Assignment 2:1.

- **Prove properties of matrix multiplication**
- **Write notebook in a structured manner**
- **Calculate inverse of a matrix using numpy (inbuilt api and/or manual coding) - Show how numpy is faster than traditional looping :**

You have to print time for both cases

Use a large sized matrix (10000 x 10000) or something even larger. You can use any example

In [1]:

```
import numpy as np
import random
import time
```

In [2]:

```
A = np.array([[np.random.randint(0,100) for j in range(10)] for i in range(10)])
B =  np.array([[np.random.randint(0,100) for j in range(10)] for i in range(10)])
C = np.array([[np.random.randint(0,100) for j in range(10)] for i in range(10)])
```

In [4]:

```
def check(X,Y):
    if X.shape != Y.shape:
        return False
    else:
        for i in range(X.shape[0]):
            for j in range(X.shape[1]):
                if X[i][j] != Y[i][j]:
                    return False
        return True
```

Associative Law

A(BC) = (AB)C

In [5]:

```
a = np.matmul(A, np.matmul(B,C))
b = np.matmul(np.matmul(A,B),C)
if check(a,b):
    print("Matrices are Associative")
else:
    print("Matricesare not Associative")
```

```
Matrices are Associative
```

Distributive

A(B + C) == (AB) + (AC

In [7]:

```
abc = np.matmul(A,np.add(B,C))
ab_ac = np.add(np.matmul(A,B),np.matmul(A,C))
if check(abc,ab_ac):
    print("Matrices are Distributive!")
else:
    print("Matrices are not Distributive")
```

```
Matrices are Distributive!
```

Non - commutative

AB != BA

In [6]:

```
ab = np.matmul(A,B)
ba = np.matmul(B,A)
if check(ab,ba):
    print("Matrices are Commutative")
else:
    print("Matrices are Non Commutative")
```

```
Matrices are Non Commutative
```

2. Inverse of Matrix using Numpy:

In [14]:

```
a_inv = np.linalg.inv(A)
b_inv = np.linalg.inv(B)
c_inv = np.linalg.inv(C)
```

In [15]:

```
a_inv
```

Out[15]:

```
array([[-4.58737080e-03, -5.15340614e-04, -8.20577804e-04,
         6.01661981e-03, -3.57587729e-03,  3.80775779e-03,
        -1.11669927e-03,  5.34024573e-03, -2.64638370e-03,
        -1.35214563e-03],
       [ 9.05609807e-03, -1.13423123e-02,  3.70969090e-03,
         1.42099686e-03,  1.38414581e-02,  1.85705451e-03,
         9.44279913e-03, -8.78084833e-03, -1.39052862e-02,
        -1.84632598e-03],
       [ 6.27794942e-04, -4.77409999e-03, -1.05876997e-02,
         2.30597963e-02, -2.89867832e-02, -5.88055149e-03,
         2.84967645e-03,  3.54984600e-03,  1.70724187e-02,
        -6.41363329e-03],
       [ 3.43651732e-03, -3.28594427e-03,  4.78148553e-04,
         1.64094526e-02,  2.40134903e-04, -9.29514863e-03,
        -5.97992811e-03, -2.52514091e-03, -9.59112939e-03,
         5.30520732e-03],
       [ 3.88663339e-03, -7.72055237e-03,  2.42879076e-03,
        -1.93680328e-02,  4.82157440e-02,  7.73312390e-03,
        -1.51544588e-02,  2.37832936e-03, -3.01685617e-02,
         2.57977452e-02],
       [-7.32009351e-03,  2.65874080e-02,  4.05035069e-03,
        -2.51928267e-02, -8.14683789e-03,  1.73113561e-02,
        -1.72175263e-03, -5.32704578e-03,  2.33037322e-02,
        -1.23485045e-02],
       [ 5.55913239e-03,  1.56255541e-02,  5.36128210e-03,
        -3.00840434e-03, -5.01000259e-02, -8.55732910e-03,
         9.55990139e-03,  3.31746683e-03,  3.04178674e-02,
        -1.50888722e-02],
       [-5.28500799e-03,  1.99120399e-02,  1.39845387e-02,
        -1.58302280e-02, -2.27909168e-02,  2.94673743e-03,
        -8.74347939e-03, -2.48133104e-05,  2.98556487e-02,
        -7.97150298e-03],
       [ 3.12100979e-03, -3.02796596e-02, -1.56252103e-02,
         6.47824586e-03,  7.05545348e-02,  3.64659028e-03,
         1.10144117e-03,  5.75100517e-03, -5.17934646e-02,
         1.61802728e-02],
       [-5.51796895e-03,  5.14256687e-03, -4.82788336e-04,
         1.79237759e-02, -4.31864485e-02, -1.96515781e-02,
```

```
         9.46416642e-03, -1.56920577e-03,  2.41635865e-02,
        -1.16171836e-03]])
```

In [16]:

```
b_inv
```

Out[16]:

```
array([[-0.00447291, -0.00095446,  0.0089484 ,  0.00447938, -0.00359636,
         0.00072199,  0.00396691, -0.01145415,  0.00588158, -0.00260866],
       [-0.00783593, -0.00026571,  0.0004922 ,  0.00450548,  0.00254425,
        -0.0097444 , -0.00158963,  0.00460866,  0.0033078 ,  0.00703625],
       [-0.01160113, -0.00764922, -0.00592041,  0.00081809,  0.01965892,
         0.00214419, -0.01787863,  0.01995029,  0.00138735, -0.00303233],
       [ 0.00930569,  0.00483851,  0.00317697,  0.00089843, -0.01476893,
         0.00339886,  0.00656837, -0.00146564, -0.00815637, -0.00107341],
       [ 0.00993179,  0.00699153, -0.015157  ,  0.00165572, -0.01051794,
         0.01828866, -0.01649258,  0.00919438, -0.00790089,  0.00353224],
       [ 0.00392929, -0.00288489, -0.01738539,  0.00585294,  0.01466852,
         0.00856349, -0.01656814,  0.01530406, -0.00174617, -0.00685375],
       [-0.00279399, -0.00479355,  0.00583899, -0.00118808,  0.01594523,
        -0.00680342,  0.00439689, -0.00442916, -0.00337188,  0.00135658],
       [ 0.00784899, -0.00304481,  0.00712694, -0.00507683, -0.01300316,
        -0.00127422,  0.02255173, -0.01431985,  0.00120903,  0.00353147],
       [ 0.0104072 ,  0.00273301,  0.00725351, -0.00928733, -0.00951022,
        -0.00125832,  0.00486684, -0.00724401,  0.00297085, -0.00037763],
       [-0.02011102,  0.00843198,  0.01000306, -0.00057969,  0.00188421,
        -0.01049694,  0.00831953, -0.00719623,  0.00837037,  0.00180178]])
```

In [17]:

```
c_inv
```

Out[17]:

```
array([[ 0.0283443 , -0.00687736,  0.00671273,  0.01053384, -0.01369639,
         0.03602635, -0.03386528, -0.00519058, -0.02237197,  0.00330411],
       [ 0.04231502, -0.01534499, -0.01695095,  0.01975467, -0.01870402,
         0.05685648, -0.04188124, -0.00754651, -0.02778872,  0.00164937],
       [ 0.02944295, -0.00542937, -0.00525842,  0.00262609, -0.02055199,
         0.03739563, -0.02385757, -0.00984021, -0.0200052 ,  0.01546528],
       [ 0.02320986, -0.00187456, -0.0002216 ,  0.00977176, -0.00725061,
         0.01852397, -0.0109768 , -0.01427495, -0.02055105,  0.00546282],
       [-0.01787132,  0.0022093 ,  0.00517478, -0.01147   ,  0.01304863,
        -0.02672216,  0.03229081,  0.00138296,  0.01275133, -0.00307613],
       [-0.06051131,  0.00600193,  0.00723742, -0.00960707,  0.03098647,
        -0.07450217,  0.0635402 ,  0.0232082 ,  0.03404741, -0.01508696],
       [-0.06384361,  0.02486575,  0.01135513, -0.0163986 ,  0.0265357 ,
        -0.07848357,  0.04557069,  0.03118158,  0.04758208, -0.02116479],
       [-0.1041832 ,  0.02935845,  0.02890716, -0.02141314,  0.0412208 ,
        -0.12206515,  0.08778994,  0.0322691 ,  0.07206104, -0.03056178],
       [-0.01395094,  0.00949067,  0.00404315, -0.00609208,  0.00896087,
        -0.00513911, -0.00043192,  0.00022982,  0.00924932, -0.00220615],
       [ 0.06994899, -0.01946963, -0.02609398,  0.00907701, -0.026658  ,
         0.0708156 , -0.05210675, -0.02719244, -0.03632123,  0.0295969 ]])
```

Numpy Faster than traditional loops:

In [19]:

```
X= np.array([[np.random.randint(0,100)for j in range(1000)]for i in range(1000)])
Y=np.array([[np.random.randint(0,100)for j in range(1000)] for i in range(1000)])
```

Traditional Looping

In [20]:

```
start = time.time()
```

```
result = np.array([[0 for i in range(1000)] for j in range(1000)])
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        result[i][j] = X[i][j] + Y[i][j]
time_elapsed = time.time() - start
time_elapsed
```

Out[20]:

```
1.0311837196350098
```

Numpy Array

In [21]:

```
start_numpy = time.time()
result1 = np.add(X,Y)
elapsed_time = time.time() - start_numpy
elapsed_time
```

Out[21]:

```
0.0019757747650146484
```

In [ ]:

```

```