Creating Numpy array

```
import numpy as np
```

```
np.array([2,4,56,422,32,1])  # 1D array
```

```
array([  2,   4,  56, 422,  32,   1])
```

```
a = np.array([2,4,56,422,32,1])  #Vector
```

```
type(a)
```

```
numpy.ndarray
```

```
# 2D Array (Matrix)
```

```
new = np.array([[45,34,22,2],[24,55,3,22]])
print(new)
```

```
[[45 34 22  2]
 [24 55  3 22]]
```

```
# 3 D ---- # Tensor
```

```
np.array ([[2,3,33,4,45],[23,45,56,66,2],[357,523,32,24,2],
[32,32,44,33,234]])
```

```
array([[  2,   3,  33,   4,  45],
       [ 23,  45,  56,  66,   2],
       [357, 523,  32,  24,   2],
       [ 32,  32,  44,  33, 234]])
```

dtype The desired data-type for the array.If not given,then the type will be determine as the minimum type required to hold the objects in the sequence.

```
np.array([11,23,44] , dtype = float)
```

```
array([11., 23., 44.])
```

```
np.array([11,23,44] , dtype =bool) # Here True becoz , python treats
Non - zero
```

```
array([ True,  True,  True])
```

```
np.array([11,23,44] , dtype =bool) # Here True becoz , python treats
Non-zero
```

```
array([ True,  True,  True])
```

```
np.array([11,23,44] , dtype =complex)
```

```
array([11.+0.j, 23.+0.j, 44.+0.j])
```

arange

arrange can be called with a verying number of positional arguments

```
np.arange(1,25)  # 1 -included , 25 - last one got excluded
```

```
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])
```

```
np.arange(1,25,2) #strides ---> Alternate numbers
```

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

reshape

both of number products should be equal to number of items present inside the array.

```
np.arange(1,11).reshape(5,2) # converted 5 rows and 2 columns
```

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

```
np.arange(1,11).reshape(2,5) # converted 2 rows and 5 columns
```

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

```
np.arange(1,13).reshape(3,4) # converted 3 rows and 4 columns
```

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

ones & Zeros

you can initialize the values and create values . ex:in deep learning weight shape

```
# np.ones and np.zeros
np.ones((3,4)) # we have to mention iside tuple
```

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

```
np.zeros((3,4))
```

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

```
# Another Type ---> random()
np.random.random((3,4))
```

```
array([[0.43859811, 0.13297164, 0.2545046 , 0.27959907],
       [0.06963859, 0.57129159, 0.17650449, 0.2268196 ],
       [0.61385974, 0.89838252, 0.88521966, 0.88364076]])
```

linespace

```
np.linspace(-10,10,10) # here: lower range,upper range,number of items to gen
```

```
array([-10.        ,  -7.77777778,  -5.55555556,  -3.33333333,
        -1.11111111,   1.11111111,   3.33333333,   5.55555556,
         7.77777778,  10.        ])
```

```
np.linspace(-2,12,6)
```

```
array([-2. ,  0.8,  3.6,  6.4,  9.2, 12. ])
```

identity

```
# creating the identity matrix
np.identity(3)
```

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

```
np.identity(6)
```

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

Array Attributes

```
a1 = np.arange(10) #1D
a1
```

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

```
a2 =np.arange(12, dtype =float).reshape(3,4)  # Matrix
a2
```

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

```
a3 = np.arange(8).reshape(2,2,2) # 3D --> Tensor
a3
```

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

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

ndim

```
a1.ndim
```

```
1
```

```
a2.ndim
```

```
2
```

```
a3.ndim
```

```
3
```

shape

```
a1.shape # 1D array has 10 Items
```

```
(10,)
```

```
a2.shape # 3 rows and 4 columns
```

```
(3, 4)
```

```
a3.shape # first ,2 says it consists of 2Darrays .2,2 gives no.of rows 
and column
```

```
(2, 2, 2)
```

size

```
a3
```

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

a3.size # it has 8 items . like shape :2,2,2 = 8

8

a2

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

a2.size

12
```

item size

```
a1

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





a1.itemsize # bytes

4

a2.itemsize # integer 64 gives = 8 bytes

8

a3.itemsize # integer 32 gives = 4 bytes

4
```

dtype

```
print(a1.dtype)
print(a2.dtype)
print(a3.dtype)

int32
float64
int32
```

changing data type

```
#astype

x = np.array([33,22,2.5])
x

array([33. , 22. ,  2.5])

 x.astype(int)

array([33, 22,  2])
```

Array operations

```
z1 = np.arange(12).reshape(3,4)
z2 = np.arange(12,24).reshape(3,4)
z1

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

z2

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

scalar operations

```
# arithmetic
z1 + 2

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

# Subtraction
z1 - 2

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

#Multiplication
z1 * 2

array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14],
       [16, 18, 20, 22]])
```

```
# power
z1 ** 2
```

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

```
## Module
z1 % 2
```

```
array([[0, 1, 0, 1],
       [0, 1, 0, 1],
       [0, 1, 0, 1]], dtype=int32)
```

relational operators

```
z2
```

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

```
z2 > 20
```

```
array([[False, False, False, False],
       [False, False, False, False],
       [False,  True,  True,  True]])
```

Vector Operation

```
z1
```

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

```
z2
```

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

```
#Arthemetic
z1 + z2 # both numpy array shape is same, we can add item wise
```

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

```
z1 * z2
```

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

```
z1 - z2
```

```
array([[-12, -12, -12, -12],
       [-12, -12, -12, -12],
       [-12, -12, -12, -12]])
```

```
z1/z2
```

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

Array Functions

```
k1 = np.random.random((3,3))
k1 = np.round(k1*100)
k1
```

```
array([[23., 35., 37.],
       [19., 18., 55.],
       [67., 87., 82.]])
```

```
# Max
np.max(k1)
```

```
87.0
```

```
# min
np.min(k1)
```

```
18.0
```

```
# sum
np.sum(k1)
```

```
423.0
```

```
#prod ----> Multiplication
```