```
import numpy as np
```

```
a = np.array([2, 6, 5])
b = np.array([1, 2, 3])
print ("Divide", np.divide(a, b), np.divide(b, a))
```

```
Divide [2.         3.         1.66666667] [0.5        0.33333333 0.6       ]
```

```
print(a/b)
```

```
[2.         3.         1.66666667]
```

```
print(a//b)
```

```
[2 3 1]
```

```
print(a*b)
```

```
[ 2 12 15]
```

```
a=np.array([[1,2],[3,4]])
b=np.array([[5,6],[7,8]])
print(a,'\n',b)
```

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

```
print(a+b)
print('\n')
print(a*b)
print('\n')
print(a/b)
print('\n')
print(a//b)
print('\n')
print(np.multiply(a,b))
```

```
[[ 6  8]
 [10 12]]


[[ 5 12]
 [21 32]]


[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

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


[[ 5 12]
 [21 32]]
```

Modulo operation

```
a = np.arange(0, 8)
print(a)
print ('\n')
print ("Remainder", np.remainder(a, 3))
print ('\n')
print ("Remainder", np.remainder(a, np.max(a)))
```

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


Remainder [0 1 2 0 1 2 0 1]


Remainder [0 1 2 3 4 5 6 0]
```

```
print ("Mod", np.mod(a, 3))
```

```
Mod [0 1 2 0 1 2 0 1]
```

```
print ("% operator", a % 3)
```

```
% operator [0 1 2 0 1 2 0 1]
```

```
a = np.arange(-4, 4)
print(a)
print ('\n')
print ("Remainder", np.remainder(a, 3))
```

```
[-4 -3 -2 -1  0  1  2  3]


Remainder [2 0 1 2 0 1 2 0]
```

```
print ("Fmod", np.fmod(a, 2))
```

```
Fmod [ 0 -1  0 -1  0  1  0  1]
```

Changing Shape

```
a = np.arange(8)
print ('The original array:',a)
print ('\n')
b = a.reshape(4,2)
print ('The modified array:\n',b)

```

```
The original array: [0 1 2 3 4 5 6 7]


The modified array:
 [[0 1]
 [2 3]
 [4 5]
 [6 7]]
```

```
print (a.flat)
```

```
<numpy.flatiter object at 0x205f700>
```

flatten(order) order

'C'– row major (default. 'F': column major 'A': flatten in column-major order, if a is Fortran contiguous in memory, row-major order otherwise 'K': flatten a in the order the elements occur in the memory

```
import numpy as np
a = np.arange(8).reshape(2,4)
print ('The original array is:\n' ,a)

print ('\n' )
# default is column-major

print ('The flattened array is:' )
b=a.flatten()
print(b)

print ('The flattened array in F-style ordering:' )

b=a.flatten(order = 'F')
print(b)
```

```
The original array is:
 [[0 1 2 3]
 [4 5 6 7]]


The flattened array is:
[0 1 2 3 4 5 6 7]
The flattened array in F-style ordering:
[0 4 1 5 2 6 3 7]
```

numpy.ravel(a, order)

This function returns a flattened one-dimensional array. A copy is made only if needed. The returned array will have the same type as that of the input array. The function takes one parameter.

```
import numpy as np
a = np.arange(8).reshape(2,4)

print ('The original array is:' )
print (a )
print ('\n')

print ('After applying ravel function:' )
print (a.ravel())
print ('\n' )

print ('Applying ravel function in F-style ordering:' )
print (a.ravel(order = 'F'))
```

```
The original array is:
[[0 1 2 3]
 [4 5 6 7]]


After applying ravel function:
[0 1 2 3 4 5 6 7]


Applying ravel function in F-style ordering:
[0 4 1 5 2 6 3 7]
```

Transpose operation

```
import numpy as np
a = np.arange(15).reshape(3,5)
print ('The original array is:' )
print (a )
print ('\n' )
```

```
The original array is:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

```
print (np.transpose(a))
```

```
[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
```

```
 [ 3  8 13]
 [ 4  9 14]]
```

```
print (a.T)
```

```
[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]
```

```
print (np.rollaxis(a,0) )
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

```
print (np.rollaxis(a,1) )
```

```
[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]
```

```
print (np.swapaxes(a,0,0))
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

```
#numpy.swapaxes(arr, axis1, axis2)
#This function interchanges the two axes of an array.
print (np.swapaxes(a,0,0))
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

```
print (np.swapaxes(a,0,1))
```

```
[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]
```

Dimension change

- Expand

- squeeze

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

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

numpy.expand_dims(arr, axis)

```
y = np.expand_dims(x, axis = 0)
print(y)
print (x.ndim,y.ndim )
```

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

```
y = np.expand_dims(x, axis = 1)
print(y)
print (x.ndim,y.ndim )
```

```
[[[1 2]]

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

numpy.squeeze(arr, axis): removes one-dimensional entry from the shape of the given array.

arr: Input array

axis: int or tuple of int. selects a subset of single dimensional entries in the shape

```
y = np.squeeze(x)
print(y)
```

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

```
print (x.shape, y.shape)
```

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

```
y=np.arange(20).reshape(2,2,5)
print(y)
print(y.shape)
```

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

```
 [[10 11 12 13 14]
  [15 16 17 18 19]]]
(2, 2, 5)
```

```
yy=np.squeeze(y)
print(yy)
print(yy.shape)
```

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

 [[10 11 12 13 14]
  [15 16 17 18 19]]]
(2, 2, 5)
```

NumPy - Matrix Library

- NumPy package contains a Matrix library numpy.matlib.
- This module has functions that return matrices instead of ndarray objects.

```
import numpy.matlib
import numpy as np
a=np.matlib.empty((2,2))
print (a)
print(type(a))
b=np.ones(4)
print(type(b))
```

```
[[2.5e-323 3.0e-323]
 [3.5e-323 4.0e-323]]
<class 'numpy.matrix'>
<class 'numpy.ndarray'>
```

```
a=np.matlib.ones((2,2))
print(a)
```

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

numpy.matlib.eye(n, M,k, dtype) n: The number of rows in the resulting matrix

M: The number of columns, defaults to n

k: Index of diagonal

dtype: Data type of the output

```
b= np.matlib.eye(n = 3, M = 4, k = 0, dtype = float)
print(b)
```

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

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

```
b= np.matlib.eye(n = 4, M = 4, k = 0, dtype = int)
print(b*5)
```

```
[[5 0 0 0]
 [0 5 0 0]
 [0 0 5 0]
 [0 0 0 5]]
```

```
b= np.matlib.eye(n = 4, M = 4, k = 1, dtype = int)
print(b*5)
```

```
[[0 5 0 0]
 [0 0 5 0]
 [0 0 0 5]
 [0 0 0 0]]
```

```
b= np.matlib.eye(n = 4, M = 4, k = 2, dtype = int)
print(b*5)
```

```
[[0 0 5 0]
 [0 0 0 5]
 [0 0 0 0]
 [0 0 0 0]]
```

```
print (np.matlib.identity(4, dtype = float))
```

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

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

```
[[0.78655276 0.04105795 0.39871213]
 [0.77818371 0.69576757 0.54566996]
 [0.2831413  0.29581873 0.83618334]]
```

```
i = np.matrix('1,2;3,4')
print (i)
```

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

```
j = np.asarray(i)
print (j )
```

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

```
k = np.asmatrix (j)
print (k)
```

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

Matrix creation from string

```
A = np.mat('1 2 3; 4 5 6; 7 8 9')
print ("Creation from string", A)
```

```
Creation from string [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
A = np.mat('1 2 3')
print ("Creation from string", A)
print(A.shape)
```

```
Creation from string [[1 2 3]]
(1, 3)
```

```
A = np.mat('1 2 3; 4 5 6; 7 8 9')
print ("Creation from string", A)
print("transpose of A",A.T)
```

```
Creation from string [[1 2 3]
 [4 5 6]
 [7 8 9]]
transpose of A [[1 4 7]
 [2 5 8]
 [3 6 9]]
```

```
a=np.mat('1 2; 3 4')
print ("Inverse A", a.I)
```

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

Linear algebra

- The **numpy.linalg** package contains linear algebra functions.

```
A = np.mat("0 1 2;1 0 3;4 -3 8")
print ("A\n", A)
```

```
A
 [[ 0  1  2]
 [ 1  0  3]
 [ 4 -3  8]]
```

```
print (np.linalg.det(A))
```

```
-2.0
```

```
inverse = np.linalg.inv(A)
print ("inverse of A\n", inverse)
```

```
inverse of A
 [[-4.5  7.  -1.5]
 [-2.   4.  -1. ]
 [ 1.5 -2.   0.5]]
```

```
print ("Check\n", A * inverse)
```

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

```
print ("Check\n", A.T)
```

```
Check
 [[ 0  1  4]
 [ 1  0 -3]
 [ 2  3  8]]
```