In [1]:
```
# NUMPY (Library) - it gives multidimensional array object and various deri
#Faster than list,, less memory use
# Uses - mathematical, logical, shape manipulation, sorting, selection, bas
#More numerical operations performed in NUMPY as compare to LIST
```

In [ ]:
```
```

In [2]:
```
import numpy as np
x = np.array([1,2,3,4])
print(x)
print(type(x))
```

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

In [3]:
```
y_list = [1,2,3,4]
print(type(y_list))
```

```
<class 'list'>
```

Creating NP Array

In [4]:
```
import numpy as np
n = np.array([1,2,3])
print(n)
```

```
[1 2 3]
```

In [5]:
```
type(n)
print(n.ndim)
```

```
1
```

user input array

In [6]:
```
import numpy as np
```

In [7]:
```
l = []
for i in range(1,5):
    a = int(input("Enter array items: "))
    l.append(a)

print(np.array(l))
print(type(l))
```

```
Enter array items: 4
Enter array items: 6
Enter array items: 7
Enter array items: 4
[4 6 7 4]
<class 'list'>
```

Types of Array

1D, 2D, 3D, Higher demensional array

In [8]:
```
a2 = np.array([[1,2,3,4],[1,2,3,4]])
print(a2)
print(a2.ndim)
```

```
[[1 2 3 4]
 [1 2 3 4]]
2
```

In [9]:
```
a3 = np.array([[[1,2,3,4], [1,2,3,4], [1,2,3,4]]])
print(a3)
print(a3.ndim)
```

```
[[[1 2 3 4]
  [1 2 3 4]
  [1 2 3 4]]]
3
```

In [10]:
```
#n dimensional array

a4 = np.array([1,2,3,4],ndmin = 10 )
print(a4)
print(a4.ndim)
```

```
[[[[[[[[[[1 2 3 4]]]]]]]]]]
10
```

Special types of array (filled with specific value)

Numpy array creation using functions

1. array filled with 0's
2. array filled with 1's

3. create an empty array
4. array with an range element
5. array diagonal element filled with 1's

In [11]:
```
#Zeros

ar = np.zeros(4)
ar1 = np.zeros((3,4))#passing rows n columns values
print(ar)
print(ar1)
```

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

In [12]:
```
#Ones -> drawback is it will take previous variable memory

ar1 = np.ones(4)
ar2 = np.ones((3,4))#passing rows n columns values
print(ar1)
print(ar2)

print(type(ar1))
```

```
[1. 1. 1. 1.]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
<class 'numpy.ndarray'>
```

In [13]:
```
#Arange (asc or desc)

ar = np.arange(0,50,2,dtype = int)
print(ar)
print(type(ar))
```

```
[ 0  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
 48]
<class 'numpy.ndarray'>
```

In [14]:
```
#Diagonal Matrix with value 1

dg = np.eye(4,4)
print(dg)
```

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

Arrays with Random Numbers

In [15]:
```
#rand()-> it gives +ve random numbers between 0 & 1

rn = np.random.rand(4)
rn1 = np.random.rand(4,4)
print(rn)
print()
print(rn1)
```

```
[0.23836568 0.15048031 0.27173132 0.18762964]

[[0.24231109 0.20504271 0.11757998 0.77813215]
 [0.57602377 0.98764502 0.76417045 0.81621198]
 [0.69383849 0.93252348 0.2209297  0.53910585]
 [0.1493271  0.46938945 0.82056402 0.73434148]]
```

In [16]:
```
#randn()-> it giver both +ve & -ve number close to 0
rnn = np.random.randn(4)
rnn1 = np.random.randn(4,4)
print(rnn)
print()
print(rnn1)
```

```
[-1.30226805  0.0744325  -1.40601905 -0.08377008]

[[ 1.01436594  1.90656037  0.5408086   1.05053002]
 [ 2.11219629 -0.11011584  0.94387281 -0.20278839]
 [ 0.9045895   0.77639527  1.76724775 -0.34083278]
 [-0.39497074  0.36772203  0.04932773 -0.93036015]]
```

In [17]:
```
#randint-> generate specific total integers numbers between max and min num

rint = np.random.randint(1,50,10)
print(rint)
```

```
[ 2 41 42 16  5 33 15 14 47  2]
```

Data type changing

In [18]:
```
dt = np.array((1,2,3,4),dtype='int')
print(dt)
print(dt.dtype)
```

```
[1 2 3 4]
int32
```

In [19]:
```
dt = np.array([1,2,3,4])

new = np.float32(dt)

#again new to reverse dtype
new1 = np.int_(new)

print(dt)
print(dt.dtype)

print(new)
print(new.dtype)

print(new1)
print(new1.dtype)
```

```
[1 2 3 4]
int32
[1. 2. 3. 4.]
float32
[1 2 3 4]
int32
```

In [20]:
```
import numpy as np
```

In [21]:
```
#We can use astype() for data type conversion
dt1 = np.array([1,2,3,4])

var = dt1.astype(float)

print(dt1)
print(dt1.dtype)

print(var)
print(var.dtype)
```

```
[1 2 3 4]
int32
[1. 2. 3. 4.]
float64
```

Shape n Reshape

In [22]:
```
var1 = np.array([[1,2,3,4],[1,2,3,4]])
print(var1)
print(var1.shape)
```

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

In [23]:
```
#Multidimensional
var2 = np.array([1,2,3,4], ndmin = 5)
print(var2)
print(var2.ndim)
print(var2.shape)
```

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

In [24]:
```
##RESHAPE
```

In [25]:
```
var3 = np.array([1,2,3,4,5,6])
print(var3)
print(var3.ndim)

print()

x = var3.reshape(2,3)            #passing rows, columns values in reshape()
print(x)
print(x.ndim)
```

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

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

In [26]:
```
#if we want to reshape in again 1dim array

one = x.reshape(-1)
print(one)
print(one.ndim)
```

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

Numpy arithmetics operations

In [27]:
```
#Addition, subtraction, multiply, division, power -> only symbol need to ch
#1Dimensional Array
```

In [28]:
```
var = np.array([1,2,3,4])
add = np.add(var,3)
print(add)
```

```
[4 5 6 7]
```

In [29]:
```
var1 = np.array([1,2,3,4])
var2 = np.array([1,2,3,4])
add12 = np.add(var1,var2)
print(add12)
```

```
[2 4 6 8]
```

In [30]:
```
#2D Array
```

In [31]:
```
var3 = np.array([[1,2,3,4],[1,2,3,4]])
var4 = np.array([[1,2,3,4],[1,2,3,4]])
print(var3)
print()
print(var4)
print()

add2 = np.add(var3,var4)
print(add2)
```

```
[[1 2 3 4]
 [1 2 3 4]]

[[1 2 3 4]
 [1 2 3 4]]

[[2 4 6 8]
 [2 4 6 8]]
```

In [32]:
```
#Arithmetic Functions
"""
min, max, argmin, argmax(it gives position value), sqrt, sin, cos, cumsum
"""
```

Out[32]:
```
'\nmin, max, argmin, argmax(it gives position value), sqrt, sin, cos, cums
um \n'
```

In [33]:
```
var4 = np.array([1,2,3,4,5,6,7,8])
print("min",np.min(var4),np.argmin(var4))
print("max",np.max(var4),np.argmax(var4))
#same as it is for sin,cos,sqrt
```

```
min 1 0
max 8 7
```

In [34]:
```
var5 = np.array([[2,3,4],[6,7,8]])
print("min", np.min(var5,axis=0))#axis=0 -> it refer column values
```

```
min [2 3 4]
```

Indexing & Slicing

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

In [36]:
```
#INDEXING
```

In [37]:
```
#1D
var6 = np.array([1,2,3,4,5])
#index value->   0,1,2,3,4
#index value->   -5,-4,-3,-2,-1

print(var6[1])
print(var6[-4])
```

```
2
2
```

In [38]:
```
#2D
var7 = np.array([[4,5,6],[7,8,9]])
#index value->     0    ,   1
#inside index->  [0,1,2],[0,1,2]
print(var7)
print()

print(var7[0,2])
print(var7[1,2])
```

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

6
9
```

In [44]:
```
#3D
var8 = np.array([[[1,2,3],[4,5,6]]])
print(var8)
print(var8.ndim)
print()
print(var8[0,0,1])
print(var8[0,1,1])
```

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

2
5
```

In [45]:
```
#SlICING
```

In [52]:
```
#1D
var9 = np.array([1,2,3,4,5,6,7])

print(var9)
print()

#slicing
print("1-6", var9[0:6])
print('2-end', var9[2:])
print('start-5', var9[:5])
#steps slicing
print('2steps data', var9[0:7:2])
```

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

1-6 [1 2 3 4 5 6]
2-end [3 4 5 6 7]
start-5 [1 2 3 4 5]
2steps data [1 3 5 7]
```

In [60]:
```
#2D
var10 = np.array([[1,2,3,4],[5,6,7,8]])

print(var10)
print()

#slicing
print('0 Index val->1-3', var10[0,0:3])
#                                index val, start:stop:end
print('1 Index val->1-3', var10[1,0:3])

var12 = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print(var12.ndim)
```

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

0 Index val->1-3 [1 2 3]
1 Index val->1-3 [5 6 7]
2
```

In [59]:
```
#3D
var11 = np.array([[[1,2,3,4],[5,6,7,8]]])
print(var11)
print(var11.ndim)

print()
print('0 index val-> 1-4', var11[0,1,0:])
```

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

0 index val-> 1-4 [5 6 7 8]
```

Iterating/Iterations

In [4]:
```
#1D
var12 = np.array([1,2,3,4,5])
print(var12)
print()

for i in var12:
    print(i)
```

```
[1 2 3 4 5]

1
2
3
4
5
```

In [7]:
```
#2D
var13 = np.array([[1,2,3,4],[5,6,7,8]])
print("Non-iterable\n",var13)
print()

print("Iterable")
for j in var13:
    for k in j:
        print(k)
```

```
Non-iterable
 [[1 2 3 4]
 [5 6 7 8]]

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

In [14]:
```
#3d

var14 = np.array([[[1,2,3,4],[5,6,7,8]]])
print("Non-iterable\n",var14)
print()
print("array dimension",var14.ndim)
print("Iterable")
for j in var14:
    for k in j:
        for a in k:
            print(a)
```

```
Non-iterable
 [[[1 2 3 4]
  [5 6 7 8]]]

array dimension 3
Iterable
1
2
3
4
5
6
7
8
```

In [15]:
```
#Alternate way by using-> np.nditer()
#3d

var15 = np.array([[[1,2,3,4],[5,6,7,8]]])
print("Non-iterable\n",var15)
print()
print("array dimension",var15.ndim)
print("Iterable")
for i in np.nditer(var15):
    print(i)
```

```
Non-iterable
 [[[1 2 3 4]
  [5 6 7 8]]]

array dimension 3
Iterable
1
2
3
4
5
6
7
8
```

In [16]:
```
#If we want to print both index and its value in case of large dataset then

#3d

var16 = np.array([[[1,2,3,4],[5,6,7,8]]])
print("Non-iterable\n",var16)
print()
print("array dimension",var16.ndim)
print("Iterable")
for i,d in np.ndenumerate(var16):
    print(i,d)
```

```
Non-iterable
 [[[1 2 3 4]
  [5 6 7 8]]]

array dimension 3
Iterable
(0, 0, 0) 1
(0, 0, 1) 2
(0, 0, 2) 3
(0, 0, 3) 4
(0, 1, 0) 5
(0, 1, 1) 6
(0, 1, 2) 7
(0, 1, 3) 8
```

Copy Vs View

Both function use to copy array data into another diffferent variable The only difference is -> 1.Copy owns the data , where View doesn't 2. The changes in copy data doesn't effect in original array data 3. The changes in View data will effect original array data and vice versa

In [18]:
```
#COPY
var17 = np.array([1,2,3,4])
co = var17.copy()

#if we make some changes in original data then,
var17[1] = 5

print("original data: ",var17)
print("copied data: ",co)
```

```
original data:  [1 5 3 4]
copied data:  [1 2 3 4]
```

In [19]:
```
#VIEW
var18 = np.array([1,2,3,4])
vi = var18.view()

#if we make some changes in original data then, automatically data in View
var18[1] = 5

print("original data: ",var18)
print("copied data: ",vi)
```

```
original data:  [1 5 3 4]
copied data:  [1 5 3 4]
```

JOIN & SPLIT function

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

#JOIN-> merging two or more array data but number of elements should be same for JOIN()

In [4]:
```
#1D
var19 = np.array([1,2,3,4])
var20 = np.array([5,6,7,8])

print("Merged array", np.concatenate((var19,var20)))
```

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

In [9]:
```
#2D -> array will merge on basis of axis=0,1

var21 = np.array([[1,2,3,4],[5,6,7,8]])
var22 = np.array([[55,66,77,88],[11,22,33,44]])

print("0 axis Merged array", np.concatenate((var21,var22),axis = 0))    #iT
print()
print("1 axis Merged array", np.concatenate((var21,var22),axis = 1))
#It will concatenate on basis of column
```

```
0 axis Merged array [[ 1  2  3  4]
 [ 5  6  7  8]
 [55 66 77 88]
 [11 22 33 44]]

1 axis Merged array [[ 1  2  3  4 55 66 77 88]
 [ 5  6  7  8 11 22 33 44]]
```

In [15]:
```
#Another way of concatenation basis on rows, column, diagonal(height)

var23 = np.array([1,2,3,4])
var24 = np.array([5,6,7,8])

ar_st = np.stack((var23,var24),axis = 0)
ar_st1 = np.stack((var23,var24),axis = 1)

#Rows(Horizontal)
ar_row = np.hstack((var23,var24))

#Column(Vertical)
ar_ver = np.vstack((var23,var24))

#Diagonal(Height)
ar_dia = np.dstack((var23,var24))

print("Axis 0 concatenate",ar_st)
print()
print("Axis 1 concatenate",ar_st1)
print()
print("Rows(Horizontal) concatenate",ar_row)
print()
print("Column (Vertical) concatenate",ar_ver)
print()
print("Diagonal (Height) concatenate",ar_dia)
```

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

Axis 1 concatenate [[1 5]
 [2 6]
 [3 7]
 [4 8]]

Rows(Horizontal) concatenate [1 2 3 4 5 6 7 8]

Column (Vertical) concatenate [[1 2 3 4]
 [5 6 7 8]]

Diagonal (Height) concatenate [[[1 5]
  [2 6]
  [3 7]
  [4 8]]]
```

#SPLIT -> breaks array into multiple array

In [16]:
```
#1D

var25 = np.array([1,2,3,4,5,6,7,8,9])
ar_sp = np.array_split(var25,3)
print()
print(ar_sp)
```

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

In [20]:
```
#2D

var26 = np.array([[1,2,3],[4,5,6]])
ar_sp = np.array_split(var26,3)
ar_sp_ax = np.array_split(var26,3,axis = 0)
ar_sp_ax1 = np.array_split(var26,3,axis = 1)
print()
print(ar_sp)
print()
print(ar_sp_ax)
print()
print(ar_sp_ax1)
```

```
[array([[1, 2, 3]]), array([[4, 5, 6]]), array([], shape=(0, 3), dtype=int
32)]

[array([[1, 2, 3]]), array([[4, 5, 6]]), array([], shape=(0, 3), dtype=int
32)]

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

Search(), Sort(), Search Sort(), Filter()

In [21]:
```
#Search Array -> for a certain value nad return the indexes that get a matc
```

In [24]:
```
var27 = np.array([2,3,1,4,8,6,9,5,12,2,2,2])

x = np.where(var27 == 2)
y = np.where((var27%2) == 0)
print(x)
print()
print(y)
```

```
(array([ 0,  9, 10, 11], dtype=int64),)

(array([ 0,  3,  4,  5,  8,  9, 10, 11], dtype=int64),)
```

In [27]:
```
#Search Sorted Array-> it helps to insert value at correct index value, whe
var27 = np.array([2,12,34,39,44,45])
ss = np.searchsorted(var27, 37)   #passing parameters like variable name, a
ss1 = np.searchsorted(var27, [3,6,9])
print(ss)
print()
print(ss1)
```

```
3

[1 1 1]
```

In [28]:
```
#SORT -> use to sort any alphanumerical values into ascending or decending

#1D
var28 = np.array([2,12,34,39,44,45])
print(np.sort(var28))
```

```
[ 2 12 34 39 44 45]
```

In [29]:
```
#2D
var29 = np.array([[2,12,34],[39,44,45]])
print(np.sort(var29))
```

```
[[ 2 12 34]
 [39 44 45]]
```

In [30]:
```
#Alphabets
var30 = np.array([["s","w","a"],["g","t","p"]])
print(np.sort(var30))
```

```
[['a' 's' 'w']
 ['g' 'p' 't']]
```

In [33]:
```
#Filter Array
var31 = np.array(["a","h","w","g"])
var32 = [True,False,False,True]

new_var = var31[var32]
print(new_var)
```

```
['a' 'g']
```

Shuffle()

In [34]:
```
var33 = np.array([1,2,3,4,5])
np.random.shuffle(var33)
print(var33)
```

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

Unique

In [36]:
```
var34 = np.array([1,2,3,4,2,5,2,6,2,8,2])
new = np.unique(var34, return_index=True,return_counts=True)
print(new)
```

```
(array([1, 2, 3, 4, 5, 6, 8]), array([0, 1, 2, 3, 5, 7, 9], dtype=int64),
array([1, 5, 1, 1, 1, 1, 1], dtype=int64))
```

Resize

In [38]:
```
var35 = np.array([2,3,5,22,6,8])
new1 = np.resize(var35,(3,2))
print(new1)
```

```
[[ 2  3]
 [ 5 22]
 [ 6  8]]
```

INSERT() & DELETE()

In [51]:
```
#1D
var36 = np.array([2,3,4,5,6])
print(var36)
print()
print(var36.ndim)

#np.insert(var_nmae,index_val,insert_val)
var37 = np.insert(var36, 3, 33)
print(var37)
print()

#multiple index val insertion = np.insert(var_nmae,(index_val1index_val2),i
var38 = np.insert(var36, (3,5), 33)
print(var38)
print()

#np.append -> directly insert value into array but float number insertion i
var44 = np.append(var36,3.3)
print(var44)
```

```
[2 3 4 5 6]

1
[ 2  3  4 33  5  6]

[ 2  3  4 33  5  6 33]

[2.  3.  4.  5.  6.  3.3]
```

In [55]:
```
#2D

var39 = np.array([[1,2,3],[4,5,6]])

var40 = np.insert(var39, 2, 8, axis = 0)         #Row insertion done
print(var40)
print()

var41 = np.insert(var39, 2, 8, axis = 1)         #Column insertion done
print(var41)
print()

var42 = np.insert(var39,2,[7,8,9], axis=0)
print(var42)
print()

#var43 = np.insert(var39,2,[7,8,9], axis=1)       #It will throw an error b
#print(var43)

var45 = np.append(var39,[[7.7,8.8,9.9]], axis=0)
print(var45)
print()
```

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

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

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

[[1.  2.  3. ]
 [4.  5.  6. ]
 [7.7 8.8 9.9]]
```

In [65]:
```
#DELETE
var46 = np.array([1,2,3,4,5,6,7])
print(var46)
var47 = np.delete(var46, 2)
print()
print(var47)
print()
var52 = np.delete(var46,[0,1])
print(var52)
```

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

[1 2 4 5 6 7]

[3 4 5 6 7]
```

In [63]:
```
var48 = np.array([[1,2,3,4],
                  [5,6,7,8],
                  [9,10,11,12]])
print(var48)
print()
var49 = np.delete(var48,0,axis =1)        #Deleting first column
print(var49)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

[[ 2  3  4]
 [ 6  7  8]
 [10 11 12]]
```

In [64]:
```
var50 = np.array([[1,2,3,4],
                  [5,6,7,8],
                  [9,10,11,12]])
print(var50)
print()
var51 = np.delete(var50,0,axis =0)        #Deleting first row
print(var51)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

[[ 5  6  7  8]
 [ 9 10 11 12]]
```

MATRIX

In [80]:
```
var52 = np.matrix([[1,2],[1,2]])
var53 = np.matrix([[1,2],[1,2]])

print(var52)
print(type(var52))
print()

print(var52*var53)
print()
print(var52.dot(var53))
```

```
[[1 2]
 [1 2]]
<class 'numpy.matrix'>

[[3 6]
 [3 6]]

[[3 6]
 [3 6]]
```

In [72]:
```
var54 = np.array([[1,2,3],[1,2,3]])
print(var54*var54)
```

```
[[1 4 9]
 [1 4 9]]
```

MATRIX() -> 1.Transpose 2.Swapaxes 3.Inverse 4.Power 5.Determinate

In [74]:
```
#Transpose

var55 = np.matrix([[1,2],[3,4],[5,6]])
print(np.transpose(var55))
print()
print(var55.T)
```

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

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

In [78]:
```
#Swapaxes
var56 = np.matrix([[1,2,3],[4,5,6]])
print(np.swapaxes(var56,0,1))
```

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

In [79]:
```
#INVERSE
var57 = np.matrix([[1,2],[3,4]])
print(var57)
print()

print(np.linalg.inv(var57))
```

```
[[1 2]
 [3 4]]

[[-2.   1. ]
 [ 1.5 -0.5]]
```

In [84]:
```
#POWER
var58 = np.matrix([[1,2],[3,4]])
print(var58)
print()

print(np.linalg.matrix_power(var58,2))
print()
print(np.linalg.matrix_power(var58,0))
print()
print(np.linalg.matrix_power(var58,-2))
print()
```

```
[[1 2]
 [3 4]]

[[ 7 10]
 [15 22]]

[[1 0]
 [0 1]]

[[ 5.5  -2.5 ]
 [-3.75  1.75]]
```

In [85]:
```
#DETERMINATE

#-       -
#| a b c |
#| d e f |  = a(ef - fh) - b(di - fg) + c(dh - eg)
#| g h i |
#-       -
```

In [87]:
```
var59 = np.matrix([[1,2],[3,4]])
print(var59)
print()
print(np.linalg.det(var59))
```

```
[[1 2]
 [3 4]]

-2.0000000000000004
```

In [ ]:
```

```