import numpy as np

```
import numpy as np
```

Creating a Vector

```
row = np.array([1,2,3,4])
print(row)
```

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

```
column = np.array([[1],
                  [2],
                  [3],
                  [4]])
print(column)
```

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

Creting Matrix (two dimensional array)

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

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

Creting Sparse matrix

```
from scipy import sparse
matrix = np.array([[0,0,0],
                   [0,0,1],
                   [3,0,1]])

sparse_matrix = sparse.csr_matrix(matrix)
print(sparse_matrix)
```

```
  (1, 2)    1
  (2, 0)    3
  (2, 2)    1
```

```
mat=np.array([[0,0,0,0,0,0,0,0],
              [0,1,0,0,0,0,0,0],
              [3,0,0,0,0,0,0,0]])
sparse_mat = sparse.csr_matrix(mat)
print(sparse_mat)
```

```
  (1, 1)   1
  (2, 0)   3
```

Selecting Element

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

```
6
```

```
M[0:3]
```

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

```
M[:5]
```

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

```
M[3:]
```

```
array([4, 5, 6])
```

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

```
5
```

```
matrix[0:1,1:3]
```

```
array([[2, 3]])
```

```
matrix[2:3,:]
```

```
array([[7, 8, 9]])
```

```
matrix[:,0:1]
```

```
array([[1],
       [4],
       [7]])
```

Describe The matrix

Use shape,size,ndim

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

```
matrix.shape  #(Number 0f row and number of column)
```

```
(3, 4)
```

```
matrix.size  #(Row*Column)
```

```
12
```

```
matrix.ndim  #View number of dimension
```

```
2
```

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

```
m.ndim
```

```
3
```

Add any didgit in matrix

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

```
matrix+100
```

```
array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

```
matrix-1
```

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

Finding the Maximum and Minimum Value in the matrix

```
matrix = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]])
m=np.max(matrix)
print(m)
```

```
9
```

```
min=np.min(matrix)
print(min)
```

```
1
```

Finding the minimum and maximum in column and row by using axis

```
np.max(matrix,axis=0) #finding the maximum in in column usnig axis=0
```

```
array([7, 8, 9])
```

```
np.min(matrix,axis=0)
```

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

```
np.max(matrix,axis=1)  #findig the max value in the row by using axis=1
```

```
array([3, 6, 9])
```

```
np.min(matrix,axis=1)
```

```
array([1, 4, 7])
```

Calculating the Averege,Variance and Standard

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

```
mean_matrix=np.mean(matrix)  #Find the mean
print(mean_matrix)
```

```
5.0
```

```
standard_matrix = np.std(matrix) #Find the Standard Deviation
print(standard_matrix)
```

```
2.581988897471611
```

```
var_matrix=np.var(matrix)
var_matrix
```

```
6.666666666666667
```

Find the mean value in the column and row

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

```
np.mean(matrix,axis=0)
```

```
array([4., 5., 6.])
```

```
np.mean(matrix,axis=1)
```

```
array([2., 5., 8.])
```

Reshaping Arrays

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

```
matrix.reshape(2,6)
```

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

```
matrix.reshape(3,4)
```

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

```
matrix.reshape(12)
```

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

```
matrix.reshape(-1,1) #Here is the one usefull argument is -1 means "As many needed column and row"
```

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

```
matrix.reshape(1,-1)
```

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

Transporting a Matrix

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

```
matrix.T  #this function is swaped the matrix element
```

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

```
m=np.array([[1,2,3,4,5,6,7,8]])
m.T
```

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

Flattening a Matrix

you need to transform a matrix into a one dimension matrix

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

```
matrix.flatten()
```

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

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

```
three_dimension.ndim
```

```
3
```

```
one=three_dimension.flatten()
one
```

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

```
one.ndim
```

```
1
```

Finding the Rank of a Matrix

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

```
np.linalg.matrix_rank(matrix)
```

```
2
```

```
three_dimension
```

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

```
np.linalg.matrix_rank(three_dimension)
```

```
array([2])
```

Calculating the Determinant

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

```
np.linalg.det(matrix)
```

```
0.0
```

```
np.linalg.det(three_dimension)
```

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

Getting the Diagonal of a Matrix

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

```
matrix.diagonal()
```

```
array([1, 5, 9])
```

```
matrix.diagonal(offset=1) #Return diagonal one above the main diagonal
```

```
array([2, 6])
```

```
matrix.diagonal(offset=-1)  #Return diagonal one below the main
diagonal
```

```
array([4, 8])
```

Calculating the Trace of a Matrix

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

```
matrix.trace()  #means calculating the diagonal of a matrix
```

```
15
```

```
m=np.array([[2,4,6],
            [7,8,9],
            [1,9,7]])
```

```
m.trace()
```

```
17
```

Finding the Eigenvalue and Eigenvectors

```
matrix = np.array([[1,-1,2],
                   [1,1,6],
                   [3,8,9]])
```

```
eigenvalues,eigenvectors =np.linalg.eig(matrix)
```

```
eigenvalues
```

```
array([13.33584952,  0.88475897, -3.2206085 ])
```

```
eigenvectors
```

```
array([[-0.10854731, -0.96648374, -0.43414013],
       [-0.44191126,  0.20298526, -0.70116895],
       [-0.89046725,  0.15718192,  0.56558329]])
```

```
matrix1 = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]])
```

```
eigenvlaues,eigenvectors = np.linalg.eig(matrix1)
```

```
eigenvalues
```

```
array([ 1.61168440e+01, -1.11684397e+00, -1.30367773e-15])
```

```
eigenvectors
```

```
array([[-0.23197069, -0.78583024,  0.40824829],
       [-0.52532209, -0.08675134, -0.81649658],
       [-0.8186735 ,  0.61232756,  0.40824829]])
```

Calculating the Dot Products

```
vactor_a = np.array([1,2,3])
vector_b = np.array([4,5,6])
```

```
np.dot(vactor_a,vector_b)
```

```
32
```

```
matrix1 = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]])
matrix = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]])
```

```
np.dot(matrix,matrix1)
```

```
array([[ 30,  36,  42],
       [ 66,  81,  96],
       [102, 126, 150]])
```

Adding and Subtracting Matrices

```
matrix1 = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]])

matrix2 = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]])
```

```
np.add(matrix1,matrix2)
```

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

```
np.subtract(matrix1,matrix2)
```

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

We can simply use the "+" and "-" for subtracting and adding

```
matrix1+matrix2
```

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

```
matrix1-matrix2
```

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

Multiplying Matrices

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

n=np.array([[4,5],
            [6,7]])

np.dot(m,n)

array([[16, 19],
       [36, 43]])
```

Another way to multiply matrices using "@" operator

```
m@n

array([[16, 19],
       [36, 43]])
```

If we want to do element-wise multiplication,we can use the "*" operator

```
m

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

n

array([[4, 5],
       [6, 7]])

m*n

array([[ 4, 10],
       [18, 28]])
```

Inverting a Matrix

You want to calculate the inverse of a square matrix

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

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

np.linalg.inv(m)

array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

```
n=np.array([[1,4],
            [2,5]])
```

```
np.linalg.inv(n)
```

```
array([[-1.66666667,  1.33333333],
       [ 0.66666667, -0.33333333]])
```

Genrating Random values

```
np.random.randint(0,5,3)
```

```
array([4, 4, 2])
```

```
np.random.randint(1,9,6)
```

```
array([2, 7, 2, 8, 7, 3])
```

```
np.random.randint(1,11,9).reshape(3,3)
```

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

```
np.random.randint(1,15,12).reshape(4,3)
```

```
array([[ 5,  8, 12],
       [14,  5,  8],
       [ 5, 11, 10],
       [ 1,  3, 11]])
```

```
np.random.seed(1)
np.random.randint(1,5,5)
```

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

```
np.random.seed(1)
np.random.randint(1,9,6).reshape(2,3)
```

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

```
np.random.normal(1.0,2.0,10).reshape(2,5)
```

```
array([[-0.60434568,  0.10224438, -1.21187015, -2.3090309 , -
3.72693721],
       [ 3.2706907 , -1.03402827,  2.27472363, -0.71981321,
4.54521526]])
```

```
np.random.normal(3,5,10).reshape(5,2)
```

```
array([[-2.55181527,  3.90607133],
       [ 5.82172433,  0.16744885],
```

```
       [ 6.64987798,  4.86496895],
       [ 5.66905456,  2.5401335 ],
       [12.56910194,  4.65398565]])
```

Creating a empty array

```
import numpy as np

empty1 = np.empty([4,4],dtype="int")
print(empty1)

[[39353344        0        0        0]
 [       0        0        0        0]
 [       0        0        0        0]
 [       0        0        0        0]]

emp=np.empty([2,2],dtype="int")
print(emp)

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

Creating a full() array

```
full_array = np.full([3,3],45)

full_array

array([[45, 45, 45],
       [45, 45, 45],
       [45, 45, 45]])

full = np.full([4,4],20,dtype="complex")

full

array([[20.+0.j, 20.+0.j, 20.+0.j, 20.+0.j],
       [20.+0.j, 20.+0.j, 20.+0.j, 20.+0.j],
       [20.+0.j, 20.+0.j, 20.+0.j, 20.+0.j],
       [20.+0.j, 20.+0.j, 20.+0.j, 20.+0.j]])
```

Create a Numpy array filled with all ones

```
ones_array = np.ones(3,dtype=int)
ones_array

array([1, 1, 1])

ones = np.ones([3,3])

ones
```

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

```python
ones = np.ones([4,3],dtype="complex")
```

```python
ones
```

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

Remove non-numeric values

```python
matrix = np.array([[1,2,3],
                   [4,np.nan,np.nan],
                   [7,8,9],
                   [np.nan,2,3.0]])
matrix
```

```
array([[ 1.,  2.,  3.],
       [ 4., nan, nan],
       [ 7.,  8.,  9.],
       [nan,  2.,  3.]])
```

```python
matrix[~np.isnan(matrix).any(axis=1)]
```

```
array([[1., 2., 3.],
       [7., 8., 9.]])
```

```python
matrix = np.array([[1,2,3],
                   [4,np.nan,np.nan],
                   [7,8,9],
                   [np.nan,2,3.0],
                   [2,3,4],
                   [np.nan,np.nan,np.nan]])
matrix
```

```
array([[ 1.,  2.,  3.],
       [ 4., nan, nan],
       [ 7.,  8.,  9.],
       [nan,  2.,  3.],
       [ 2.,  3.,  4.],
       [nan, nan, nan]])
```

```python
matrix[~np.isnan(matrix).any(axis=1)]
```

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