Assignment 2:

1. Prove properties of matrix multiplication
2. Write notebook in a structured manner
3. Calculate inverse of a matrix using numpy (inbuilt api and/or manual coding)
4. 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 [35]:
```
import numpy as np
import time
import random
```

1. Properties of matrix multiplication:

In [11]:
```
A = np.array([[np.random.randint(0,100) for j in range(10)]for i in rang
e(10)])
B = np.array([[np.random.randint(0,100) for j in range(10)]for i in rang
e(10)])
C = np.array([[np.random.randint(0,100) for j in range(10)]for i in rang
e(10)])
```

In [12]:
```
def proof(A,B):
    if A.shape != B.shape:
        return False
    else:
        for i in range(A.shape[0]):
            for j in range(A.shape[1]):
                if A[i][j] != B[i][j]:
                    return False
        return True
```

Associative

A(BC) = (AB)C

In [39]:
```
a = np.matmul(A,np.matmul(B,C))
b = np.matmul(np.matmul(A,B),C)
if proof(a,b):
    print("Matrices are Associative!")
else:
    print("Matrices are not Associative!")
```

```
Matrices are Associative!
```

Distributive

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

In [38]:
```
abc = np.matmul(A,np.add(B,C))
abac = np.add(np.matmul(A,B),np.matmul(A,C))
if proof(abc,abac):
    print("Matrices are Distributive!")
else:
    print("Matrices are not Distributive!")
```

```
Matrices are Distributive!
```

Non-commutative

AB != BA

In [37]:
```
ab = np.matmul(A,B)
ba = np.matmul(B,A)
if proof(ab,ba):
    print("Matrices are Commutative!")
else:
    print("Matrices are Non-Commutative!")
```

```
Matrices are Non-Commutative!
```

2. Inverse of matric using numpy:

In [40]:
```
a_inv = np.linalg.inv(A)
print(a_inv)
```

```
[[-0.03213879 -0.02092325  0.00041342  0.03861828  0.00875809  0.0018934
9
   0.0019616  -0.01816177  0.00674989  0.016823  ]
 [-0.13481586 -0.19682148  0.08470027  0.15931876 -0.02322925  0.0494137
4
   0.0379285  -0.03862529  0.05222222  0.01818692]
 [ 0.16318488  0.20913688 -0.08535443 -0.20336091  0.00297078 -0.0388141
8
  -0.03248468  0.06059391 -0.05327425 -0.0328155 ]
 [ 0.10798723  0.13202365 -0.05224322 -0.12340043  0.01353329 -0.0340625
9
  -0.02521809  0.03397224 -0.04182525 -0.01176434]
 [-0.19076334 -0.23964452  0.09535905  0.23396193 -0.01718411  0.0422764
9
   0.0386561  -0.05829026  0.07026756  0.03894683]
 [-0.21174359 -0.29023542  0.12096714  0.26987078 -0.01707597  0.0593089
2
   0.04072205 -0.07458142  0.08081002  0.03876697]
 [ 0.0741537   0.1069694  -0.0349402  -0.09601027  0.01220462 -0.0200462
8
  -0.02333061  0.02790297 -0.03217117 -0.01561798]
 [ 0.1373109   0.19279919 -0.09103944 -0.16944153  0.01775255 -0.0403287
  -0.0247424   0.04599158 -0.05106707 -0.02784652]
 [-0.13921282 -0.18762434  0.08077384  0.16028485 -0.01742795  0.0413148
2
   0.03653628 -0.04628791  0.06231542  0.01671094]
 [ 0.24329472  0.32600385 -0.13001566 -0.29110217  0.02404252 -0.0628390
8
  -0.05201983  0.07352278 -0.097741   -0.04362312]]
```

In [42]:
```
b_inv = np.linalg.inv(B)
print(b_inv)
```

```
[[-9.20162020e-04 -3.77031960e-02  2.69989878e-02  2.86892297e-02
  -1.80879726e-03  3.86466055e-03 -1.05425232e-03 -2.72631510e-02
   1.11090302e-03  2.74991520e-03]
 [-2.91120497e-03  1.17052189e-02  9.40213635e-04 -8.37282338e-03
   3.85925000e-04 -8.70190701e-03  1.15009178e-02  3.97709002e-03
  -4.25078956e-04 -7.37029788e-04]
 [-5.64135291e-03  6.02486208e-03 -1.43609198e-02  9.20564377e-03
   1.43695005e-02  6.15219412e-03 -1.40699119e-02  8.48889454e-03
  -7.49896527e-03 -1.23025200e-02]
 [-1.38513153e-02 -3.97254884e-02 -5.62257902e-03 -3.52912507e-02
   4.23137621e-02 -3.02760963e-03  1.47393201e-02  2.61483025e-02
  -1.12034395e-02  9.90589890e-03]
 [ 6.08057709e-04  1.52133230e-02 -8.89920224e-03 -1.64370086e-02
  -1.74759607e-03 -9.23333515e-05  1.82587616e-02  3.81868935e-03
   3.34817481e-04 -6.54500781e-03]
 [ 3.12216453e-02 -7.63378446e-03  3.07170257e-02  6.06635121e-02
  -4.49001674e-02 -3.02189203e-03 -2.91777375e-02 -4.57018098e-02
   2.17932404e-02  5.11568680e-03]
 [-7.33552354e-03  9.75620569e-03 -1.40356629e-02 -2.06438481e-02
   1.56512001e-02  4.82741469e-03  2.54719569e-03  1.12168068e-02
  -2.35504157e-03  3.51912314e-03]
 [ 5.03751942e-03 -1.20287963e-02 -6.23838049e-04  1.80834229e-03
  -1.33854768e-03 -5.83381767e-04 -6.16855250e-03  4.95384101e-03
  -4.71816353e-03  1.39742843e-02]
 [ 4.00907313e-03  3.28224832e-02  1.75598812e-03  1.78221145e-02
  -3.46122442e-02 -3.43722393e-03  3.62887907e-03 -8.27336849e-03
   1.61897305e-02 -1.72883008e-02]
 [-9.11985565e-03  2.14406663e-02 -1.63663453e-02 -3.59345244e-02
   1.47953008e-02  6.78057012e-03  4.76603378e-03  2.65131493e-02
  -1.12215321e-02 -1.58709252e-03]]
```

In [43]:
```
c_inv = np.linalg.inv(C)
print(c_inv)
```

```
[[ 6.57437743e-03  6.83522585e-03  1.90721705e-04  6.69044527e-03
  -1.25602565e-02  6.52841053e-03 -1.37951330e-02  2.59571612e-03
  -3.30232386e-03  3.87835872e-03]
 [-3.41389187e-03  8.62112243e-04 -3.62923403e-02  1.08584412e-02
  -1.02971308e-04  1.23411723e-02  5.60241792e-03  3.93550523e-03
   3.71553327e-02 -1.91286471e-02]
 [ 7.07257020e-03  1.25589201e-02 -1.58101991e-02  3.61048665e-03
  -5.01675694e-03  3.15123204e-03 -2.70688080e-03 -6.16135064e-03
   5.13377294e-03  4.72967775e-03]
 [ 4.35296769e-03 -7.10416426e-03  3.75977457e-02 -1.73839716e-02
   1.16291590e-02 -2.42370239e-02  9.92012435e-03 -4.33864303e-03
  -2.56314274e-02  6.95921815e-03]
 [ 1.66417370e-02  6.19462450e-03  2.70361575e-02 -8.35095215e-03
  -9.56772608e-03  5.67318972e-03 -1.50238070e-02 -9.10420553e-03
  -3.85050448e-02  2.30744800e-02]
 [-9.82098633e-05 -4.55201285e-03 -6.19661717e-03 -3.01734052e-03
   1.48365355e-02 -1.69441949e-02  3.11529831e-03  6.91307580e-03
   8.87630128e-03 -6.62027993e-03]
 [-3.64379670e-04 -2.42106630e-04  1.23759141e-02 -1.32690814e-02
   7.58136590e-04 -5.34680249e-03 -6.34259539e-03  1.11090364e-03
  -5.02914021e-03  1.64218455e-02]
 [-5.36314747e-03 -4.88821402e-03 -1.13533318e-02  1.56407968e-02
  -1.23317047e-02  2.16136358e-02  4.63766383e-03  3.98891179e-03
   5.24526213e-03 -9.30322530e-03]
 [-1.49711173e-02 -7.45117066e-03  7.28636110e-04  1.31467166e-02
  -4.10943353e-05  1.11146439e-02  2.94584985e-03 -7.96263040e-04
   4.28916596e-03 -5.19517941e-03]
 [-8.71324916e-03  1.64948194e-03 -1.22410199e-02  2.03983876e-03
   9.12253211e-03  1.60595456e-03  1.16184103e-02  3.13298721e-03
   1.04161087e-02 -1.49789885e-02]]
```

3. Numpy faster than traditional looping?

In [45]:
```
X = np.array([[np.random.randint(0,100) for j in range(1000)]for i in ra
nge(1000)])
Y = np.array([[np.random.randint(0,100) for j in range(1000)]for i in ra
nge(1000)])
```

Traditional looping

In [46]:
```
start = time.time()
res = 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]):
        res[i][j] = X[i][j] + Y[i][j]

loop_time = time.time()-start
print(loop_time)
```

```
1.6967103481292725
```

Numpy Array

In [47]:
```
start_time = time.time()
res_np = np.add(X,Y)

np_time = time.time()-start_time
print(np_time)
```

```
0.0029938220977783203
```