[1 2 3]
 [4 5 6]]
```

Out[30]: 2

In [74]:
```
arr2.ndim
```

Out[74]: 2

In [79]:
```
arr=np.array([10,20,30,40])           #dtype
print(arr)
arr.dtype         #prints the type of data
```

```
[10 20 30 40]
```

Out[79]: dtype('int64')

In [80]:
```
arr=np.array([1.2,2.3,3.4])
int_arr=arr.astype(int)
print(int_arr)
```

```
[1 2 3]
```

In [76]:
```
np.array([1,2,3,4],dtype=float)
```

Out[76]: array([1., 2., 3., 4.])

In [77]:
```
np.array([1,2,3,4],float)
```

Out[77]: array([1., 2., 3., 4.])

In [81]:
```
arr2=np.array([[1,2,3],[4,5,6]])         #dtype
print(arr2)
arr2.dtype
```

```
[[1 2 3]
 [4 5 6]]
```

Out[81]: dtype('int64')

In [83]:
```
arr2=np.array([[1,2,3],[4,5,6]])          #change data type
print(arr2)
print(arr2.dtype)
arr3=arr2.astype(float)
print(arr3)
arr3.dtype
```

```
[[1 2 3]
 [4 5 6]]
int64
[[1. 2. 3.]
 [4. 5. 6.]]
```

Out[83]: dtype('float64')

mathametical functions

In [43]:
```
arr=np.array([1,2,3,4,5,6])

print(arr+5)
print(arr*2)
print(arr**2)
print(arr-5)
print(arr/2)
print(arr//2)       #same all
print(arr%2)
```

```
[ 6  7  8  9 10 11]
[ 2  4  6  8 10 12]
[ 1  4  9 16 25 36]
[-4 -3 -2 -1  0  1]
[0.5 1.  1.5 2.  2.5 3. ]
[0 1 1 2 2 3]
[1 0 1 0 1 0]
```

aggeragation fns sum,mean,min,max,std,var

In [41]:
```
arr=np.array([1,2,3,4,5,6])

print(np.sum(arr))
print(np.mean(arr))
print(np.min(arr))
print(np.max(arr))
print(np.std(arr))
print(np.var(arr))
```

```
21
3.5
1
6
1.707825127659933
2.9166666666666665
```

indexing,slicing

In [48]:
```
arr=np.array([10,20,30,40])  #accesing elements
print(arr[2])

#2d array - arr[row,column]
#3d array - arr[2,3,4]

#accesing multiple elemnts
print(arr[[0,3,1]])
```

```
[10 40 20]
```

In [47]:
```
#slicing

#array[start:stop:step]
# -1 reverse array
```

iterating

In [86]:
```
arr=np.array([1,2,3])   #iterating through 1d array
```

In [87]:
```
for x in arr:
    print(x)
```

```
1
2
3
```

In [88]:
```
arr=np.array([[1,2,3],[4,5,6]])   #2d array
for x in arr:
    print(x)
```

```
[1 2 3]
[4 5 6]
```

In [89]:
```
arr=np.array([[1,2,3],[4,5,6]])  #2d to print scalar means one by one
for x in arr:
    for j in x:
        print(j)
```

```
1
2
3
4
5
6
```

In [93]:
```
arr=np.array([[[1,2],[3,4]],[[5,6],[7,8]]])    #short cut to print scalar values
for x in np.nditer(arr):
```

```
    print(x)
```

```
1
2
3
4
5
6
7
8
```

array modification

In [92]:
```
arr=np.array([10,20,30,40])          #insert -add value at a mentioned index
print(arr)
new_arr = np.insert(arr,2,100)         #(og array,index,value)
print(new_arr)
```

```
[10 20 30 40]
[ 10  20 100  30  40]
```

In [96]:
```
arr2=np.array([[1,2,3],[4,5,6]])
print(arr2)
new_arr2=np.insert(arr2,1,[7,8],axis=None)
print(new_arr2)
```

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

In [97]:
```
arr=np.array([10,20,30,40])              # append - add value at last
new_arr = np.append(arr,[50,60])
new_arr
```

Out[97]:
```
array([10, 20, 30, 40, 50, 60])
```

joining numpy arrays

np.concatenate((array1,array2),axis=0) axis 0 -vertical stacking axis 1 - horizontal stacking

In [100…
```
arr1=np.array([1,2,3])
arr2= np.array([4,5,6])
arr = np.concatenate((arr1,arr2))
print(arr)
```

```
[1 2 3 4 5 6]
```

In [101…
```
np.concatenate((arr1,arr2),axis=1)
print(arr)
```

```
---------------------------------------------------------------------------
AxisError                                 Traceback (most recent call last)
Cell In[101], line 1
----> 1 np.concatenate((arr1,arr2),axis=1)
      2 print(arr)

AxisError: axis 1 is out of bounds for array of dimension 1
```

Joining Arrays Using Stack Functions

In [102…
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.stack((arr1, arr2), axis=1)
print(arr)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

In [103…
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])    #Stacking Along Rows
arr = np.hstack((arr1, arr2))
print(arr)
```

```
[1 2 3 4 5 6]
```

In [104…
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])    #Stacking Along colums
arr = np.vstack((arr1, arr2))
print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

In [105…
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])    #Stacking Along height depth
arr = np.dstack((arr1, arr2))
print(arr)
```

```
[[[1 4]
  [2 5]
  [3 6]]]
```

In [ ]:
```
```

In [107…
```
# Splitting NumPy Arrays

arr=np.array([1,2,3,4,5,6])
newarr=np.array_split(arr,3)     #splits the array
print(newarr[0])
print(newarr[1])
print(newarr[2])
```

```
[1 2]
[3 4]
[5 6]
```

In [108…
```
#transpose an array
e1= np.array([[1,2],[3,4]])      #transpose the array
transposed=np.transpose(e1)
transposed
```

Out[108…
```
array([[1, 3],
       [2, 4]])
```

In [109…
```
identity_mat=np.eye(4)
identity_mat
```

Out[109…
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [110…
```
g=np.array([1,2,3,4])    #mathematical functions
added=np.add(g,2)
print(added)
#or
added=g+2
added
```

```
[3 4 5 6]
```

Out[110…
```
array([3, 4, 5, 6])
```

In [111…
```
#square each element
squared=np.power(g,2)
squared
```

Out[111…
```
array([ 1,  4,  9, 16])
```

In [112…
```
#squareroot of each element
sqrt=np.sqrt(g)
sqrt
```

Out[112…
```
array([1.        , 1.41421356, 1.73205081, 2.        ])
```

In [ ]: