In [1]:

```
import numpy as np
```

In [2]:

```
l=[1,2,3]
ar=np.array(l)
ar
```

Out[2]:

```
array([1, 2, 3])
```

In [3]:

```
type(ar)
```

Out[3]:

```
numpy.ndarray
```

In [4]:

```
t=(1,2,3)
ar2=np.array(t)
ar2
```

Out[4]:

```
array([1, 2, 3])
```

advantages of numpy array over list

- consumes less memory
- fast as compared with python list
- convinient to use

In [5]:

```
l
```

Out[5]:

```
[1, 2, 3]
```

In [7]:

```
l+10  # element wise operation is not possible in list
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_5736\4038289675.py in <module>
----> 1 l+10  # element wise operation is not possible in list

TypeError: can only concatenate list (not "int") to list
```

In [8]:

```
ar
```

Out[8]:

```
array([1, 2, 3])
```

In [9]:

```
ar+10          #
```

Out[9]:

```
array([11, 12, 13])
```

In [10]:

```
ar.ndim
```

Out[10]:

```
1
```

In [11]:

```
ar.dtype  # data type of items
```

Out[11]:

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

In [12]:

```
ar.shape
```

Out[12]:

```
(3,)
```

In [13]:

```
ar.size
```

Out[13]:

```
3
```

In [14]:

```
ar.itemsize
```

Out[14]:

```
4
```

In [16]:

```
a1=np.array([10,20,30],dtype="complex")
a1
```

Out[16]:

```
array([10.+0.j, 20.+0.j, 30.+0.j])
```

In [18]:

```
a2=np.array([10,20,30],ndmin=2)
a2
```

Out[18]:

```
array([[10, 20, 30]])
```

In [19]:

```
a2.ndim
```

Out[19]:

```
2
```

In [20]:

```
b=np.array([10,22,53,67,89,45])
b
```

Out[20]:

```
array([10, 22, 53, 67, 89, 45])
```

In [21]:

```
b.sum()
```

Out[21]:

```
286
```

In [22]:

```
b.max()
```

Out[22]:

```
89
```

In [23]:

```
b.min()
```

Out[23]:

```
10
```

In [54]:

```
b.mean()
```

Out[54]:

```
0.5
```

In [55]:

```
b.std()  # standard deviation of data point of an array
```

Out[55]:

```
0.5
```

In [53]:

```
b.argmax()
```

Out[53]:

```
0
```

In [25]:

```
b.argmin()
```

Out[25]:

```
0
```

In [26]:

```
c=np.array([2,4,8,9,64,81])
c
```

Out[26]:

```
array([ 2,  4,  8,  9, 64, 81])
```

In [56]:

```
print(c.mean())
print(c.std())
```

```
5.0
2.8284271247461903
```

In [27]:

```
print(np.sqrt(c))
```

```
[1.41421356 2.         2.82842712 3.         8.         9.        ]
```

In [28]:

```
np.log(c)
```

Out[28]:

```
array([0.69314718, 1.38629436, 2.07944154, 2.19722458, 4.15888308,
       4.39444915])
```

In [57]:

```
n1=np.random.randint(100,500,25).reshape(5,5)
n1
```

Out[57]:

```
array([[474, 344, 405, 168, 480],
       [212, 448, 258, 326, 154],
       [226, 142, 497, 279, 497],
       [206, 197, 424, 378, 461],
       [234, 231, 243, 328, 409]])
```

In [58]:

```
n2=np.random.randint(10,50,25).reshape(5,5)
n2
```

Out[58]:

```
array([[43, 35, 23, 19, 29],
       [22, 37, 37, 38, 24],
       [23, 45, 44, 37, 31],
       [11, 32, 46, 17, 28],
       [22, 35, 44, 14, 12]])
```

In [60]:

```
np.add(n1,n2)
```

Out[60]:

```
array([[517, 379, 428, 187, 509],
       [234, 485, 295, 364, 178],
       [249, 187, 541, 316, 528],
       [217, 229, 470, 395, 489],
       [256, 266, 287, 342, 421]])
```

In [61]:

```
np.diff(n1,n2)
```

...

In [62]:

```
np.divide(n1,n2)
```

Out[62]:

```
array([[11.02325581,  9.82857143, 17.60869565,  8.84210526, 16.55172414],
       [ 9.63636364, 12.10810811,  6.97297297,  8.57894737,  6.41666667],
       [ 9.82608696,  3.15555556, 11.29545455,  7.54054054, 16.03225806],
       [18.72727273,  6.15625   ,  9.2173913 , 22.23529412, 16.46428571],
       [10.63636364,  6.6       ,  5.52272727, 23.42857143, 34.08333333]])
```

In [63]:

```
np.remainder(n1,n2)
```

Out[63]:

```
array([[ 1, 29, 14, 16, 16],
       [14,  4, 36, 22, 10],
       [19,  7, 13, 20,  1],
       [ 8,  5, 10,  4, 13],
       [14, 21, 23,  6,  1]], dtype=int32)
```

In [29]:

```
a1=np.array([90,56.2,'rama'])
a1
```

Out[29]:

```
array(['90', '56.2', 'rama'], dtype='<U32')
```

In [31]:

```
a1.dtype
```

Out[31]:

```
dtype('<U32')
```

difference between range and arange

- range is builtin function in python where as arange belongs to numpy

In [33]:

```
b=np.array(range(1,10,2))
b
```

Out[33]:

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

In [34]:

```
c=np.arange(1,10,2)
c
```

Out[34]:

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

In [35]:

```
a=np.arange(25)
a
```

Out[35]:

```
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])
```

In [36]:

```
a.ndim
```

Out[36]:

```
1
```

In [37]:

```
b=a.reshape(5,5)  # reshaping 1D into 2D
b
```

Out[37]:

```
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]])
```

In [38]:

```
b.ndim
```

Out[38]:

```
2
```

In [40]:

```
np.full([2,2],45)   #returns 2*2 array  filled with value 45
```

Out[40]:

```
array([[45, 45],
       [45, 45]])
```

In [41]:

```
np.eye(2)
```

Out[41]:

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

In [42]:

```
def grt(a,b):
    if a>b:
        return a
    else:
        return b
```

In [43]:

```
grt(90,150)
```

Out[43]:

```
150
```

In [44]:

```
l1=[10,9,8]
l2=[11,3,16]
grt(l1,l2)
```

Out[44]:

```
[11, 3, 16]
```

In [45]:

```
# vectorization is used to implement array operations without using loops

vgrt=np.vectorize(grt)
vgrt(l1,l2)
```

Out[45]:

```
array([11,  9, 16])
```

In [46]:

```
a=np.array([[1,2],[3,4]])
a
```

Out[46]:

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

In [47]:

```
b=np.identity(2)
b
```

Out[47]:

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

In [48]:

```
a+b
```

Out[48]:

```
array([[2., 2.],
       [3., 5.]])
```

In [49]:

```
a*b   # normal multiplication
```

Out[49]:

```
array([[1., 0.],
       [0., 4.]])
```

In [50]:

```
np.dot(a,b)  # matrix multiplication
```

Out[50]:

```
array([[1., 2.],
       [3., 4.]])
```

In [64]:

```
a-b
```

Out[64]:

```
array([[0., 2.],
       [3., 3.]])
```

In [65]:

```
a/b
```

```
C:\Users\meena\AppData\Local\Temp\ipykernel_5736\1348051284.py:1: RuntimeW
arning: divide by zero encountered in true_divide
  a/b
```

Out[65]:

```
array([[ 1., inf],
       [inf,  4.]])
```

In [66]:

```
a%b
```

```
C:\Users\meena\AppData\Local\Temp\ipykernel_5736\1820107994.py:1: RuntimeW
arning: invalid value encountered in remainder
  a%b
```

Out[66]:

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

In [67]:

```
np.exp(n1)
```

Out[67]:

```
array([[7.17107760e+205, 2.49632873e+149, 7.74934812e+175,
        9.15109281e+072, 2.89301918e+208],
       [1.17606185e+092, 3.66376739e+194, 1.11680238e+112,
        3.80190360e+141, 7.60939648e+066],
       [1.41433702e+098, 4.67537478e+061, 6.98807417e+215,
        1.47285655e+121, 6.98807417e+215],
       [2.91516588e+089, 3.59760050e+085, 1.38312148e+184,
        1.45651231e+164, 1.62089976e+200],
       [4.21607925e+101, 2.09906226e+100, 3.41632440e+105,
        2.80924790e+142, 4.23100071e+177]])
```

In [68]:

```
np.log(10)
```

Out[68]:

```
2.302585092994046
```

In [69]:

```
np.log(1)
```

Out[69]:

```
0.0
```

aggregation

- numpy sum function allows you to use optional argument called axis.
- axis=0 returns sum of each columns in numpy array
- axis=1 means row sum

In [70]:

```
a=np.array([[10,20],[30,40]])
a
```

Out[70]:

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

In [71]:

```
a.sum(axis=0)  # column sum , 10+30 , 20+40
```

Out[71]:

```
array([40, 60])
```

In [72]:

```
a.sum(axis=1)  #row sum
```

Out[72]:

```
array([30, 70])
```

saving data

In [75]:

```
# save numpy array as csv file

from numpy import asarray
from numpy import savetxt

data=asarray([[0,1,2,3,4,5,6,7,8,9]])
#data
# save data to csv file
savetxt('data.csv',data,delimiter=',')
```

load numpy array from .csv file

In [76]:

```
from numpy import loadtxt

new_data=loadtxt("data.csv",delimiter=',')
new_data
```

Out[76]:

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

save numpy to .npz file(binary)

- save the numpy array into a native binary format that is efficient to both save and load

In [79]:

```
# save numpy array as npy file

from numpy import asarray
from numpy import save

data=asarray([[0,1,2,3,4,5,6,7,8,9]])
save('data1.npy',data)
```

In [80]:

```
# load

from numpy import load
fdata=load('data1.npy')
print(fdata)
```

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

save numpy to .npz file

- suppose if the dataset contains a large amount of data combination of integers, collection of rescaled images(pixels) has to store in zip file.
- savez_compressed() function supports creating multiple arrays to a single file

In [ ]:

```
from numpy import asarray
from numpy import savez_compressed

data=asarray([[0,10,20,30,40]])
savez_compressed('cdata.npz',data)
```

In [ ]:

In [ ]:

In [ ]: