Numpy

The basic ndarray is created using an array function in NumPy: numpy.array

syntax **: numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0) returns a array object**

```
>>> import numpy as np

>>> import numpy.matlib

# Eg : one dimensional
>>> a=np.array([1,2,3,4])

>>> print("one dim")
one dim

>>> print(a)
[1 2 3 4]

>>> print(type(a))
<type 'numpy.ndarray'>

#more than one dimension
>>> np.array([1,2,3],dtype=complex)
array([1.+0.j, 2.+0.j, 3.+0.j])


#using ndim
>>> b=np.array([1,2,3,4,5],ndmin=2)

>>> print("Two dimensional")
Two dimensional
>>> print(b.ndim)
2

>>> print(b.shape)
(1, 5)

>>> print(b)
[[1 2 3 4 5]]
>>>




#dtype:
>>> np.array([1,2,3],dtype=complex)
```

array([1.+0.j, 2.+0.j, 3.+0.j])
>>>

Numpy Attributes :

ndarray.ndim------

>>>import numpy as np

>>> import numpy.matlib

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

>>> arrey.ndim

2

>>> print(arrey)

[[1 2 3]

 [4 5 6]]

ndarray.shape------

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

>>> print(array)

[[1 2 3]

 [4 5 6]]

>>> print(array.shape)

(2, 3)

#resize ndarray

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

>>> array.shape=(3,2)

>>> print(array)

[[1 2]

 [3 4]

 [5 6]]

#Resize: NumPy also provides a reshape function to resize an array.

>>> barray=array.reshape(2,3)

```
>>> print(barray)
[[1 2 3]
 [4 5 6]]
```

ndarray.size :

```
>>> array.size
6
```

ndarray.dtype------

```
 >>>array.dtype
dtype('int64')
```

ndarray.iteamsize-----

```
#ax = np.array([1,2,3,4,5], dtype = np.int16)
>>> ax=np.array([1,2,3], dtype=np.float64)
>>> ax.itemsize
8
```

ndarray.data------

```
>>> ax.data
<read-write buffer for 0x7fe76ab2ec10, size 24, offset 0 at 0x7fe76ab30d70>
>>>
```

Data types,Array creation, Numeric Ranges,Indexing and slicing.

dtype------

syntax − **numpy.dtype(object, align, copy)**

Object − To be converted to data type object
Align − If true, adds padding to the field to make it similar to C-struct
Copy − Makes a new copy of dtype object. If false, the result is reference to builtin data type object.

```
>>> dt=np.dtype(np.int64)
>>> dt
dtype('int64')
```

```
>>> dt=np.dtype(np.float64)
>>> dt
dtype('float64')
```

Array creation-------

numpy.empty---

Syntax: numpy.empty(shape, dtype = float, order = 'C')

Shape : Shape of an empty array in int or tuple of int
Dtype : Desired output data type. Optional
Order :'C' for C-style row-major array, 'F' for FORTRAN style column-major array

```
>>> np.empty([3,3], dtype=int)
array([[140653566028760,        18671072, 140653333071504],
       [140653332580064, 140653332570624, 140653332580624],
       [140653570155784,               0, 140653570364688]])
>>>
>>> np.empty([3,3], dtype=float)
array([[0.00000000e+000, 0.00000000e+000, 6.94919798e-310],
       [6.94919796e-310, 6.94919796e-310, 6.94919796e-310],
       [4.23829199e-106, 0.00000000e+000, 6.94920971e-310]])
```

numpy.zeros---

Syntax : numpy.zeros(shape, dtype = float, order = 'F')

```
>>> print(np.zeros(5))
[0. 0. 0. 0. 0.]

>>> np.zeros((3,3,))
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

numpy.ones---

Syntax : numpy.ones(shape, dtype = None, order = 'C')

```
>>> np.ones(5)
array([1., 1., 1., 1., 1.])

>>> np.ones([2,2],dtype=int)
array([[1, 1],
       [1, 1]])
>>>
```

Note: zeros_like,ones_like, empty_like arange,fromfunction, fromfile

numpy.asarray------

syntax : numpy.asarray(a, dtype = None, order = None)

```
>>> x=[1,2,3]
>>> a=np.asarray(x)
>>> print(a)
[1 2 3]



>>> print(type(a))
<type 'numpy.ndarray'>

>>> a.shape
```

```
(3,)
```

Numeric ranges

syntax: numpy.arange(start, stop, step, dtype)

```
>>> np.arange(5,9,2)
array([5, 7])
```

numpy.linspace

syntax: numpy.linspace(start, stop, num, endpoint, retstep, dtype)

retstep : If true, returns samples and step between the consecutive numbers.

endpoint : True by default, hence the stop value is included in the sequence. If false, it is not included

```
>>> np.linspace(10,20,num=10,endpoint=True,retstep=True)
(array([10.        , 11.11111111, 12.22222222, 13.33333333, 14.44444444,
       15.55555556, 16.66666667, 17.77777778, 18.88888889, 20.        ]),
1.1111111111111112)
```

numpy.logspace

syntax : numpy.logscale(start, stop, num, endpoint, base, dtype)

```
>>> np.logspace(1.0,2.0, num=10)
array([ 10.        ,  12.91549665,  16.68100537,  21.5443469 ,
        27.82559402,  35.93813664,  46.41588834,  59.94842503,
        77.42636827, 100.        ])
>>>

>>> np.logspace(1.0,2.0, num=10,base=6)
array([ 6.        ,  7.32170962,  8.93457195, 10.90272356, 13.30442932,
       16.23519468, 19.81156349, 24.17575249, 29.50130657, 36.        ])
>>>
```

resize changes the shape and size of array in-place.

```
>>> o=np.linspace(0,6,10)
>>> print(o)
[0.         0.66666667 1.33333333 2.         2.66666667 3.33333333
 4.         4.66666667 5.33333333 6.        ]
>>> o.resize(3,3)



>>> o
array([[0.        , 0.66666667, 1.33333333],
       [2.        , 2.66666667, 3.33333333],
       [4.        , 4.66666667, 5.33333333]])
>>>
```

eye returns a 2-D array with ones on the diagonal and zeros elsewhere.

```
>>> np.eye(2)
array([[1., 0.],
       [0., 1.]])
>>>
>>> y=[1,2,3]
>>> np.diag(y)
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
>>>
```

Create an array using repeating list (pythonic way)

```
>>> np.repeat([1,2,3],4)
array([1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3])
>>>
>>> p=np.ones([2,3], int)
>>> p
array([[1, 1, 1],
```

```
       [1, 1, 1]])
>>>
#vstack to stack arrays in sequence vertically (row wise).
>>> np.vstack([p,2*p])
array([[1, 1, 1],
       [1, 1, 1],
       [2, 2, 2],
       [2, 2, 2]])
#hstack to stack arrays in sequence horizontally (column wise)

>>> np.hstack([p,2*p])
array([[1, 1, 1, 2, 2, 2],
       [1, 1, 1, 2, 2, 2]])
>>>
```

Indexing / Slicing

```
>>> s=np.arange(16)*6
>>> s
array([ 0,  6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90])
>>>
#indexing
>>> s[0],s[4],s[-1]
(0, 24, 90)
```

To indicate a range. array[start:stop] Leaving start or stop empty will default to the beginning/end of the array.

```
>>> s[1:6]
array([ 6, 12, 18, 24, 30])
>>>
#Use negatives to count from the back.

>>> s[-6:]
```

array([60, 66, 72, 78, 84, 90])

>>>

```
#can be used to indicate step-size. array[start:stop:stepsize]
#Here we are starting 5th element from the end, and counting backwards by 2
until the beginning of the array is reached.
```

>>> s[6::4]

array([36, 60, 84])

>>>

```
#Let's look at a multidimensional array.
```

>>> m=np.arange(36)

>>> m.resize((6,6))

>>> m

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]])

>>>

```
#Use bracket notation to slice: array[row, column]
```

>>> m[2,2]

14

```
#to select a range of rows or columns
```

>>> m[3,3:]

array([21, 22, 23])

>>>

```
#We can also perform conditional indexing. Here we are selecting values from the array that are greater than 30.

>>> m[m>30]
array([31, 32, 33, 34, 35])
>>>

#Here we are assigning all values in the array that are greater than 30 to the value of 30

>>> m[m>30]=30
>>> m
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, 30, 30, 30, 30, 30]])
>>>
>>> x=np.arange(10)
>>> print(x)
[0 1 2 3 4 5 6 7 8 9]
>>> s=slice(2,7,2)
>>> print("Done", x[s])
('Done', array([2, 4, 6]))
```

Topics : Math functions, Basic operations, Statistical Functions, Copies & Views, Broadcasting, Iterating Over Array, ix() function

Trigonometric Functions:

NumPy has standard trigonometric functions which return trigonometric ratios for a given angle in radians.
np.sin()
np.cos()
np.tan()

arcsin, arcos, and arctan functions return the trigonometric inverse of sin, cos, and tan of the given angle.

>>> print(np.sin(0))

```
0.0
>>> a = np.array([0,30,45,60,90])
>>>
>>> print ('Sine of different angles:')
Sine of different angles:
>>>

>>> print (np.sin(a*np.pi/90))
[0.00000000e+00 8.66025404e-01 1.00000000e+00 8.66025404e-01
 1.22464680e-16]

>>> print (np.sin(a*np.pi/180))
[0.         0.5        0.70710678 0.8660254  1.        ]
>>>

>>>
>>> print (np.sin(a*np.pi/270))
[0.         0.34202014 0.5        0.64278761 0.8660254 ]
>>>
>>> print (np.sin(a*np.pi/360))
[0.         0.25881905 0.38268343 0.5        0.70710678]
>>>
>>> print (np.cos(a*np.pi/90) )
[ 1.000000e+00  5.000000e-01  6.123234e-17 -5.000000e-01 -1.000000e+00]
>>>
>>> print (np.cos(a*np.pi/180) )
[1.00000000e+00 8.66025404e-01 7.07106781e-01 5.00000000e-01
 6.12323400e-17]
>>>
>>> print (np.cos(a*np.pi/270) )
[1.         0.93969262 0.8660254  0.76604444 0.5       ]
>>>
>>> print (np.cos(a*np.pi/360) )
[1.         0.96592583 0.92387953 0.8660254  0.70710678]
```

```
>>>
>>>
>>> print (np.tan(a*np.pi/90))
[ 0.00000000e+00  1.73205081e+00  1.63312394e+16 -1.73205081e+00
 -1.22464680e-16]
>>>
>>> print (np.tan(a*np.pi/180))
[0.00000000e+00 5.77350269e-01 1.00000000e+00 1.73205081e+00
 1.63312394e+16]
>>>
>>> print (np.tan(a*np.pi/270))
[0.         0.36397023 0.57735027 0.83909963 1.73205081]
>>>
>>> print (np.tan(a*np.pi/360))
[0.         0.26794919 0.41421356 0.57735027 1.        ]
>>>
```

```
#inverse tri
```

```
>>>
>>> import numpy as np
>>>
>>> import numpy.matlib
>>>
>>> a = np.array([0,30,45,60,90])
>>>
>>> sin = np.sin(a*np.pi/180)
>>> print( sin )
[0.         0.5        0.70710678 0.8660254  1.        ]
>>>
```

```
>>> print ('Compute sine inverse of angles. Returned values are in radians.')
Compute sine inverse of angles. Returned values are in radians.
>>>
>>> inv = np.arcsin(sin)
>>>
>>> print (inv )
[0.         0.52359878 0.78539816 1.04719755 1.57079633]
>>>
>>> print( 'Check result by converting to degrees:' )
Check result by converting to degrees:
>>>
>>> print (np.degrees(inv))
[ 0. 30. 45. 60. 90.]
>>>
>>>
>>> sin = np.sin(a*np.pi/360)
>>> print( sin )
[0.         0.25881905 0.38268343 0.5        0.70710678]
>>>
>>> print ('Compute sine inverse of angles. Returned values are in radians.')
Compute sine inverse of angles. Returned values are in radians.
>>>
>>> inv = np.arcsin(sin)
>>> print (inv )
[0.         0.26179939 0.39269908 0.52359878 0.78539816]
>>>
>>> print( 'Check result by converting to degrees:' )
Check result by converting to degrees:
>>>
>>> print (np.degrees(inv))
[ 0.  15.  22.5 30.  45. ]
>>>
>>>
>>>
```

```
>>> print ('arccos and arctan functions behave similarly:' )
arccos and arctan functions behave similarly:
>>>
>>> cos = np.cos(a*np.pi/180)
>>> print (cos)
[1.00000000e+00 8.66025404e-01 7.07106781e-01 5.00000000e-01
 6.12323400e-17]
>>>
>>> print ('Inverse of cos:')
Inverse of cos:
>>>
>>> inv = np.arccos(cos)
>>> print (inv)
[0.         0.52359878 0.78539816 1.04719755 1.57079633]
>>>
>>> print ('In degrees:')
In degrees:
>>>
>>> print (np.degrees(inv))
[ 0. 30. 45. 60. 90.]
>>>
>>> print ('arccos and arctan functions behave similarly:' )
arccos and arctan functions behave similarly:
>>>
>>> cos = np.cos(a*np.pi/360)
>>> print (cos)
[1.         0.96592583 0.92387953 0.8660254  0.70710678]
>>>
>>> print ('Inverse of cos:')
Inverse of cos:
>>>
>>> inv = np.arccos(cos)
>>> print (inv)
[0.         0.26179939 0.39269908 0.52359878 0.78539816]
```

```
>>>
>>> print ('In degrees:')
In degrees:
>>>
>>> print (np.degrees(inv))
[ 0.  15.  22.5 30.  45. ]
>>>
>>>
>>> print ('Tan function:' )
Tan function:
>>>
>>> tan = np.tan(a*np.pi/180)
>>> print (tan)
[0.00000000e+00 5.77350269e-01 1.00000000e+00 1.73205081e+00
 1.63312394e+16]
>>>
>>> print ('Inverse of tan:')
Inverse of tan:
>>>
>>> inv = np.arctan(tan)
>>> print (inv)
[0.         0.52359878 0.78539816 1.04719755 1.57079633]
>>>
>>> print ('In degrees:' )
In degrees:
>>>
>>> print (np.degrees(inv))
[ 0. 30. 45. 60. 90.]
>>>
>>> print (np.degrees(inv))
[ 0. 30. 45. 60. 90.]
>>>
>>>
>>> print ('Tan function:' )
```

Tan function:

```
>>>
>>> tan = np.tan(a*np.pi/360)
>>> print (tan)
[0.         0.26794919 0.41421356 0.57735027 1.        ]
>>>
>>> print ('Inverse of tan:')
Inverse of tan:
>>>
>>> inv = np.arctan(tan)
>>> print (inv)
[0.         0.26179939 0.39269908 0.52359878 0.78539816]
>>>
>>> print ('In degrees:' )
In degrees:
>>> print (np.degrees(inv))
[ 0.  15.  22.5 30.  45. ]
>>>
```

numpy.around()------

syntax : numpy.around(a,decimals)

```
#round off
>>> a = np.array([1.0,5.55, 123, 0.567, 25.532])
>>> print ('Original array:')
Original array:
>>>
>>> print (a )
[  1.      5.55  123.      0.567  25.532]
>>>
>>> print ('After rounding:')
After rounding:
>>>
>>> print (np.around(a))
```

```
[  1.   6. 123.   1.  26.]
>>>
>>> print (np.around(a, decimals = 1))
[  1.    5.6 123.    0.6  25.5]
>>>
```

numpy.floor()------

```
>>> a = np.array([-1.7, 1.5, -0.2, 0.6, 10])
>>>
>>> print ('array:')
array:
>>>
>>> print (a)
[-1.7  1.5 -0.2  0.6 10. ]
>>>
>>> print ('The modified array:')
The modified array:
#returns largest intgres
>>>
>>> print (np.floor(a))
[-2.  1. -1.  0. 10.]
>>>
#returns lowest intgers
>>> print (np.ceil(a))
[-1.  2. -0.  1. 10.]
>>>
```

Basic operations:

Input arrays for performing arithmetic operations such as add(), subtract(), multiply(), and divide() must be either of the same shape or should conform to array broadcasting rules. Use +, -, *, / and ** to perform element wise addition, subtraction, multiplication, division and power.

```
>>> x=np.array([1,2,3])
>>> y=np.array([4,5,6])
>>>
>>> print(x + y) # elementwise addition     [1 2 3] + [4 5 6] = [5  7  9]
[5 7 9]
>>>
>>> print(x - y) # elementwise subtraction  [1 2 3] - [4 5 6] = [-3 -3 -3]
[-3 -3 -3]
>>>
>>>
>>>
>>> print(x * y) # elementwise multiplication  [1 2 3] * [4 5 6] = [4  10  18]
[ 4 10 18]
>>>
>>> print(x / y) # elementwise divison        [1 2 3] / [4 5 6] = [0.25  0.4  0.5]
[0 0 0]
>>>
>>>
>>> print(x**2) # elementwise power  [1 2 3] ^2 =  [1 4 9]
[1 4 9]
>>>
>>>
>>> a = np.arange(9, dtype = np.float).reshape(3,3)
>>>
>>> print ('First array:')
First array:
>>>
>>> print (a )
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
>>>
>>> print ('Second array:' )
```

```
Second array:
>>>
>>> b = np.array([10,10,10])
>>>
>>> print (b )
[10 10 10]
>>>
>>> print ('Add the two arrays:')
Add the two arrays:
>>>
>>> print (np.add(a,b))
[[10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]]
>>>
>>> print ('Subtract the two arrays:')
Subtract the two arrays:
>>>
>>> print (np.subtract(a,b))
[[-10.  -9.  -8.]
 [ -7.  -6.  -5.]
 [ -4.  -3.  -2.]]
>>>
>>> print ('Multiply the two arrays:')
Multiply the two arrays:
>>>
>>> print (np.multiply(a,b))
[[ 0. 10. 20.]
 [30. 40. 50.]
 [60. 70. 80.]]
>>>
>>> print ('Divide the two arrays:')
Divide the two arrays:
```

```
>>>
>>> print (np.divide(a,b))
[[0.  0.1 0.2]
 [0.3 0.4 0.5]
 [0.6 0.7 0.8]]
>>>
```

Statistical Functions:------

```
>>> a = np.array([-4, -2, 1, 3, 5])
>>> a.sum()
3
>>>
>>> a.max()
5
>>>
>>> a.min()
-4
>>>
>>> np.average(a)
0.6
>>>
>>> a.mean()
0.6
>>>
>>> a.std() #Standard deviation is the square root of the average of squared deviations from mean
3.2619012860600183
>>>
>>>
>>> np.var([1,2,3,4])
1.25
>>>
>>> a.argmax()
4
```

```
>>>

>>> a.argmin()

0

>>>
```

Copies & Views :

#no copy

```
>>> a = np.arange(6)
>>>
>>> print ('Our array is:' )
Our array is:
>>>
>>> print (a )
[0 1 2 3 4 5]
>>>
>>> print ('Applying id() function:')
Applying id() function:
>>>
>>> print (id(a))
140653333007216
>>>
>>> print ('a is assigned to b:' )
a is assigned to b:
>>>
>>> b = a
>>>
>>> print (b)
[0 1 2 3 4 5]
>>>
>>> print ('b has same id():')
b has same id():
>>>
>>> print (id(b))
140653333007216
>>>
>>> print ('Change shape of b:')
Change shape of b:
>>>
>>> b.shape = 3,2
>>>


>>> print (b)
[[0 1]
 [2 3]
 [4 5]]
>>>
>>> print ('Shape of a also gets changed:')
Shape of a also gets changed:
>>>
>>> print (a)
[[0 1]
 [2 3]
 [4 5]]
>>>
```

#view

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

>>>
```

#print 'Array a:'

```
>>> print (a )
```

```
[1 2 3 4]
>>>
>>> print(id(a))
140653333006576
>>>
#Create view of a:
>>> b = a.view()
>>>
>>> print( b )
[1 2 3 4]
>>>
>>> b.shape=(2,2)
>>>
>>> print(id(b))
140653333007136
>>>

>>> print (b is a)
False
>>>
>>> print(b.shape)
(2, 2)
>>>
>>> print(a.shape)
(4,)
>>>

#copy
>>> a = np.array([[10,10], [2,3], [4,5]])
>>>
>>> print ('Array a is:')
Array a is:
```

```
>>>
>>> print( a)
[[10 10]
 [ 2  3]
 [ 4  5]]
>>>
# 'Create a deep copy of a:'
>>> b = a.copy()
>>>
>>> print ('Array b is:')
Array b is:
>>>
```

```
>>> print (b)
[[10 10]
 [ 2  3]
 [ 4  5]]
>>>
#b does not share any memory of a
>>> print ('Can we write b is a')
Can we write b is a
>>>
>>> print (b is a)
False
```

Broadcasting-------

```
#normal example
>>> a = np.array([1,2,3,4])
>>> b = np.array([10,20,30,40])
```

```
>>>
>>> print(a.shape)
(4,)
>>>
>>> print(b.shape)
(4,)
>>>
>>> c = a * b
>>>
>>> print (c)
[ 10  40  90 160]
>>>
```

```
#Broadcasting
>>> x = np.arange(4)
>>> y = np.ones(5)
>>> xb=x.reshape(4,1)
>>>
>>> print(xb)
[[0]
 [1]
 [2]
 [3]]
>>>
#bd
>>> print(xb + y)
[[1. 1. 1. 1. 1.]
 [2. 2. 2. 2. 2.]
 [3. 3. 3. 3. 3.]
 [4. 4. 4. 4. 4.]]
>>>
>>> (xb + y).shape
```

(4, 5)

>>>

Note : If the dimensions of two arrays are dissimilar, element-to-element operations are not possible. However, operations on arrays of non-similar shapes is still possible in NumPy, because of the broadcasting capability.

```
#Matrix operations
>>> z = np.array([y, y**2])
>>>
>>> print(len(z)) # number of rows of array
2
```

Let's look at transposing arrays. Transposing permutes the dimensions of the array.

```
>>> y=np.arange(5)
>>>
>>> z = np.array([y, y ** 2])
>>>
>>> z
array([[ 0,  1,  2,  3,  4],
       [ 0,  1,  4,  9, 16]])
>>>

#The shape of array z is (2,3) before transposing.
>>> z.shape
(2, 5)
>>>
>>> z.T
array([[ 0,  0],
       [ 1,  1],
       [ 2,  4],
       [ 3,  9],
       [ 4, 16]])
```

>>>

Dot Product:

[x1,x2,x2]clo[y1,y2,y3] = x1y1+x2xy2+x3y3

```
>>> x=np.array([1,2,3])
>>> y=np.array([4,5,6])
>>> x.dot(y) # dot product  1*4 + 2*5 + 3*6
32
```

Iterating Over Arrays

create a new 4 by 3 array of random numbers 0-9

```
>>> tp = np.random.randint(0, 10, (4,3))
>>> tp
array([[3, 4, 4],
       [6, 6, 2],
       [6, 6, 2],
       [8, 5, 9]])
>>>
```

```
#Iterate by row:
>>> for row in tp:
...     print(row)
...
[3 4 4]
[6 6 2]
[6 6 2]
[8 5 9]
```

```
#Iterate by index:
>>> for i, row in enumerate(tp):
...     print('row', i, 'is', row)
...
```

```
('row', 0, 'is', array([3, 4, 4]))
('row', 1, 'is', array([6, 6, 2]))
('row', 2, 'is', array([6, 6, 2]))
('row', 3, 'is', array([8, 5, 9]))
>>>
```

```
#Use zip to iterate over multiple iterables.
```

```
>>> tp2=tp*2
>>> tp2
array([[ 6,  8,  8],
       [12, 12,  4],
       [12, 12,  4],
       [16, 10, 18]])
>>>

>>> for i, j in zip(tp, tp2):
...     print(i,'+',j,'=',i+j)
...
(array([3, 4, 4]), '+', array([6, 8, 8]), '=', array([ 9, 12, 12]))
(array([6, 6, 2]), '+', array([12, 12,  4]), '=', array([18, 18,  6]))
(array([6, 6, 2]), '+', array([12, 12,  4]), '=', array([18, 18,  6]))
(array([8, 5, 9]), '+', array([16, 10, 18]), '=', array([24, 15, 27]))
>>>

>>> a = np.arange(0,60,5)
>>> a = a.reshape(3,4)
>>>
>>> print ('Original array is:')
Original array is:
>>>
>>> print (a)
[[ 0  5 10 15]
```

```
 [20 25 30 35]
 [40 45 50 55]]
>>>
>>> print ('Modified array is:')
Modified array is:
>>>
>>> for x in np.nditer(a):
...    print (x)
...
0
5
10
15
20
25
30
35
40
45
50
55
>>>
```

ix_() function:

```
>>> a = np.array([2,3,4,5])
>>> b = np.array([8,5,4])
>>> c = np.array([5,4,6,8,3])
>>>
>>> ax,bx,cx = np.ix_(a,b,c)
>>> result = ax+bx*cx
>>> result
array([[[42, 34, 50, 66, 26],
        [27, 22, 32, 42, 17],
```

```
  [22, 18, 26, 34, 14]],

 [[43, 35, 51, 67, 27],
  [28, 23, 33, 43, 18],
  [23, 19, 27, 35, 15]],

 [[44, 36, 52, 68, 28],
  [29, 24, 34, 44, 19],
  [24, 20, 28, 36, 16]],

 [[45, 37, 53, 69, 29],
  [30, 25, 35, 45, 20],
  [25, 21, 29, 37, 17]]])
>>> result[3,2,4]
17
>>> a[3]+b[2]*c[4]
17
>>>
```

Topics : Matlib subpackage, matrix, linear algebra method, matplotlib using numpy.

```
#NumPy package contains a Matrix library numpy.matlib.

>>> import numpy.matlib

#matlib.empty()
#numpy.matlib.empty(shape, dtype, order)
>>> print (np.matlib.empty((2,2)))
[[4.94e-323 1.98e-322]
 [4.45e-322 7.91e-322]]

ones
>>> print (np.matlib.ones((2,2)))
[[1. 1.]
 [1. 1.]]
```

```
#random

>>> print (np.matlib.rand(3,3))

[[0.21960921 0.76984359 0.08066243]

 [0.15064553 0.35117811 0.32533956]

 [0.27112169 0.84364676 0.42206457]]

>>>

#This function returns the matrix filled with zeros.
#numpy.matlib.zeros()
>>> print (np.matlib.zeros((2,2)))

[[0. 0.]

 [0. 0.]]

>>>

#numpy.matlib.eye()
#This function returns a matrix with 1 along the diagonal elements and the
zeros elsewhere. The function takes the following parameters.
#numpy.matlib.eye(n, M,k, dtype)
>>> print (np.matlib.eye(n = 3, M = 3, k = 1, dtype = float))

[[0. 1. 0.]

 [0. 0. 1.]

 [0. 0. 0.]]

>>>

#numpy.matlib.identity()
#The numpy.matlib.identity() function returns the Identity matrix of the
given size.
#An identity matrix is a square matrix with all diagonal elements as 1.
>>> np.matlib.identity(3)

matrix([[1., 0., 0.],

        [0., 1., 0.],

        [0., 0., 1.]])

>>>



#creation matrix

>>> i = np.matrix('1,2,3,4')

>>>



>>> print(i)
```

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

```
#array to matrix
```

```
>>> list=[1,2,3,4]
>>>
>>> k = np.asmatrix (list)
>>>
>>> print(k)
[[1 2 3 4]]
>>>
>>> print(type(k))
<class 'numpy.matrixlib.defmatrix.matrix'>
>>>
```

NumPy package contains **numpy.linalg module** that provides all the functionality required for linear algebra

```
#det
```

```
>>> b = np.array([[6,1,1], [4, -2, 5], [2,8,7]])
>>>
>>> print (b)
[[ 6  1  1]
 [ 4 -2  5]
 [ 2  8  7]]
>>>
>>>
>>> print (np.linalg.det(b))
-306.0
>>>
>>> print (6*(-2*7 - 5*8) - 1*(4*7 - 5*2) + 1*(4*8 - -2*2))
-306
```

```
#dot
#Dot product of the two arrays
```

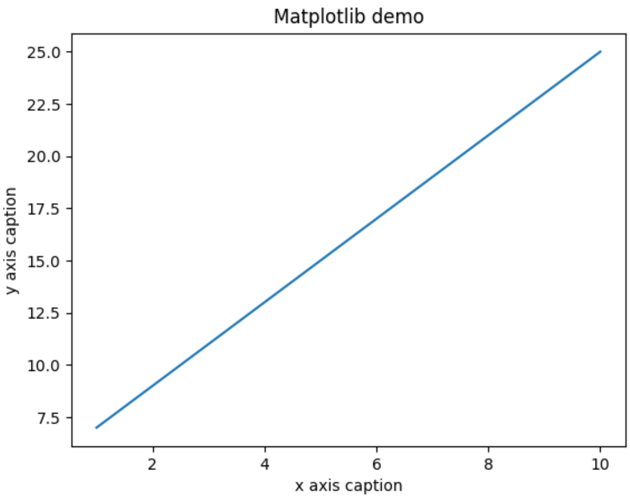
```
#vdot
#Dot product of the two vectors
#linear
>>> dou = np.array([[1,2],[3,4]])

>>> bou = np.array([[11,12],[13,14]])

>>>

>>> print(np.dot(dou,bou)) #[[1*11+2*13, 1*12+2*14],[3*11+4*13, 3*12+4*14]]

[[37 40]

 [85 92]]

>>>

>>>

>>> print(np.vdot(dou,bou)) #1*11 + 2*12 + 3*13 + 4*14

130


#Solve the system of equations 3 * x0 + x1 = 9 and x0 + 2 * x1 = 8:

>>> al = np.array([[3,1], [1,2]])

>>> bl = np.array([9,8])

>>> x = np.linalg.solve(al, bl)

>>>

>>> print(x)

[2. 3.]

>>>
```

```
>>> a = np.array([[1,1,1],[0,2,5],[2,5,-1]])

>>>

#'Array a
```

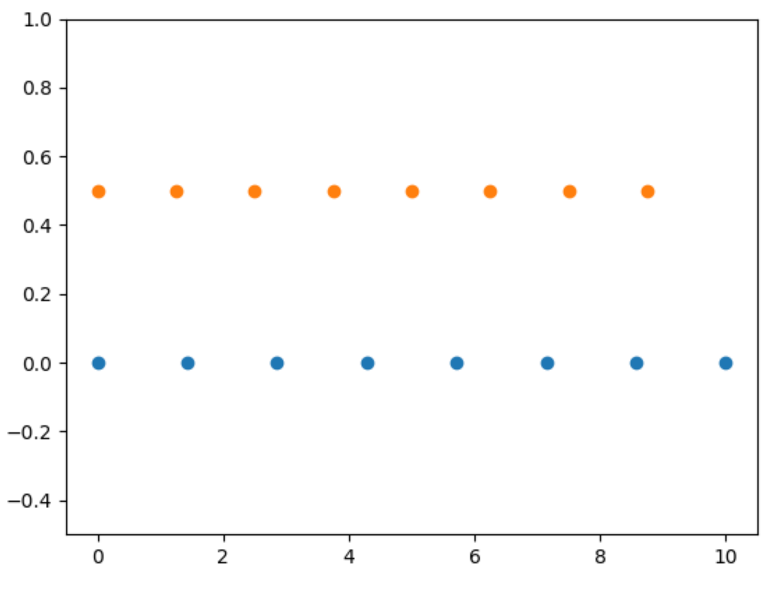
```
>>> print (a)
[[ 1  1  1]
 [ 0  2  5]
 [ 2  5 -1]]
>>>
>>> ainv = np.linalg.inv(a)
>>>
>>> print(ainv)
[[ 1.28571429 -0.28571429 -0.14285714]
 [-0.47619048  0.14285714  0.23809524]
 [ 0.19047619  0.14285714 -0.0952381 ]]
>>>
```

Using Matplotlib with numpy

```
>>> from matplotlib import pyplot as plt

>>>
>>> x = np.arange(1,11)
>>>
>>> y = 2 * x + 5
>>>
>>> plt.title("Matplotlib demo")
Text(0.5,1,'Matplotlib demo')
>>>
>>> plt.xlabel("x axis caption")
Text(0.5,0,'x axis caption')
>>>
>>> plt.ylabel("y axis caption")
Text(0,0.5,'y axis caption')
>>>
>>> plt.plot(x,y)
[<matplotlib.lines.Line2D object at 0x7fec5a13db50>]
>>>
>>> plt.show()
```



```
>>> N = 8
>>>
>>> y = np.zeros(N)
>>>
>>> y
array([0., 0., 0., 0., 0., 0., 0., 0.])
>>>

>>> x1 = np.linspace(0, 10, N, endpoint=True)
>>>
>>> x2 = np.linspace(0, 10, N, endpoint=False)
>>>
>>> plt.plot(x1, y, 'o')
[<matplotlib.lines.Line2D object at 0x7fec600a8710>]
>>>
>>> plt.plot(x2, y + 0.5, 'o')
[<matplotlib.lines.Line2D object at 0x7fec601253d0>]
>>>
>>> plt.ylim([-0.5, 1])
(-0.5, 1)
>>>
>>> plt.show()
```



```
>>> import time
>>> import numpy as np
>>>
>>> size_of_vec = 100000
>>>
>>> def pure_python_version():
...     t1 = time.time()
...     X = range(size_of_vec)
...     Y = range(size_of_vec)
...     Z = []
...     for i in range(len(X)):
...         Z.append(X[i] + Y[i])
...     return time.time() - t1
...
>>>
>>> def numpy_version():
...     t1 = time.time()
...     X = np.arange(size_of_vec)
...     Y = np.arange(size_of_vec)
...     Z = X + Y
...     return time.time() - t1
...
>>>
>>> t1 = pure_python_version()
>>> t2 = numpy_version()
```

```
>>> print(t1, t2)
(0.020282983779907227, 0.0008230209350585938)
>>>
```