Prepared by Asif Bhat

Linear Algebra with Python - Part 1

Linear Algebra

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [2]:
```
v = [3,4]
u = [1,2,3]
```

In [3]:
```
v ,u
```

Out[3]:
```
([3, 4], [1, 2, 3])
```

In [4]:
```
type(v)
```

Out[4]:
```
list
```

In [5]:
```
w = np.array([9,5,7])
```

In [6]:
```
type(w)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
w.shape[0]
```

Out[7]:
```
3
```

In [8]:
```
w.shape
```

Out[8]:
```
(3,)
```

Reading elements from an array

In [9]:
```
a = np.array([7,5,3,9,0,2])
```

In [10]:
```
a[0]
```

Out[10]:
```
7
```

In [11]:
```
a[1:]
```
Out[11]: `array([5, 3, 9, 0, 2])`

In [12]:
```
a[1:4]
```
Out[12]: `array([5, 3, 9])`

In [13]:
```
a[-1]
```
Out[13]: `2`

In [14]:
```
a[-3]
```
Out[14]: `9`

In [15]:
```
a[-6]
```
Out[15]: `7`

In [16]:
```
a[-3:-1]
```
Out[16]: `array([9, 0])`

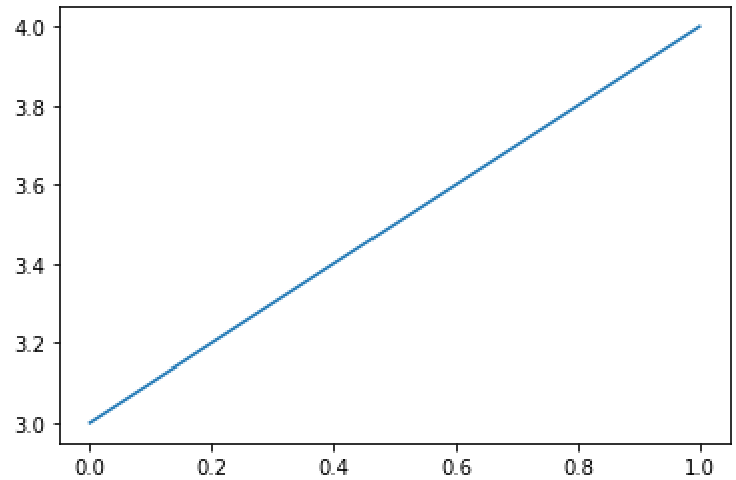
Plotting a Vector

What is vector : https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=1 (https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=1)

In [17]:
```
v = [3,4]
u = [1,2,3]
```

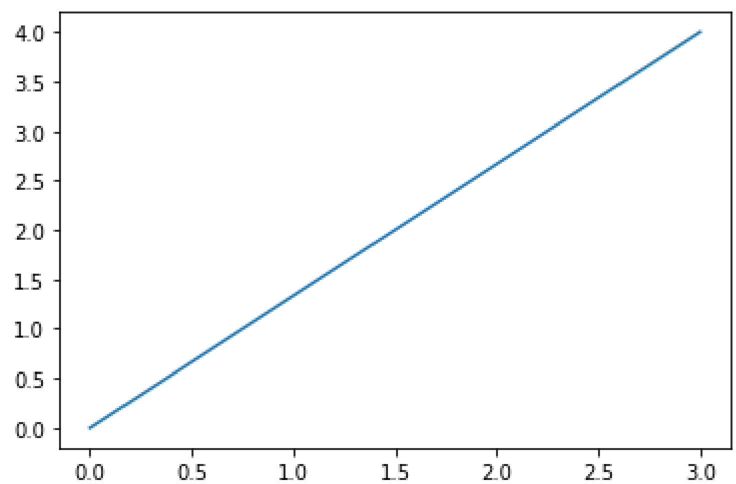
In [18]:
```
plt.plot (v)
```

Out[18]: [<matplotlib.lines.Line2D at 0x7fe638fcaa90>]



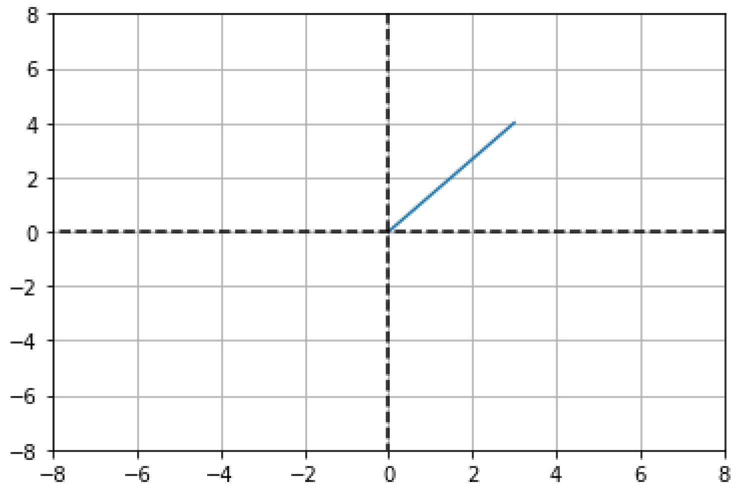
In [19]:
```
plt.plot([0,v[0]] , [0,v[1]])
```

Out[19]: [<matplotlib.lines.Line2D at 0x7fe6489648e0>]



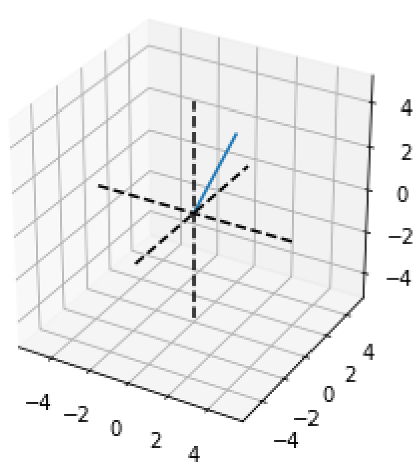
Plot 2D Vector

```
In [20]: plt.plot([0,v[0]] , [0,v[1]])
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.show()
```



Plot the 3D vector

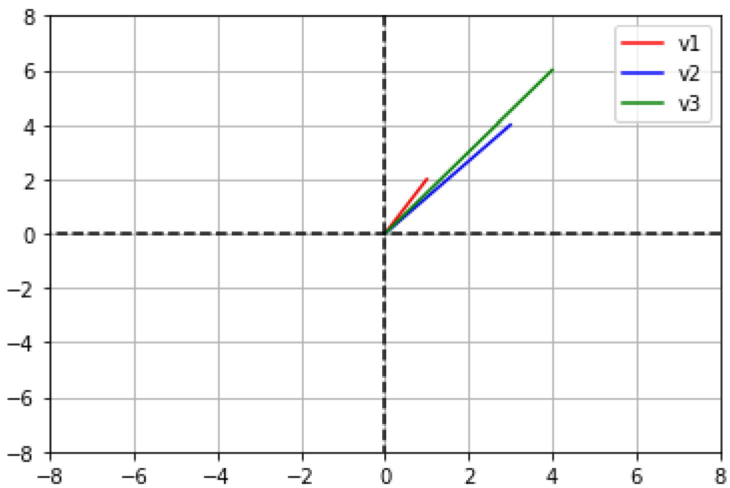
```
In [21]: fig = plt.figure()
ax = fig.add_subplot(111, projection='3d', auto_add_to_figure=False)
u = [1, 2, 3]  # Define the vector
ax.plot([0, u[0]], [0, u[1]], [0, u[2]])  # Plot the vector
ax.set_box_aspect([1, 1, 1])  # Set the aspect ratio of the 3D box
ax.plot([0, 0], [0, 0], [-5, 5], 'k--')  # Plot the x, y, and z axes
ax.plot([0, 0], [-5, 5], [0, 0], 'k--')
ax.plot([-5, 5], [0, 0], [0, 0], 'k--')
fig.add_axes(ax)  # Add the Axes3D object to the figure
plt.show()
```



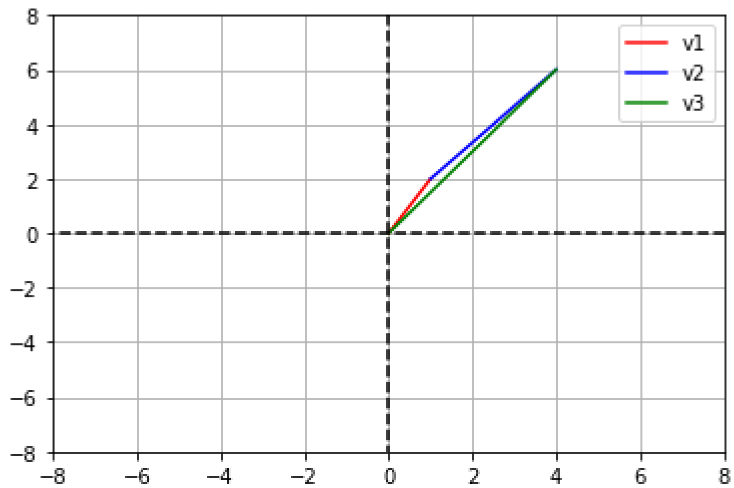
Vector Addition

In [22]:
```
v1 = np.array([1,2])
v2 = np.array([3,4])
v3 = v1+v2
v3 = np.add(v1,v2)
print('V3 =' ,v3)
plt.plot([0,v1[0]] , [0,v1[1]] , 'r' , label = 'v1')
plt.plot([0,v2[0]] , [0,v2[1]], 'b' , label = 'v2')
plt.plot([0,v3[0]] , [0,v3[1]] , 'g' , label = 'v3')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```

```
V3 = [4 6]
```

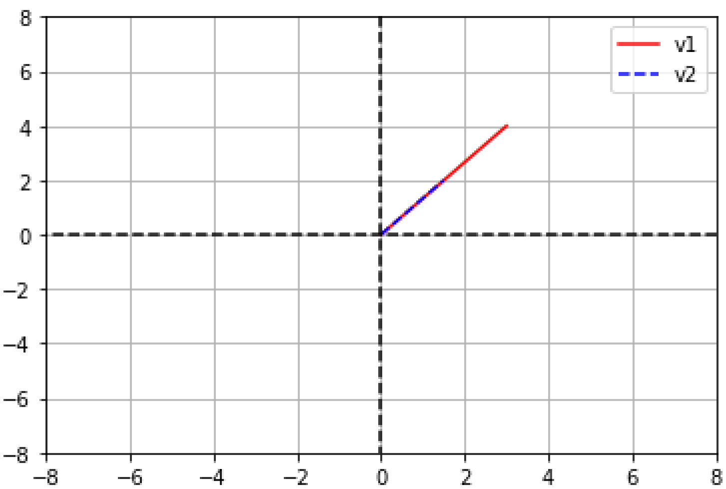


In [23]:
```
plt.plot([0,v1[0]] , [0,v1[1]] , 'r' , label = 'v1')
plt.plot([0,v2[0]]+v1[0] , [0,v2[1]]+v1[1], 'b' , label = 'v2')
plt.plot([0,v3[0]] , [0,v3[1]] , 'g' , label = 'v3')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```

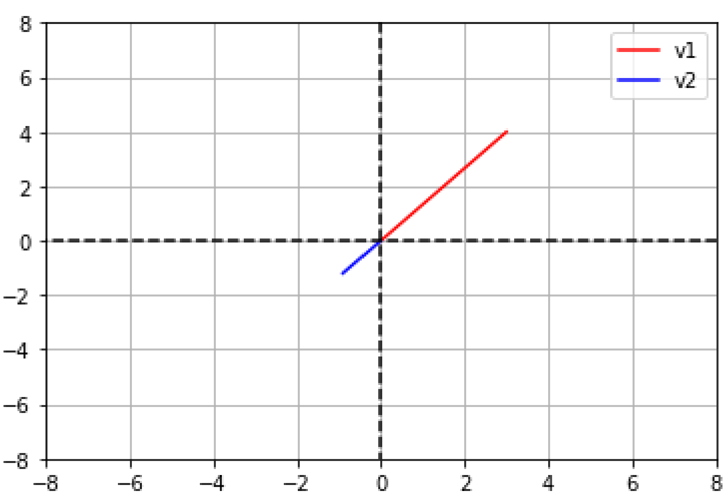


Scalar Multiplication

In [24]:
```
u1 = np.array([3,4])
a = .5
u2 = u1*a
plt.plot([0,u1[0]] , [0,u1[1]] , 'r' , label = 'v1')
plt.plot([0,u2[0]] , [0,u2[1]], 'b--' , label = 'v2')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```



In [25]:
```
u1 = np.array([3,4])
a = -.3
u2 = u1*a
plt.plot([0,u1[0]] , [0,u1[1]] , 'r' , label = 'v1')
plt.plot([0,u2[0]] , [0,u2[1]], 'b' , label = 'v2')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```



Multiplication of vectors

In [26]:
```
a1 = [5 , 6 ,8]
a2 = [4, 7 , 9]
print(np.multiply(a1,a2))
```

```
[20 42 72]
```

Dot Product

Dot Product :

- https://www.youtube.com/watch?v=WNulhXo39_k (https://www.youtube.com/watch?v=WNulhXo39_k)
- https://www.youtube.com/watch?v=LyGKycYT2v0 (https://www.youtube.com/watch?v=LyGKycYT2v0)

In [27]:
```
a1 = np.array([1,2,3])
a2 = np.array([4,5,6])

dotp = a1@a2
print(" Dot product - ",dotp)

dotp = np.dot(a1,a2)
print(" Dot product usign np.dot",dotp)

dotp = np.inner(a1,a2)
print(" Dot product usign np.inner", dotp)

dotp = sum(np.multiply(a1,a2))
print(" Dot product usign np.multiply & sum",dotp)

dotp = np.matmul(a1,a2)
print(" Dot product usign np.matmul",dotp)

dotp = 0
for i in range(len(a1)):
    dotp = dotp + a1[i]*a2[i]
print(" Dot product usign for loop" , dotp)
```

```
 Dot product -  32
 Dot product usign np.dot 32
 Dot product usign np.inner 32
 Dot product usign np.multiply & sum 32
 Dot product usign np.matmul 32
 Dot product usign for loop 32
```

Length of Vector

In [28]:
```
v3 = np.array([1,2,3,4,5,6])
length = np.sqrt(np.dot(v3,v3))
length
```

Out[28]: 9.539392014169456

In [29]:
```
v3 = np.array([1,2,3,4,5,6])
length = np.sqrt(sum(np.multiply(v3,v3)))
length
```

Out[29]: 9.539392014169456

In [30]:
```
v3 = np.array([1,2,3,4,5,6])
length = np.sqrt(np.matmul(v3,v3))
length
```

Out[30]: 9.539392014169456

Normalized Vector

How to normalize a vector : https://www.youtube.com/watch?v=7fn03DIW3Ak (https://www.youtube.com/watch?v=7fn03DIW3Ak)

In [31]:
```
v1 = [2,3]
length_v1 = np.sqrt(np.dot(v1,v1))
norm_v1 = v1/length_v1
length_v1 , norm_v1
```

Out[31]: (3.605551275463989, array([0.5547002 , 0.83205029]))

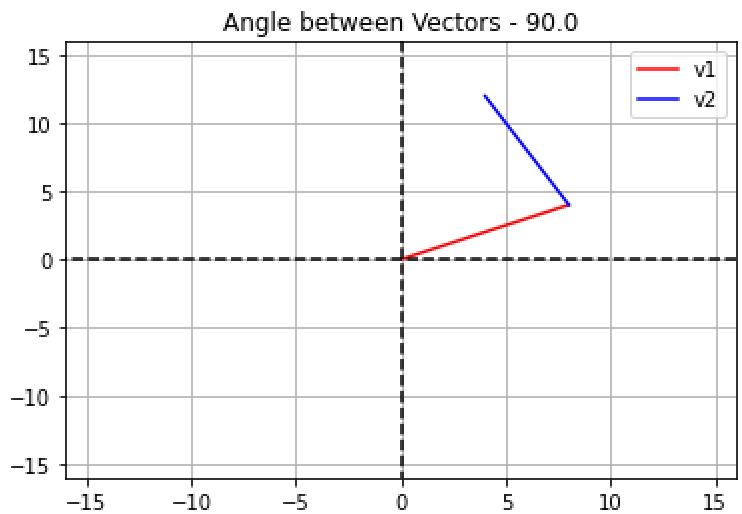
In [32]:
```
v1 = [2,3]
norm_v1 = v1/np.linalg.norm(v1)
norm_v1
```

Out[32]: array([0.5547002 , 0.83205029])

Angle between vectors

Angle between two vectors : https://www.youtube.com/watch?v=WDdR5s0C4cY (https://www.youtube.com/watch?v=WDdR5s0C4cY)

In [33]:
```
#First Method
v1 = np.array([8,4])
v2 = np.array([-4,8])
ang = np.rad2deg(np.arccos( np.dot(v1,v2) / (np.linalg.norm(v1)*np.linalg.n
plt.plot([0,v1[0]] , [0,v1[1]] , 'r' , label = 'v1')
plt.plot([0,v2[0]]+v1[0] , [0,v2[1]]+v1[1], 'b' , label = 'v2')
plt.plot([16,-16] , [0,0] , 'k--')
plt.plot([0,0] , [16,-16] , 'k--')
plt.grid()
plt.axis((-16, 16, -16, 16))
plt.legend()
plt.title('Angle between Vectors - %s'  %ang)
plt.show()
```



In [34]:
```
#Second Method
v1 = np.array([4,3])
v2 = np.array([-3,4])
lengthV1 = np.sqrt(np.dot(v1,v1))
lengthV2  = np.sqrt(np.dot(v2,v2))
ang = np.rad2deg(np.arccos( np.dot(v1,v2) / (lengthV1 * lengthV2)))
print('Angle between Vectors - %s' %ang)
```

```
Angle between Vectors - 90.0
```

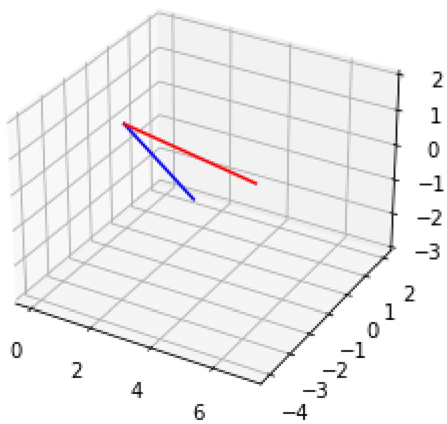
In [35]:
```
v1 = np.array([1,2,-3])
v2 = np.array([7,-4,2])

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot([0, v1[0]], [0, v1[1]], [0, v1[2]], 'b')
ax.plot([0, v2[0]], [0, v2[1]], [0, v2[2]], 'r')

ang = np.rad2deg(np.arccos(np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg
plt.title('Angle between vectors: %s degrees.' % ang)

plt.show()
```

Angle between vectors: 103.01589221967096 degrees.



Inner & outer products

Inner and Outer Product : https://www.youtube.com/watch?v=FCmH4MqbFGs&t=2s (https://www.youtube.com/watch?v=FCmH4MqbFGs&t=2s)

In [36]:
```
# https://www.youtube.com/watch?v=FCmH4MqbFGs

v1 = np.array([1,2,3])
v2 = np.array([4,5,6])
np.inner(v1,v2)

print("\n Inner Product ==>  \n", np.inner(v1,v2))
print("\n Outer Product ==>  \n", np.outer(v1,v2))
```

```
 Inner Product ==>
 32

 Outer Product ==>
 [[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
```

Vector Cross Product

Vector Cross Product : [https://www.youtube.com/watch?v=pWbOisq1MJU (https://www.youtube.com/watch?v=pWbOisq1MJU)](https://www.youtube.com/watch?v=pWbOisq1MJU)

In [37]:
```
v1 = np.array([1,2,3])
v2 = np.array([4,5,6])
print("\nVector Cross Product ==>  \n", np.cross(v1,v2))
```

```

Vector Cross Product ==>
 [-3  6 -3]
```

Matrix Operations

Matrix Creation

In [38]:
```
A = np.array([[1,2,3,4] , [5,6,7,8] , [10 , 11 , 12 ,13] , [14,15,16,17]])
```

In [39]:
```
A
```

Out[39]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [10, 11, 12, 13],
       [14, 15, 16, 17]])
```

In [40]:
```
type(A)
```

Out[40]:
```
numpy.ndarray
```

In [41]:
```
A.dtype
```

Out[41]:
```
dtype('int64')
```

In [42]:
```
B = np.array([[1.5,2.07,3,4] , [5,6,7,8] , [10 , 11 , 12 ,13] , [14,15,16,1
B
```

Out[42]:
```
array([[ 1.5 ,  2.07,  3.  ,  4.  ],
       [ 5.  ,  6.  ,  7.  ,  8.  ],
       [10.  , 11.  , 12.  , 13.  ],
       [14.  , 15.  , 16.  , 17.  ]])
```

In [43]:
```
type(B)
```

Out[43]:
```
numpy.ndarray
```

In [44]:
```
B.dtype
```
Out[44]:
```
dtype('float64')
```

In [45]:
```
A.shape
```
Out[45]:
```
(4, 4)
```

In [46]:
```
A[0,]
```
Out[46]:
```
array([1, 2, 3, 4])
```

In [47]:
```
A[:,0]
```
Out[47]:
```
array([ 1,  5, 10, 14])
```

In [48]:
```
A[0,0]
```
Out[48]:
```
1
```

In [49]:
```
A[0][0]
```
Out[49]:
```
1
```

In [50]:
```
A[1:3 , 1:3]
```
Out[50]:
```
array([[ 6,  7],
       [11, 12]])
```

Matrix Types :

- https://www.youtube.com/watch?v=alc9i7V2e9Q&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=5 (https://www.youtube.com/watch?v=alc9i7V2e9Q&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=5)
- https://www.youtube.com/watch?v=nfG4NwLhH14&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=6 (https://www.youtube.com/watch?v=nfG4NwLhH14&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=6)

Zero Matrix

Zero Matrix - https://www.web-formulas.com/Math_Formulas/Linear_Algebra_Definition_of_Zero_Matrix.aspx (https://www.web-formulas.com/Math_Formulas/Linear_Algebra_Definition_of_Zero_Matrix.aspx)

In [51]:
```
np.zeros(9).reshape(3,3)
```

Out[51]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [52]:
```
np.zeros((3,3))
```

Out[52]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

Matrix of Ones

Matrix of Ones - [https://en.wikipedia.org/wiki/Matrix_of_ones (https://en.wikipedia.org/wiki/Matrix_of_ones)](https://en.wikipedia.org/wiki/Matrix_of_ones)

In [53]:
```
np.ones(9).reshape(3,3)
```

Out[53]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [54]:
```
np.ones((3,3))
```

Out[54]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

Matrix with Random Numbers

In [55]:
```
X = np.random.random((3,3))
X
```

Out[55]:
```
array([[0.30026247, 0.28235787, 0.7030474 ],
       [0.92228293, 0.35079884, 0.29033682],
       [0.39624807, 0.6496108 , 0.61999024]])
```

Identity Matrix

Identity Matrix : [https://en.wikipedia.org/wiki/Identity_matrix (https://en.wikipedia.org/wiki/Identity_matrix)](https://en.wikipedia.org/wiki/Identity_matrix)

In [56]:
```
I = np.eye(9)
I
```

Out[56]:
```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

Diagonal Matrix

Diagonal Matrix : https://en.wikipedia.org/wiki/Diagonal_matrix (https://en.wikipedia.org/wiki/Diagonal_matrix)

In [57]:
```
D = np.diag([1,2,3,4,5,6,7,8])
D
```

Out[57]:
```
array([[1, 0, 0, 0, 0, 0, 0, 0],
       [0, 2, 0, 0, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 0, 0, 0],
       [0, 0, 0, 4, 0, 0, 0, 0],
       [0, 0, 0, 0, 5, 0, 0, 0],
       [0, 0, 0, 0, 0, 6, 0, 0],
       [0, 0, 0, 0, 0, 0, 7, 0],
       [0, 0, 0, 0, 0, 0, 0, 8]])
```

Traingular Matrices (lower & Upper triangular matrix)

Traingular Matrices : https://en.wikipedia.org/wiki/Triangular_matrix (https://en.wikipedia.org/wiki/Triangular_matrix)

In [58]:
```
M = np.random.randint(0, 10, size=(5,5))
U = np.triu(M)
L = np.tril(M)
print("lower triangular matrix - \n" , M)
print("\n")


print("lower triangular matrix - \n" , L)
print("\n")

print("Upper triangular matrix - \n" , U)
```

```
lower triangular matrix -
 [[2 7 5 3 0]
 [2 9 8 2 2]
 [8 4 7 4 8]
 [2 2 4 4 6]
 [8 1 8 0 5]]


lower triangular matrix -
 [[2 0 0 0 0]
 [2 9 0 0 0]
 [8 4 7 0 0]
 [2 2 4 4 0]
 [8 1 8 0 5]]


Upper triangular matrix -
 [[2 7 5 3 0]
 [0 9 8 2 2]
 [0 0 7 4 8]
 [0 0 0 4 6]
 [0 0 0 0 5]]
```

Concatenate Matrices

Matrix Concatenation :
[https://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html (https://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html)](https://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html)

In [59]:
```
A = np.array([[1,2] , [3,4] ,[5,6]])
B = np.array([[1,1] , [1,1]])
C = np.concatenate((A,B))
C , C.shape , type(C) , C.dtype
```

Out[59]:
```
(array([[1, 2],
        [3, 4],
        [5, 6],
        [1, 1],
        [1, 1]]),
 (5, 2),
 numpy.ndarray,
 dtype('int64'))
```

In [60]:
```
np.full((5,5) , 8)
```

Out[60]:
```
array([[8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8]])
```

In [61]:
```
M
```

Out[61]:
```
array([[2, 7, 5, 3, 0],
       [2, 9, 8, 2, 2],
       [8, 4, 7, 4, 8],
       [2, 2, 4, 4, 6],
       [8, 1, 8, 0, 5]])
```

In [62]:
```
M.flatten()
```

Out[62]:
```
array([2, 7, 5, 3, 0, 2, 9, 8, 2, 2, 8, 4, 7, 4, 8, 2, 2, 4, 4, 6, 8, 1,
       8, 0, 5])
```

Matrix Addition

Matrix Addition : https://www.youtube.com/watch?v=ZCmVpGv6_1g (https://www.youtube.com/watch?v=ZCmVpGv6_1g)

In [63]:
```
#*********************************************************#
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M)  ==>  \n", M)
print("\n Second Matrix (N)  ==>  \n", N)

C = M+N
print("\n Matrix Addition (M+N)  ==>  \n", C)

# OR

C = np.add(M,N,dtype = np.float64)
print("\n Matrix Addition using np.add  ==>  \n", C)

#*********************************************************#
```

```
 First Matrix (M)  ==>
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

 Second Matrix (N)  ==>
 [[1 1 1]
 [2 2 2]
 [3 3 3]]

 Matrix Addition (M+N)  ==>
 [[ 2  3  4]
 [ 6 -1  8]
 [10 11  3]]

 Matrix Addition using np.add  ==>
 [[ 2.  3.  4.]
 [ 6. -1.  8.]
 [10. 11.  3.]]
```

Matrix subtraction

Matrix subtraction : https://www.youtube.com/watch?v=7jb_AO_hRc8&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=8 (https://www.youtube.com/watch?v=7jb_AO_hRc8&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=8)

In [64]:

```
#*****************************************************#
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M)  ==>  \n", M)
print("\n Second Matrix (N)  ==>  \n", N)

C = M-N
print("\n Matrix Subtraction (M-N)  ==>  \n", C)

# OR

C = np.subtract(M,N,dtype = np.float64)
print("\n Matrix Subtraction using np.subtract  ==>  \n", C)

#*****************************************************#
```

```
 First Matrix (M)  ==>  
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

 Second Matrix (N)  ==>  
 [[1 1 1]
 [2 2 2]
 [3 3 3]]

 Matrix Subtraction (M-N)  ==>  
 [[ 0  1  2]
 [ 2 -5  4]
 [ 4  5 -3]]

 Matrix Subtraction using np.subtract  ==>  
 [[ 0.  1.  2.]
 [ 2. -5.  4.]
 [ 4.  5. -3.]]
```

Matrices Scalar Multiplication

Matrices Scalar Multiplication : https://www.youtube.com/watch?v=4lHyTQH1iS8&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=9 (https://www.youtube.com/watch?v=4lHyTQH1iS8&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=9)

In [65]:
```
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

C = 10

print("\n Matrix (M)  ==>  \n", M)

print("\nMatrices Scalar Multiplication ==>  \n", C*M)

# OR

print("\nMatrices Scalar Multiplication ==>  \n", np.multiply(C,M))
```

```
 Matrix (M)  ==>  
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

Matrices Scalar Multiplication ==>  
 [[ 10  20  30]
 [ 40 -30  60]
 [ 70  80   0]]

Matrices Scalar Multiplication ==>  
 [[ 10  20  30]
 [ 40 -30  60]
 [ 70  80   0]]
```

Transpose of a matrix

Transpose of a matrix : https://www.youtube.com/watch?v=g_Rz94DXvNo&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=13 (https://www.youtube.com/watch?v=g_Rz94DXvNo&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=13)

In [66]:
```
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M)  ==>  \n", M)

print("\nTranspose of M ==>  \n", np.transpose(M))

# OR

print("\nTranspose of M ==>  \n", M.T)
```

```
 Matrix (M)  ==> 
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

Transpose of M ==> 
 [[ 1  4  7]
 [ 2 -3  8]
 [ 3  6  0]]

Transpose of M ==> 
 [[ 1  4  7]
 [ 2 -3  8]
 [ 3  6  0]]
```

Determinant of a matrix

Determinant of a matrix :

- https://www.youtube.com/watch?v=21LWuY8i6Hw&t=88s (https://www.youtube.com/watch?v=21LWuY8i6Hw&t=88s)
- https://www.youtube.com/watch?v=Ip3X9LOh2dk&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=6 (https://www.youtube.com/watch?v=Ip3X9LOh2dk&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=6)

In [67]:
```
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M)  ==>  \n", M)

print("\nDeterminant of M ==>  ", np.linalg.det(M))
```

```
 Matrix (M)  ==> 
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

Determinant of M ==>   195.0
```

Rank of a matrix

In [68]:
```
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M)  ==>  \n", M)

print("\nRank of M ==> ", np.linalg.matrix_rank(M))
```

```
 Matrix (M)  ==> 
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

Rank of M ==>  3
```

Trace of matrix

In [69]:
```
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M)  ==>  \n", M)

print("\nTrace of M ==> ", np.trace(M))
```

```
 Matrix (M)  ==> 
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

Trace of M ==>  -2
```

Inverse of matrix A

Inverse of matrix : [https://www.youtube.com/watch?v=pKZyszzmyeQ (https://www.youtube.com/watch?v=pKZyszzmyeQ)](https://www.youtube.com/watch?v=pKZyszzmyeQ)

In [70]:
```python
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])

print("\n Matrix (M)  ==>  \n", M)

print("\nInverse of M ==> \n", np.linalg.inv(M))
```

```
 Matrix (M)  ==>  
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

Inverse of M ==> 
 [[-0.24615385  0.12307692  0.10769231]
 [ 0.21538462 -0.10769231  0.03076923]
 [ 0.27179487  0.03076923 -0.05641026]]
```

Matrix Multiplication (pointwise multiplication)

In [71]:
```python
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M)  ==>  \n", M)
print("\n Second Matrix (N)  ==>  \n", N)

print("\n Point-Wise Multiplication of M & N  ==> \n", M*N)

# OR

print("\n Point-Wise Multiplication of M & N  ==> \n", np.multiply(M,N))
```

```
 First Matrix (M)  ==>  
 [[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

 Second Matrix (N)  ==>  
 [[1 1 1]
 [2 2 2]
 [3 3 3]]

 Point-Wise Multiplication of M & N  ==> 
 [[ 1  2  3]
 [ 8 -6 12]
 [21 24  0]]

 Point-Wise Multiplication of M & N  ==> 
 [[ 1  2  3]
 [ 8 -6 12]
 [21 24  0]]
```

Matrix dot product

Matrix Multiplication :

- https://www.youtube.com/watch?v=vzt9c7iWPxs&t=207s (https://www.youtube.com/watch?v=vzt9c7iWPxs&t=207s)
- https://www.youtube.com/watch?v=XkY2DOUCWMU&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=4 (https://www.youtube.com/watch?v=XkY2DOUCWMU&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=4)

In [72]:

```
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M)  ==>  \n", M)
print("\n Second Matrix (N)  ==>  \n", N)

print("\n Matrix Dot Product ==> \n", M@N)

# OR

print("\n Matrix Dot Product using np.matmul ==> \n", np.matmul(M,N))

# OR

print("\n Matrix Dot Product using np.dot ==> \n", np.dot(M,N))
```

```
 First Matrix (M)  ==>
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

 Second Matrix (N)  ==>
[[1 1 1]
 [2 2 2]
 [3 3 3]]

 Matrix Dot Product ==>
[[14 14 14]
 [16 16 16]
 [23 23 23]]

 Matrix Dot Product using np.matmul ==>
[[14 14 14]
 [16 16 16]
 [23 23 23]]

 Matrix Dot Product using np.dot ==>
[[14 14 14]
 [16 16 16]
 [23 23 23]]
```

Matrix Division

In [73]:
```
M = np.array([[1,2,3],[4,-3,6],[7,8,0]])
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n First Matrix (M)  ==>  \n", M)
print("\n Second Matrix (N)  ==>  \n", N)


print("\n Matrix Division (M/N)   ==> \n", M/N)

# OR

print("\n Matrix Division (M/N)   ==> \n", np.divide(M,N))
```

```
 First Matrix (M)  ==>  
[[ 1  2  3]
 [ 4 -3  6]
 [ 7  8  0]]

 Second Matrix (N)  ==>  
[[1 1 1]
 [2 2 2]
 [3 3 3]]

 Matrix Division (M/N)   ==> 
[[ 1.          2.          3.        ]
 [ 2.         -1.5         3.        ]
 [ 2.33333333  2.66666667  0.        ]]

 Matrix Division (M/N)   ==> 
[[ 1.          2.          3.        ]
 [ 2.         -1.5         3.        ]
 [ 2.33333333  2.66666667  0.        ]]
```

Sum of all elements in a matrix

In [74]:
```python
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n Matrix (N)  ==>  \n", N)


print ("Sum of all elements in a Matrix  ==>")
print (np.sum(N))
```

```
 Matrix (N)  ==>
 [[1 1 1]
 [2 2 2]
 [3 3 3]]
Sum of all elements in a Matrix  ==>
18
```

Column-Wise Addition

In [75]:
```python
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n Matrix (N)  ==>  \n", N)

print ("Column-Wise summation ==> ")
print (np.sum(N,axis=0))
```

```
 Matrix (N)  ==>
 [[1 1 1]
 [2 2 2]
 [3 3 3]]
Column-Wise summation ==>
[6 6 6]
```

Row-Wise Addition

In [76]:
```python
N = np.array([[1,1,1],[2,2,2],[3,3,3]])

print("\n Matrix (N)  ==>  \n", N)

print ("Row-Wise summation  ==>")
print (np.sum(N,axis=1))
```

```
 Matrix (N)  ==>
 [[1 1 1]
 [2 2 2]
 [3 3 3]]
Row-Wise summation  ==>
[3 6 9]
```

Kronecker Product of matrices

Kronecker Product of matrices : [https://www.youtube.com/watch?v=e1UJXvu8VZk (https://www.youtube.com/watch?v=e1UJXvu8VZk)](https://www.youtube.com/watch?v=e1UJXvu8VZk)

In [77]:
```
M1 = np.array([[1,2,3] , [4,5,6]])
M1
```

Out[77]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [78]:
```
M2 = np.array([[10,10,10],[10,10,10]])
M2
```

Out[78]:
```
array([[10, 10, 10],
       [10, 10, 10]])
```

In [79]:
```
np.kron(M1,M2)
```

Out[79]:
```
array([[10, 10, 10, 20, 20, 20, 30, 30, 30],
       [10, 10, 10, 20, 20, 20, 30, 30, 30],
       [40, 40, 40, 50, 50, 50, 60, 60, 60],
       [40, 40, 40, 50, 50, 50, 60, 60, 60]])
```

Matrix Vector Multiplication

In [80]:
```
A = np.array([[1,2,3] ,[4,5,6]])
v = np.array([10,20,30])
print ("Matrix Vector Multiplication  ==> \n" , A*v)
```

```
Matrix Vector Multiplication  ==>
 [[ 10  40  90]
 [ 40 100 180]]
```

Matrix Vector Dot Product

In [81]:
```
A = np.array([[1,2,3] ,[4,5,6]])
v = np.array([10,20,30])

print ("Matrix Vector Multiplication  ==> \n" , A@v)
```

```
Matrix Vector Multiplication  ==>
 [140 320]
```

Matrix Powers

In [82]:
```
M1 = np.array([[1,2],[4,5]])
M1
```

Out[82]:
```
array([[1, 2],
       [4, 5]])
```

In [83]:
```
#Matrix to the power 3

M1@M1@M1
```

Out[83]:
```
array([[ 57,  78],
       [156, 213]])
```

In [84]:
```
#Matrix to the power 3

np.linalg.matrix_power(M1,3)
```

Out[84]:
```
array([[ 57,  78],
       [156, 213]])
```

Tensor

What is Tensor :

- [https://www.youtube.com/watch?v=f5liqUk0ZTw (https://www.youtube.com/watch?v=f5liqUk0ZTw)](https://www.youtube.com/watch?v=f5liqUk0ZTw)
- [https://www.youtube.com/watch?v=bpG3gqDM80w&t=634s (https://www.youtube.com/watch?v=bpG3gqDM80w&t=634s)](https://www.youtube.com/watch?v=bpG3gqDM80w&t=634s)
- [https://www.youtube.com/watch?v=uaQeXi4E7gA (https://www.youtube.com/watch?v=uaQeXi4E7gA)](https://www.youtube.com/watch?v=uaQeXi4E7gA)

In [85]:
```python
# Create Tensor

T1 = np.array([
  [[1,2,3],    [4,5,6],    [7,8,9]],
  [[10,20,30], [40,50,60], [70,80,90]],
  [[100,200,300], [400,500,600], [700,800,900]],
  ])

T1
```

Out[85]:
```
array([[[  1,   2,   3],
        [  4,   5,   6],
        [  7,   8,   9]],

       [[ 10,  20,  30],
        [ 40,  50,  60],
        [ 70,  80,  90]],

       [[100, 200, 300],
        [400, 500, 600],
        [700, 800, 900]]])
```

In [86]:
```python
T2 = np.array([
  [[3,3,3] , [4,4,4] , [5,5,5]],
  [[1,1,1] , [1,1,1] , [1,1,1]],
  [[2,2,2] , [2,2,2] , [2,2,2]]

])

T2
```

Out[86]:
```
array([[[3, 3, 3],
        [4, 4, 4],
        [5, 5, 5]],

       [[1, 1, 1],
        [1, 1, 1],
        [1, 1, 1]],

       [[2, 2, 2],
        [2, 2, 2],
        [2, 2, 2]]])
```

Tensor Addition

In [87]:
```
A = T1+T2
A
```

Out[87]:
```
array([[[  4,   5,   6],
        [  8,   9,  10],
        [ 12,  13,  14]],

       [[ 11,  21,  31],
        [ 41,  51,  61],
        [ 71,  81,  91]],

       [[102, 202, 302],
        [402, 502, 602],
        [702, 802, 902]]])
```

In [88]:
```
np.add(T1,T2)
```

Out[88]:
```
array([[[  4,   5,   6],
        [  8,   9,  10],
        [ 12,  13,  14]],

       [[ 11,  21,  31],
        [ 41,  51,  61],
        [ 71,  81,  91]],

       [[102, 202, 302],
        [402, 502, 602],
        [702, 802, 902]]])
```

Tensor Subtraction

In [89]:
```
S = T1-T2
S
```

Out[89]:
```
array([[[ -2,  -1,   0],
        [  0,   1,   2],
        [  2,   3,   4]],

       [[  9,  19,  29],
        [ 39,  49,  59],
        [ 69,  79,  89]],

       [[ 98, 198, 298],
        [398, 498, 598],
        [698, 798, 898]]])
```

In [90]:
```
np.subtract(T1,T2)
```

Out[90]:
```
array([[[ -2,  -1,   0],
        [  0,   1,   2],
        [  2,   3,   4]],

       [[  9,  19,  29],
        [ 39,  49,  59],
        [ 69,  79,  89]],

       [[ 98, 198, 298],
        [398, 498, 598],
        [698, 798, 898]]])
```

Tensor Element-Wise Product

In [91]:
```
P = T1*T2
P
```

Out[91]:
```
array([[[   3,    6,    9],
        [  16,   20,   24],
        [  35,   40,   45]],

       [[  10,   20,   30],
        [  40,   50,   60],
        [  70,   80,   90]],

       [[ 200,  400,  600],
        [ 800, 1000, 1200],
        [1400, 1600, 1800]]])
```

In [92]:
```
np.multiply(T1,T2)
```

Out[92]:
```
array([[[   3,    6,    9],
        [  16,   20,   24],
        [  35,   40,   45]],

       [[  10,   20,   30],
        [  40,   50,   60],
        [  70,   80,   90]],

       [[ 200,  400,  600],
        [ 800, 1000, 1200],
        [1400, 1600, 1800]]])
```

Tensor Element-Wise Division

In [93]:
```
D = T1/T2
D
```

Out[93]:
```
array([[[3.33333333e-01, 6.66666667e-01, 1.00000000e+00],
        [1.00000000e+00, 1.25000000e+00, 1.50000000e+00],
        [1.40000000e+00, 1.60000000e+00, 1.80000000e+00]],

       [[1.00000000e+01, 2.00000000e+01, 3.00000000e+01],
        [4.00000000e+01, 5.00000000e+01, 6.00000000e+01],
        [7.00000000e+01, 8.00000000e+01, 9.00000000e+01]],

       [[5.00000000e+01, 1.00000000e+02, 1.50000000e+02],
        [2.00000000e+02, 2.50000000e+02, 3.00000000e+02],
        [3.50000000e+02, 4.00000000e+02, 4.50000000e+02]]])
```

In [94]:
```
np.divide(T1,T2)
```

Out[94]:
```
array([[[3.33333333e-01, 6.66666667e-01, 1.00000000e+00],
        [1.00000000e+00, 1.25000000e+00, 1.50000000e+00],
        [1.40000000e+00, 1.60000000e+00, 1.80000000e+00]],

       [[1.00000000e+01, 2.00000000e+01, 3.00000000e+01],
        [4.00000000e+01, 5.00000000e+01, 6.00000000e+01],
        [7.00000000e+01, 8.00000000e+01, 9.00000000e+01]],

       [[5.00000000e+01, 1.00000000e+02, 1.50000000e+02],
        [2.00000000e+02, 2.50000000e+02, 3.00000000e+02],
        [3.50000000e+02, 4.00000000e+02, 4.50000000e+02]]])
```

Tensor Dot Product

In [95]:
```
T1
```

Out[95]:
```
array([[[  1,   2,   3],
        [  4,   5,   6],
        [  7,   8,   9]],

       [[ 10,  20,  30],
        [ 40,  50,  60],
        [ 70,  80,  90]],

       [[100, 200, 300],
        [400, 500, 600],
        [700, 800, 900]]])
```

In [96]:
```
T2
```
Out[96]:
```
array([[[3, 3, 3],
        [4, 4, 4],
        [5, 5, 5]],

       [[1, 1, 1],
        [1, 1, 1],
        [1, 1, 1]],

       [[2, 2, 2],
        [2, 2, 2],
        [2, 2, 2]]])
```
In [97]:
```
np.tensordot(T1,T2)
```
Out[97]:
```
array([[  89,   89,   89],
       [ 890,  890,  890],
       [8900, 8900, 8900]])
```

Solving Equations

$$AX = B$$

Solving Equations :

- [https://www.youtube.com/watch?v=NNmiOoWt86M (https://www.youtube.com/watch?v=NNmiOoWt86M)](https://www.youtube.com/watch?v=NNmiOoWt86M)
- [https://www.youtube.com/watch?v=a2z7sZ4MSqo (https://www.youtube.com/watch?v=a2z7sZ4MSqo)](https://www.youtube.com/watch?v=a2z7sZ4MSqo)

In [98]:
```
A = np.array([[1,2,3] , [4,5,6] , [7,8,9]])
A
```
Out[98]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```
In [99]:
```
B = np.random.random((3,1))
B
```
Out[99]:
```
array([[0.17629069],
       [0.28765383],
       [0.68182386]])
```

In [100]:
```
# Ist Method
X = np.dot(np.linalg.inv(A) , B)
X
```

Out[100]:
```
array([[-1.27364894e+15],
       [ 2.54729789e+15],
       [-1.27364894e+15]])
```

In [101]:
```
# 2nd Method
X = np.matmul(np.linalg.inv(A) , B)
X
```

Out[101]:
```
array([[-1.27364894e+15],
       [ 2.54729789e+15],
       [-1.27364894e+15]])
```

In [102]:
```
# 3rd Method
X = np.linalg.inv(A)@B
X
```

Out[102]:
```
array([[-1.27364894e+15],
       [ 2.54729789e+15],
       [-1.27364894e+15]])
```

In [103]:
```
# 4th Method
X = np.linalg.solve(A,B)
X
```

Out[103]:
```
array([[-1.27364894e+15],
       [ 2.54729789e+15],
       [-1.27364894e+15]])
```

Eigenvalues & Eigenvectors of a Matrix

Eigenvalues & Eigenvectors of a Matrix

https://www.youtube.com/watch?v=PFDu9oVAE-g&t=36s (https://www.youtube.com/watch?v=PFDu9oVAE-g&t=36s)

https://www.youtube.com/watch?v=cdZnhQjJu4I&t=557s (https://www.youtube.com/watch?v=cdZnhQjJu4I&t=557s)

In [104]:
```python
M = np.array([[4, 2], [7, 3]])

eigval, eigvec = np.linalg.eig(M)

print ("Matrix  ==> \n" , M, "\n")


print(f"Eigen Value :- \n {eigval} \n")

print(f"Eigen Value :- \n {eigvec} \n")
```

```
Matrix  ==>
 [[4 2]
 [7 3]]

Eigen Value :-
 [ 7.27491722 -0.27491722]

Eigen Value :-
 [[ 0.52119606 -0.42376194]
 [ 0.85343697  0.9057736 ]]
```

Singular Value Decomposition of a Matrix

Singular Value Decomposition : -

https://www.youtube.com/watch?v=mBcLRGuAFUk (https://www.youtube.com/watch?v=mBcLRGuAFUk)

https://www.youtube.com/watch?v=rYz83XPxiZo (https://www.youtube.com/watch?v=rYz83XPxiZo)

In [105]:
```python
M = np.array([[1, 2], [3, 4], [5, 6]])
U, s, V = np.linalg.svd(L)
print(f"\n{U}\n")
print(f"\n{s}\n")
print(f"\n{V}\n")
```

```
[[-0.07666398  0.02658081 -0.11755385  0.23046316  0.96254035]
 [-0.25636511 -0.90355717 -0.18911483 -0.28229977  0.04902842]
 [-0.63748277 -0.05978656 -0.00472725  0.73427798 -0.22551005]
 [-0.27443632 -0.04884689  0.93029359 -0.19342891  0.13941945]
 [-0.66835619  0.42061555 -0.29145868 -0.53908688  0.02863093]]


[17.44211647  8.82466033  4.36957284  3.04865669  1.22901124]


[[-0.66859102 -0.34826312 -0.62532401 -0.06293647 -0.19159263]
 [ 0.11728332 -0.91201686  0.31174355 -0.02214109  0.23831827]
 [-0.25683006 -0.03474343  0.31042259  0.85161056 -0.33350935]
 [ 0.35130153 -0.17369313  0.01756025 -0.25378904 -0.88413839]
 [ 0.59148645 -0.12474627 -0.64429442  0.45376135  0.11647955]]
```

QR Decomposition

QR Decomposition :

[https://www.youtube.com/watch?v=J41Ypt6Mftc (https://www.youtube.com/watch?v=J41Ypt6Mftc)](https://www.youtube.com/watch?v=J41Ypt6Mftc)

In [106]:
```python
M = np.array([[1, 2], [3, 4], [5, 6]])

Q, R = np.linalg.qr(M)

print(f"Q :- \n{Q}\n")
print(f"R :- \n{R}\n")
```

```
Q :-
[[-0.16903085  0.89708523]
 [-0.50709255  0.27602622]
 [-0.84515425 -0.34503278]]

R :-
[[-5.91607978 -7.43735744]
 [ 0.          0.82807867]]
```

Moore-Penrose Pseudoinverse

Moore-Penrose Pseudoinverse :

https://www.youtube.com/watch?v=vXk-o3PVUdU (https://www.youtube.com/watch?v=vXk-o3PVUdU)

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

pinv_M = np.linalg.pinv(M)

print(pinv_M)
```

```
[[-0.94444444  0.44444444]
 [-0.11111111  0.11111111]
 [ 0.72222222 -0.22222222]]
```

End