CREATING NUMPY ARRAY

In [1]:

```
import numpy as np
```

In [2]:

```
np.array([1,45,6,77,78,98])
```

Out[2]:

```
array([ 1, 45,  6, 77, 78, 98])
```

In [3]:

```
a=np.array([87,88,89,90,91,92])
print(a)
```

```
[87 88 89 90 91 92]
```

In [5]:

```
a=np.array([87,88,89,90,91,92])
print('Array is:',a)
```

```
Array is: [87 88 89 90 91 92]
```

In [6]:

```
type(a)
```

Out[6]:

```
numpy.ndarray
```

In [7]:

```
b=np.array([[1,2,3,4,5],[6,7,8,9,10]])
print('2D Array:',b)
```

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

changing the datatype of array

In [9]:

```
c=np.array([11,12,13,14],dtype=float)
print(c)
```

```
[11. 12. 13. 14.]
```

In [11]:

```
c=np.array([11,12,13,14],dtype=int)
print(c)
```

```
[11 12 13 14]
```

In [12]:

```
c=np.array([11,12,13,14],dtype=bool)
print(c)
```

```
[ True  True  True  True]
```

In [15]:

```
c=np.array([0,12,13,14],dtype=bool)
print(c)
```

```
[False  True  True  True]
```

In [16]:

```
c=np.array([11,12,13,14],dtype=complex)
print(c)
```

```
[11.+0.j 12.+0.j 13.+0.j 14.+0.j]
```

ARANGE FUNCTION

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

```
Out[17]: array([0])
```

```
In [18]: np.arange(10) #it includes the 1st num ans excludes the last (n-1)
```

```
Out[18]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [19]: np.arange(1,10)# range between numbers
```

```
Out[19]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [20]: np.arange(1,25,2) #step of two
```

```
Out[20]: array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23])
```

RESHAPE FUNCTION

```
In [21]: np.arange(1,25)
```

```
Out[21]: array([ 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 [23]: np.arange(0,25).reshape(5,5) #converted to five rows and 5 columns
```

```
Out[23]: 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 [24]: np.arange(1,11).reshape(2,5) #converted to two rows and five columns
```

```
Out[24]: array([[ 1,  2,  3,  4,  5],
                [ 6,  7,  8,  9, 10]])
```

ONES

```
In [25]: np.ones((6,5))
```

Out[25]:

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

In [29]:

```
a=np.ones((5,5)).dtype=int
print(a)
```

```
<class 'int'>
```

ZEROS

In [31]:

```
np.zeros((5,5))
```

Out[31]:

```
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.]])
```

RANDOM NUMBERS

In [32]:

```
np.random.rand(1,10) # random numbers will be dispalyed in the range given ,only fl
```

Out[32]:

```
array([[0.3213583 , 0.40162707, 0.94759656, 0.0565033 , 0.40510872,
        0.80767039, 0.85624948, 0.84779316, 0.90832846, 0.30933725]])
```

In [34]:

```
np.random.randint(1,25) # random number is called
```

Out[34]:

```
22
```

LINESPACE

In [36]:

```
np.linspace(-10,10,15)
```

Out[36]:

```
array([-10.        ,  -8.57142857,  -7.14285714,  -5.71428571,
        -4.28571429,  -2.85714286,  -1.42857143,   0.        ,
         1.42857143,   2.85714286,   4.28571429,   5.71428571,
         7.14285714,   8.57142857,  10.        ])
```

IDENTITY MATRIX

In [37]:

```
np.identity(3)
```

```
Out[37]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

```
In [38]: np.eye(3)
```

```
Out[38]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

```
In [39]: np.identity(5)
```

```
Out[39]: array([[1., 0., 0., 0., 0.],
                [0., 1., 0., 0., 0.],
                [0., 0., 1., 0., 0.],
                [0., 0., 0., 1., 0.],
                [0., 0., 0., 0., 1.]])
```

ARRAY ATTRIBUTES

```
In [41]: a=np.arange(10) #1D
         print(a)
```

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

```
In [50]: b=np.arange((25),dtype=float).reshape(5,5)
         print(b)
```

```
[[ 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 [49]: np.arange(8).reshape(2,2,2)# tensor
```

```
Out[49]: array([[[0, 1],
                 [2, 3]],

                [[4, 5],
                 [6, 7]]])
```

NDIM

```
In [52]: b.ndim
```

```
Out[52]: 2
```

SHAPE

```
In [53]: b.shape

Out[53]: (5, 5)

In [ ]: #SIZE

In [54]: b.size #num of items in the array

Out[54]: 25
```

ITEM SIZE

```
In [55]: b.itemsize

Out[55]: 8

In [56]: print(b.dtype)

float64

In [57]: print(a.dtype)

int64
```

CHANGING DATA TYPES

```
In [58]: x=np.array([12,13,15,18,12.45])
         print(x)

[12.    13.    15.    18.    12.45]

In [59]: type(x)

Out[59]: numpy.ndarray

In [61]: x.astype(int)

Out[61]: array([12, 13, 15, 18, 12])
```

ARRAY OPERATIONS

```
In [62]: z1=np.arange(12).reshape(3,4)
         print(z1)

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

In [64]:
```
z2=np.arange(12,24).reshape(3,4)
print(z2)
```

```
[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

In [ ]:
```
# SCALAR OPERATIONS
```

In [65]:
```
z1+2 #adds the value to each element
```

Out[65]:
```
array([[ 2,  3,  4,  5],
       [ 6,  7,  8,  9],
       [10, 11, 12, 13]])
```

In [66]:
```
z1
```

Out[66]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [67]:
```
z1-1 # subtraction
```

Out[67]:
```
array([[-1,  0,  1,  2],
       [ 3,  4,  5,  6],
       [ 7,  8,  9, 10]])
```

In [68]:
```
z1
```

Out[68]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [69]:
```
z1*5# multiplication
```

Out[69]:
```
array([[ 0,  5, 10, 15],
       [20, 25, 30, 35],
       [40, 45, 50, 55]])
```

In [70]:
```
z1/2 # division
```

Out[70]:
```
array([[0. , 0.5, 1. , 1.5],
       [2. , 2.5, 3. , 3.5],
       [4. , 4.5, 5. , 5.5]])
```

In [71]:
```
z1//2 #int division
```

Out[71]:
```
array([[0, 0, 1, 1],
       [2, 2, 3, 3],
       [4, 4, 5, 5]])
```

In [72]:
```
z1**2 # power off
```

Out[72]:
```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]])
```

```
In [73]: z1%5 3modulus

Out[73]: array([[0, 1, 2, 3],
                [4, 0, 1, 2],
                [3, 4, 0, 1]])

In [74]: z1>2

Out[74]: array([[False, False, False,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True]])

In [75]: z1<2

Out[75]: array([[ True,  True, False, False],
                [False, False, False, False],
                [False, False, False, False]])
```

VECTOR OPERATION

```
In [76]:     z1+z2

Out[76]: array([[12, 14, 16, 18],
                [20, 22, 24, 26],
                [28, 30, 32, 34]])

In [77]: z1*z2

Out[77]: array([[  0,  13,  28,  45],
                [ 64,  85, 108, 133],
                [160, 189, 220, 253]])

In [78]: z1/z2

Out[78]: array([[0.        , 0.07692308, 0.14285714, 0.2       ],
                [0.25      , 0.29411765, 0.33333333, 0.36842105],
                [0.4       , 0.42857143, 0.45454545, 0.47826087]])

In [79]: z1//z2

Out[79]: array([[0, 0, 0, 0],
                [0, 0, 0, 0],
                [0, 0, 0, 0]])

In [80]: z1%z2

Out[80]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]])
```

ARRAY FUNCTIONS

In [84]:
```
a=np.random.randint((3,3))
print(a)
```

```
[2 1]
```

In [85]:
```
a=np.arange(1,25).reshape(6,4)
print(a)
```

```
[[ 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 [89]:
```
a=np.random.rand(4,4)

a=np.round(a*100)
print(a)
```

```
[[50. 58. 41.  1.]
 [68. 51. 63.  4.]
 [37.  8. 17. 27.]
 [28. 59.  9. 35.]]
```

In [90]:
```
np.max(a)
```

Out[90]:
```
np.float64(68.0)
```

In [91]:
```
np.min(a)
```

Out[91]:
```
np.float64(1.0)
```

In [92]:
```
np.sum(a)# sum of all elements
```

Out[92]:
```
np.float64(556.0)
```

In [95]:
```
np.max(a,axis=1) # gives the large element in each row
```

Out[95]:
```
array([58., 68., 37., 59.])
```

In [96]:
```
np.max(a,axis=0) # gives the large element in each col
```

Out[96]:
```
array([68., 59., 63., 35.])
```

In [97]:
```
np.prod(a,axis=1) #product of each row
```

Out[97]:
```
array([118900., 873936., 135864., 520380.])
```

In [99]:
```
np.prod(a,axis=0) # product od each col
```

Out[99]:
```
array([3522400., 1396176.,  395199.,    3780.])
```

In [100…
```
a
```

```
Out[100…  array([[50., 58., 41.,  1.],
                 [68., 51., 63.,  4.],
                 [37.,  8., 17., 27.],
                 [28., 59.,  9., 35.]])
```

STATISTICAL OPERATIONS

MEAN MEDIAN MODE

```
In [101…  np.mean(a)# gives mean of all elements
```

```
Out[101…  np.float64(34.75)
```

```
In [102…  a.mean(axis=1)# gives mean of each row
```

```
Out[102…  array([37.5 , 46.5 , 22.25, 32.75])
```

```
In [103…  a.mean(axis=0)#gives mean of each col
```

```
Out[103…  array([45.75, 44.  , 32.5 , 16.75])
```

```
In [104…  np.median(a)# gives median of all elements
```

```
Out[104…  np.float64(36.0)
```

```
In [106…  np.median(a,axis=1) # gives median of each row
```

```
Out[106…  array([45.5, 57. , 22. , 31.5])
```

```
In [107…  np.median(a,axis=0)# gives median of each col
```

```
Out[107…  array([43.5, 54.5, 29. , 15.5])
```

```
In [108…  np.std(a)
```

```
Out[108…  np.float64(21.588480724682782)
```

```
In [109…  np.std(a,axis=1)
```

```
Out[109…  array([21.914607  , 25.30316186, 10.84838698, 17.89378384])
```

```
In [110…  np.std(a,axis=0)
```

```
Out[110…  array([15.03953124, 21.01190139, 21.18372016, 14.56665713])
```

```
In [111…  np.var(a)
```

```
Out[111…  np.float64(466.0625)
```

ROUND FLOOR CEIL

In [113…

```
arr=np.array([1.3,2.5,3.5,4.7,8.9])
arr
```

Out[113…

```
array([1.3, 2.5, 3.5, 4.7, 8.9])
```

In [116…

```
np.round(arr)
```

Out[116…

```
array([1., 2., 4., 5., 9.])
```

In [117…

```
np.floor(arr)
```

Out[117…

```
array([1., 2., 3., 4., 8.])
```

In [118…

```
np.ceil(arr)
```

Out[118…

```
array([2., 3., 4., 5., 9.])
```

INDEXING AND SLICING

In [120…

```
p1=np.arange(10)
p2=np.arange(12).reshape(3,4)
p3=np.arange(8).reshape(2,2,2)
```

In [121…

```
p1
```

Out[121…

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

In [122…

```
p2
```

Out[122…

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [123…

```
p3
```

Out[123…

```
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

INDEXING ON 1D ARRAY

In [124…

```
p1
```

```
Out[124…   array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [125…   p1[0]

Out[125…   np.int64(0)

In [126…   p1[-1]

Out[126…   np.int64(9)

In [127…   p1[1]

Out[127…   np.int64(1)

In [128…   p2

Out[128…   array([[ 0,  1,  2,  3],
                  [ 4,  5,  6,  7],
                  [ 8,  9, 10, 11]])
```

INDEXING ON 2D ARRAY

```
In [131…   p2[0]

Out[131…   array([0, 1, 2, 3])

In [132…   p2[2]

Out[132…   array([ 8,  9, 10, 11])

In [133…   p2[1]

Out[133…   array([4, 5, 6, 7])

In [134…   p2[-1]

Out[134…   array([ 8,  9, 10, 11])

In [135…   p2[1,2]

Out[135…   np.int64(6)

In [136…   p2[2,3]

Out[136…   np.int64(11)

In [137…   p2[1,0]

Out[137…   np.int64(4)
```

SLICING

```
In [138…  p1

Out[138…  array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [139…  p1[2:5]

Out[139…  array([2, 3, 4])

In [140…  p1[2:5:2]

Out[140…  array([2, 4])

In [141…  p2

Out[141…  array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])

In [142…  p2[0,:]# returns only first row

Out[142…  array([0, 1, 2, 3])

In [143…  p2[:,0]# returns onl;y first col

Out[143…  array([0, 4, 8])

In [144…  p2[:,2]

Out[144…  array([ 2,  6, 10])

In [145…  p2[2,:]

Out[145…  array([ 8,  9, 10, 11])

In [146…  p2[1:3,1:3]

Out[146…  array([[ 5,  6],
                 [ 9, 10]])

In [147…  p2[1:3]

Out[147…  array([[ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])

In [149…  p2[::2,::3]

Out[149…  array([[ 0,  3],
                 [ 8, 11]])
```

```
In [150…  p2[::2]

Out[150…  array([[ 0,  1,  2,  3],
                 [ 8,  9, 10, 11]])

In [151…  p2

Out[151…  array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])

In [152…  p2[::1]

Out[152…  array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])

In [155…  p2[1:]

Out[155…  array([[ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])

In [156…  p2[::2,1::2]

Out[156…  array([[ 1,  3],
                 [ 9, 11]])

  In [ ]:

  In [ ]:
```

TRANSPOSE

```
In [158…  p2

Out[158…  array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])

In [159…  np.transpose(p2)

Out[159…  array([[ 0,  4,  8],
                 [ 1,  5,  9],
                 [ 2,  6, 10],
                 [ 3,  7, 11]])

In [160…  p3.T

Out[160…  array([[[0, 4],
                  [2, 6]],

                 [[1, 5],
                  [3, 7]]])
```

```
In [161…  p1.T

Out[161…  array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [162…  p2

Out[162…  array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])
```

RAVEL FUNCTION

```
In [163…  p2.ravel()# it converts any nd array to 1d array

Out[163…  array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

  In [ ]:

  In [ ]:

  In [ ]:

In [165…  p3.ravel()

Out[165…  array([0, 1, 2, 3, 4, 5, 6, 7])

In [166…  p1

Out[166…  array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [167…  s2=np.arange(12).reshape(3,4)
          s3=np.arange(12,24).reshape(3,4)

In [168…  s3

Out[168…  array([[12, 13, 14, 15],
                 [16, 17, 18, 19],
                 [20, 21, 22, 23]])

In [169…  s2

Out[169…  array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]])
```

STACKING

HORIZONTAL STACKING

In [171…

```
s2
```

Out[171…

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [172…

```
s3
```

Out[172…

```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [174…

```
np.hstack((s2,s3)) # horizontal stacking
```

Out[174…

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

In [175…

```
np.vstack((s2,s3))# vertical stacking
```

Out[175…

```
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]])
```

In [ ]:

```

```