Prepared by Asif Bhat

Linear Algebra

In [73]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [17]:

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

In [18]:

```
v ,u
```

Out[18]:

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

In [19]:

```
type(v)
```

Out[19]:

```
list
```

In [20]:

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

In [21]:

```
type(w)
```

Out[21]:

```
numpy.ndarray
```

In [22]:

```
w.shape[0]
```

Out[22]:

```
3
```

In [23]:

```
w.shape
```

Out[23]:

```
(3,)
```

Reading elements from an array

In [24]:

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

In [25]:

```
a[0]
```

Out[25]:

```
7
```

In [26]:

```
a[1:]
```

Out[26]:

```
array([5, 3, 9, 0, 2])
```

In [27]:

```
a[1:4]
```

Out[27]:

```
array([5, 3, 9])
```

In [28]:

```
a[-1]
```

Out[28]:

```
2
```

In [29]:

```
a[-3]
```

Out[29]:

```
9
```

In [30]:

```
a[-6]
```

Out[30]:

```
7
```

In [31]:

```
a[-3:-1]
```

Out[31]:

```
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 [32]:

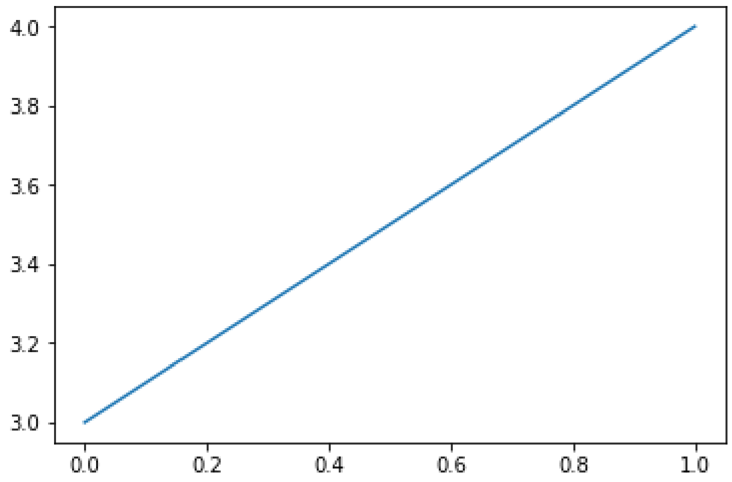
```
v = [3,4]
u = [1,2,3]
```

In [33]:

```
plt.plot (v)
```

Out[33]:

```
[<matplotlib.lines.Line2D at 0x1ec68a84d68>]
```

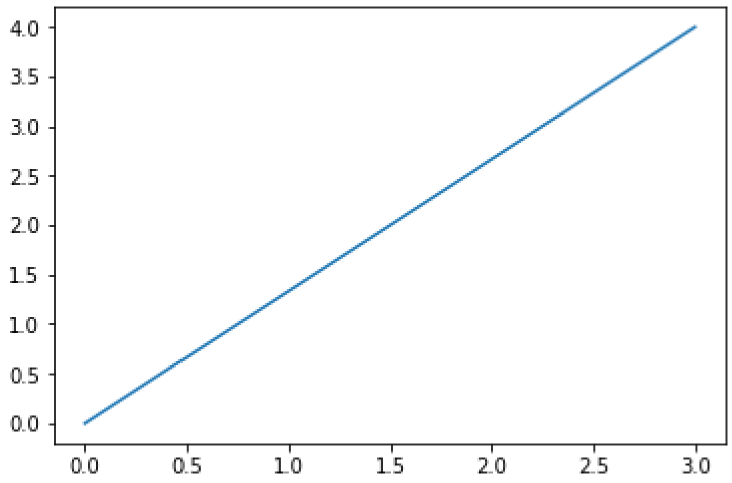


In [34]:

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

Out[34]:

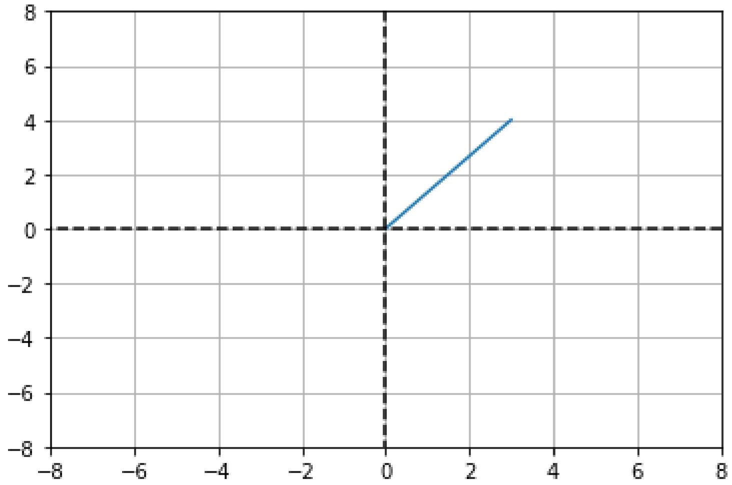
```
[<matplotlib.lines.Line2D at 0x1ec68b11f98>]
```



Plot 2D Vector

In [35]:

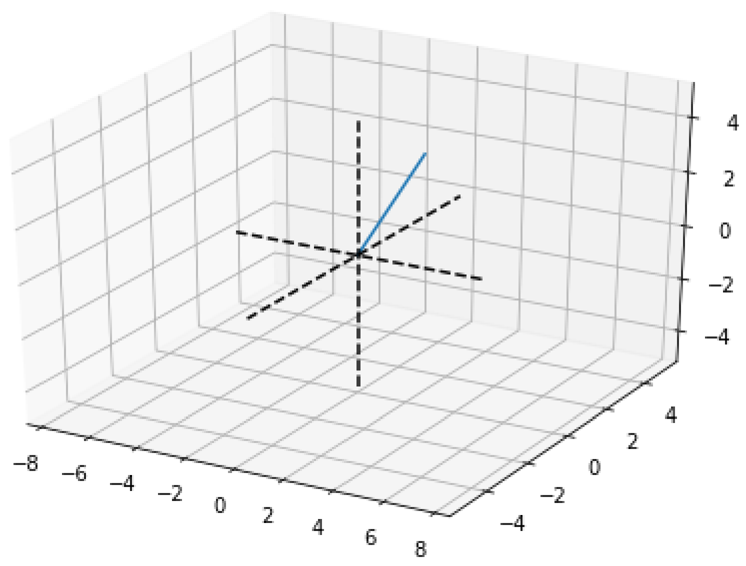
```
plt.plot([0,v[0]] , [0,v[1]])
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.show()
```



Plot the 3D vector

In [36]:

```
fig = plt.figure()
ax = Axes3D(fig)
ax.plot([0,u[0]],[0,u[1]],[0,u[2]])
plt.axis('equal')
ax.plot([0, 0],[0, 0],[-5, 5],'k--')
ax.plot([0, 0],[-5, 5],[0, 0],'k--')
ax.plot([-5, 5],[0, 0],[0, 0],'k--')
plt.show()
```

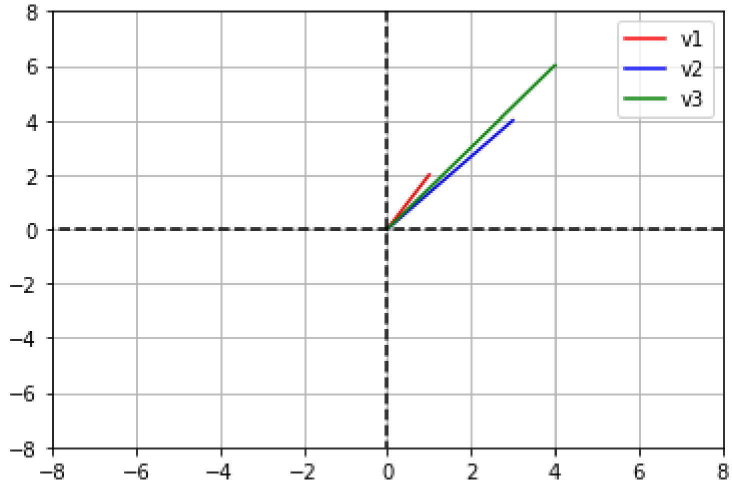


Vector Addition

In [37]:

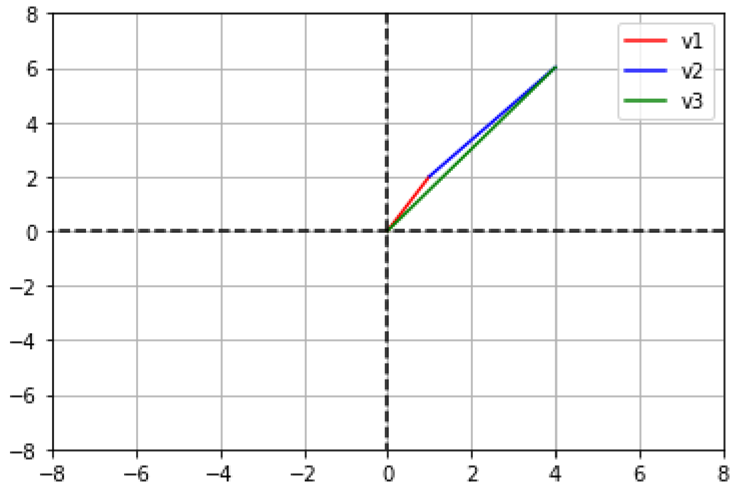
```
v1 = np.array([1,2])
v2 = np.array([3,4])
v3 = v1+v2
v3 = np.add(v1,v2)
print('V3 =' ,v3)
plt.plot([0,v1[0]] , [0,v1[1]] , 'r' , label = 'v1')
plt.plot([0,v2[0]] , [0,v2[1]], 'b' , label = 'v2')
plt.plot([0,v3[0]] , [0,v3[1]] , 'g' , label = 'v3')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] ,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

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



In [38]:

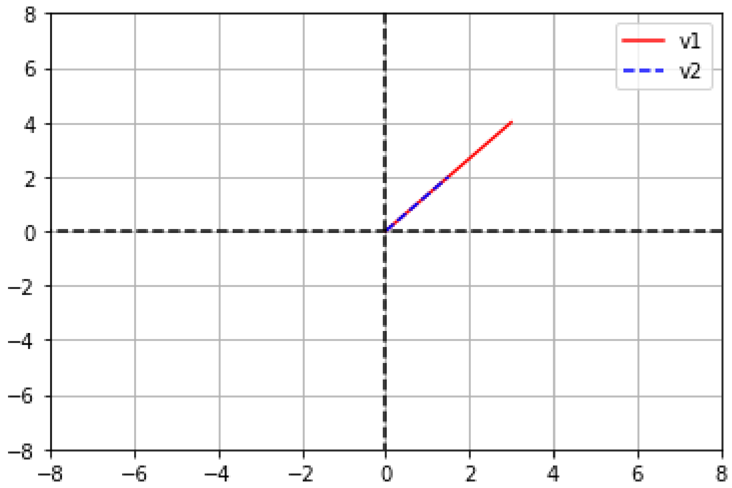
```
plt.plot([0,v1[0]] , [0,v1[1]] , 'r' , label = 'v1')
plt.plot([0,v2[0]]+v1[0] , [0,v2[1]]+v1[1], 'b' , label = 'v2')
plt.plot([0,v3[0]] , [0,v3[1]] , 'g' , label = 'v3')
plt.plot([8,-8] , [0,0] , 'k--')
plt.plot([0,0] , [8,-8] , 'k--')
plt.grid()
plt.axis((-8, 8, -8, 8))
plt.legend()
plt.show()
```



Scalar Multiplication

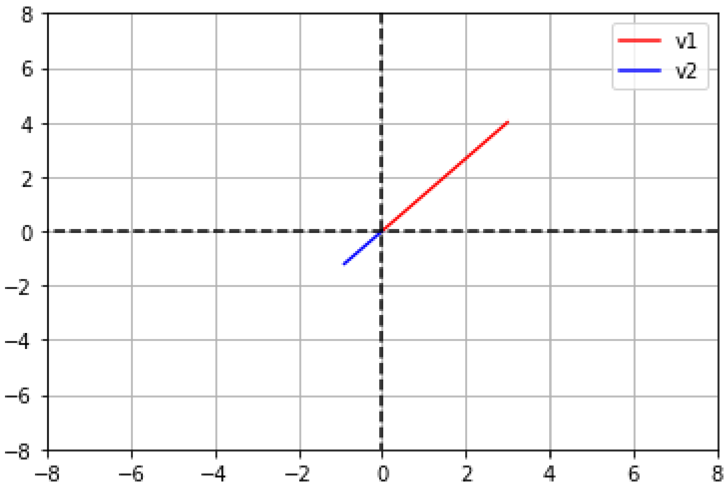
In [39]:

```
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 [40]:

```
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 [41]:

```
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=WNuIhXo39_k (https://www.youtube.com/watch?v=WNuIhXo39_k)
- https://www.youtube.com/watch?v=LyGKycYT2v0 (https://www.youtube.com/watch?v=LyGKycYT2v0)

In [42]:

```
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 [43]:

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

Out[43]:

```
9.539392014169456
```

In [44]:

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

Out[44]:

```
9.539392014169456
```

In [45]:

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

Out[45]:

```
9.539392014169456
```

Normalized Vector

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

In [46]:

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

Out[46]:

```
(3.605551275463989, array([0.5547002 , 0.83205029]))
```

In [47]:

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

Out[47]:

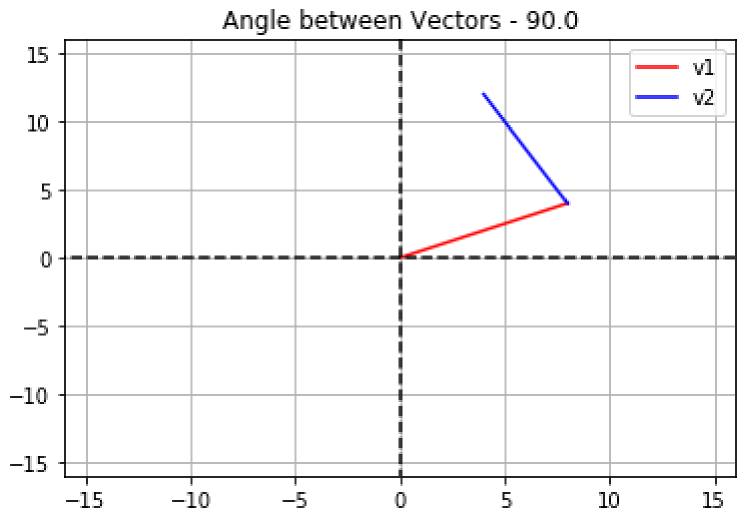
```
array([0.5547002 , 0.83205029])
```

Angle between vectors

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

In [48]:

```
#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.norm(v2))))
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 [49]:

```
#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)
```

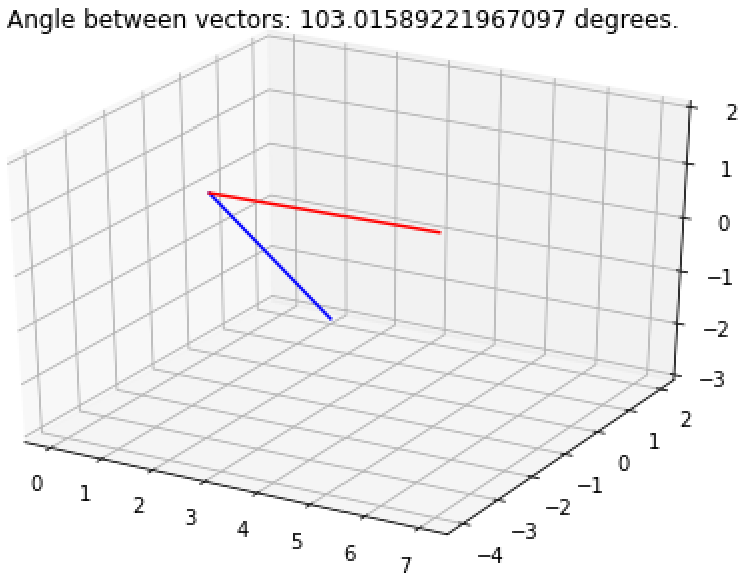
```
Angle between Vectors - 90.0
```

In [50]:

```
v1 = np.array([1,2,-3])
v2 = np.array([7,-4,2])
fig = plt.figure()
ax = Axes3D(fig)
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.norm(v2)) ))
plt.title('Angle between vectors: %s degrees.' %ang)
```

Out[50]:

```
Text(0.5,0.92,'Angle between vectors: 103.01589221967097 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)](https://www.youtube.com/watch?v=FCmH4MqbFGs&t=2s)

In [51]:

```
# 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 [52]:

```
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 [53]:

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

In [54]:

```
A
```

Out[54]:

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

In [55]:

```
type(A)
```

Out[55]:

```
numpy.ndarray
```

In [56]:

```
A.dtype
```

Out[56]:

```
dtype('int32')
```

In [57]:

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

Out[57]:

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

In [58]:

```
type(B)
```

Out[58]:

```
numpy.ndarray
```

In [59]:

```
B.dtype
```

Out[59]:

```
dtype('float64')
```

In [60]:

```
A.shape
```

Out[60]:

```
(4, 4)
```

In [61]:

```
A[0,]
```

Out[61]:

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

In [62]:

```
A[:,0]
```

Out[62]:

```
array([ 1,  5, 10, 14])
```

In [63]:

```
A[0,0]
```

Out[63]:

```
1
```

In [64]:

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

Out[64]:

```
1
```

In [65]:

```
A[1:3 , 1:3]
```

Out[65]:

```
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 [66]:

```
np.zeros(9).reshape(3,3)
```

Out[66]:

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

In [67]:

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

Out[67]:

```
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)

In [68]:

```
np.ones(9).reshape(3,3)
```

Out[68]:

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

In [69]:

```
np.ones((3,3))
```

Out[69]:

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

Matrix with Random Numbers

In [70]:

```
X = np.random.random((3,3))
X
```

Out[70]:

```
array([[0.30088476, 0.83059469, 0.44002496],
       [0.36264188, 0.14854413, 0.28324881],
       [0.37934998, 0.30606346, 0.02852849]])
```

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 [71]:

```
I = np.eye(9)
I
```

Out[71]:

```
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 [72]:

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

Out[72]:

```
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 [85]:

```
M = np.random.randn(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 - 
 [[ 0.04163482  0.84694284  1.27184552  0.49068035  1.89525349]
 [ 0.2935111  -0.38527099 -0.36726567  0.05388857 -0.03050685]
 [ 1.02687751 -1.0205883  -0.05963054  1.86996511 -0.48568312]
 [-1.17758131  1.08224614  0.62710458 -0.23134112  0.36333312]
 [-1.8826224  -0.70551637  0.09074075  1.10122071 -0.07975198]]


lower triangular matrix - 
 [[ 0.04163482  0.          0.          0.          0.        ]
 [ 0.2935111  -0.38527099  0.          0.          0.        ]
 [ 1.02687751 -1.0205883  -0.05963054  0.          0.        ]
 [-1.17758131  1.08224614  0.62710458 -0.23134112  0.        ]
 [-1.8826224  -0.70551637  0.09074075  1.10122071 -0.07975198]]


Upper triangular matrix - 
 [[ 0.04163482  0.84694284  1.27184552  0.49068035  1.89525349]
 [ 0.         -0.38527099 -0.36726567  0.05388857 -0.03050685]
 [ 0.          0.         -0.05963054  1.86996511 -0.48568312]
 [ 0.          0.          0.         -0.23134112  0.36333312]
 [ 0.          0.          0.          0.         -0.07975198]]
```

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)

In [599]:

```
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[599]:

```
(array([[1, 2],
        [3, 4],
        [5, 6],
        [1, 1],
        [1, 1]]), (5, 2), numpy.ndarray, dtype('int32'))
```

In [486]:

```
np.full((5,5) , 8)
```

Out[486]:

```
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 [490]:

```
M
```

Out[490]:

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

In [491]:

```
M.flatten()
```

Out[491]:

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

Matrix Addition

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

In [397]:

```
#*********************************************************#
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)](https://www.youtube.com/watch?v=7jb_AO_hRc8&list=PLmdFyQYShrjcoVkhCCIwxNj9N4rW1-T5I&index=8)

In [398]:

```
#********************************************************#
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=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5I&index=9 (https://www.youtube.com/watch?v=4lHyTQH1iS8&list=PLmdFyQYShrjcoVkhCClwxNj9N4rW1-T5I&index=9)

In [90]:

```
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 [425]:

```
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 [404]:

```
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 [405]:

```
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 [602]:

```
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 [407]:

```
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 [409]:

```
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 [411]:

```
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 [413]:

```
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 [414]:

```
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 [415]:

```
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 [416]:

```
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)

In [536]:

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

Out[536]:

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

In [537]:

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

Out[537]:

```
array([[10, 10, 10],
       [10, 10, 10]])
```

In [538]:

```
np.kron(M1,M2)
```

Out[538]:

```
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 [418]:

```
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 [423]:

```
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]
```

Tensor

What is Tensor :

- 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=uaQeXi4E7gA (https://www.youtube.com/watch?v=uaQeXi4E7gA)

In [93]:

```
# 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[93]:

```
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 [94]:

```
T2 = np.array([
  [[0,0,0] , [0,0,0] , [0,0,0]],
  [[1,1,1] , [1,1,1] , [1,1,1]],
  [[2,2,2] , [2,2,2] , [2,2,2]]

])

T2
```

Out[94]:

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

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

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

Tensor Addition

In [95]:

```
A = T1+T2
A
```

Out[95]:

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

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

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

In [96]:

```
np.add(T1,T2)
```

Out[96]:

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

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

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

Tensor Subtraction

In [97]:

```
S = T1-T2
S
```

Out[97]:

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

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

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

In [98]:

```
np.subtract(T1,T2)
```

Out[98]:

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

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

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

Tensor Element-Wise Product

In [511]:

```
P = T1*T2
P
```

Out[511]:

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

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

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

In [512]:

```
np.multiply(T1,T2)
```

Out[512]:

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

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

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

Tensor Element-Wise Division

In [513]:

```
D = T1/T2
D
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWa
rning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

Out[513]:

```
array([[[ inf,  inf,  inf],
        [ inf,  inf,  inf],
        [ inf,  inf,  inf]],

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

       [[ 50., 100., 150.],
        [200., 250., 300.],
        [350., 400., 450.]]])
```

In [514]:

```
np.divide(T1,T2)
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWa
rning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

Out[514]:

```
array([[[ inf,  inf,  inf],
        [ inf,  inf,  inf],
        [ inf,  inf,  inf]],

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

       [[ 50., 100., 150.],
        [200., 250., 300.],
        [350., 400., 450.]]])
```

Tensor Dot Product

In [523]:

```
T1
```

Out[523]:

```
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 [524]:

```
T2
```

Out[524]:

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

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

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

In [533]:

```
np.tensordot(T1,T2)
```

Out[533]:

```
array([[  63,   63,   63],
       [ 630,  630,  630],
       [6300, 6300, 6300]])
```

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=a2z7sZ4MSqo (https://www.youtube.com/watch?v=a2z7sZ4MSqo)

In [108]:

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

Out[108]:

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

In [109]:

```
B = np.random.random((3,1))
B
```

Out[109]:

```
array([[0.4760653 ],
       [0.69011595],
       [0.27072528]])
```

In [110]:

```
# Ist Method
X = np.dot(np.linalg.inv(A) , B)
X
```

Out[110]:

```
array([[-1.99693625e+15],
       [ 3.99387250e+15],
       [-1.99693625e+15]])
```

In [111]:

```
# 2nd Method
X = np.matmul(np.linalg.inv(A) , B)
X
```

Out[111]:

```
array([[-1.99693625e+15],
       [ 3.99387250e+15],
       [-1.99693625e+15]])
```

In [112]:

```
# 3rd Method
X = np.linalg.inv(A)@B
X
```

Out[112]:

```
array([[-1.99693625e+15],
       [ 3.99387250e+15],
       [-1.99693625e+15]])
```

In [113]:

```
# 4th Method
X = np.linalg.solve(A,B)
X
```

Out[113]:

```
array([[-1.99693625e+15],
       [ 3.99387250e+15],
       [-1.99693625e+15]])
```