In [1]:

```
import numpy as np
import time

np.random.seed(24787)
a = np.random.randint(8, size=(3,4,4))
#Question1.a "Print a"
print(a)
#Question1.a "Print shape of a"
print(a.shape)
#Question1.a "Locate the locations of all the fours in the array a print out the
find = np.where(a == 4)
print(find)
#Question1.b
b = np.tile(a, (2,2))
print(b)
print(b.shape)
#Question1.c "Sum of c in depth of b"
#Size should be a depth of 1, 8 by 8
c = b.sum(axis=0)
print(c)
print(c.shape)
```

```
[[[2 6 4 1]
  [0 4 4 3]
  [6 6 1 2]
  [7 0 6 5]]

 [[1 3 3 7]
  [4 7 2 5]
  [0 4 6 7]
  [5 5 7 1]]

 [[7 2 4 5]
  [6 7 7 0]
  [6 2 0 4]
  [2 0 7 6]]]
(3, 4, 4)
(array([0, 0, 0, 1, 1, 2, 2], dtype=int64), array([0, 1, 1, 1, 2, 0, 2], dtyp
e=int64), array([2, 1, 2, 0, 1, 2, 3], dtype=int64))
[[[2 6 4 1 2 6 4 1]
  [0 4 4 3 0 4 4 3]
  [6 6 1 2 6 6 1 2]
  [7 0 6 5 7 0 6 5]
  [2 6 4 1 2 6 4 1]
  [0 4 4 3 0 4 4 3]
  [6 6 1 2 6 6 1 2]
  [7 0 6 5 7 0 6 5]]

 [[1 3 3 7 1 3 3 7]
  [4 7 2 5 4 7 2 5]
  [0 4 6 7 0 4 6 7]
  [5 5 7 1 5 5 7 1]
  [1 3 3 7 1 3 3 7]
  [4 7 2 5 4 7 2 5]
  [0 4 6 7 0 4 6 7]
  [5 5 7 1 5 5 7 1]]

 [[7 2 4 5 7 2 4 5]
  [6 7 7 0 6 7 7 0]
  [6 2 0 4 6 2 0 4]
  [2 0 7 6 2 0 7 6]
  [7 2 4 5 7 2 4 5]
  [6 7 7 0 6 7 7 0]
  [6 2 0 4 6 2 0 4]
  [2 0 7 6 2 0 7 6]]]
(3, 8, 8)
[[10 11 11 13 10 11 11 13]
 [10 18 13  8 10 18 13  8]
 [12 12  7 13 12 12  7 13]
 [14  5 20 12 14  5 20 12]
 [10 11 11 13 10 11 11 13]
 [10 18 13  8 10 18 13  8]
 [12 12  7 13 12 12  7 13]
 [14  5 20 12 14  5 20 12]]
(8, 8)
```

In [2]:

```
#Question1.d (Complete the multiplication)
np.random.seed(24787)
#a = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
#b = np.array([[10, 11, 12],[13, 14, 15],[16, 17, 18]])
#a = np.random.random((100,100))
#b = np.random.random((100,100))
a = np.random.random((1000,1000))
b = np.random.random((1000,1000))
print(type(a))

def matmal(Matrix1=None,Matrix2=None):
    start = time.time()
    na,ma = a.shape
    nb,mb = b.shape
    NewMatrixRow = []
    NewMatrixCol = []
    if Matrix1.shape[1] != Matrix2.shape[0]:
        print("Matrices are not the right size")
        new = None
    else:
        print("Good to Go")
        for cvalue in range(mb):
            for rvalue in range(na):
                desiredvalue = np.dot(a[cvalue,:],b[:,rvalue])
                NewMatrixRow.append(desiredvalue)
            NewMatrixCol.append(NewMatrixRow)
            NewMatrixRow = []
    new2 = np.array(NewMatrixCol)
    print(new2)
    slow = time.time() - start

    return new2,slow

cQuestion,slow = matmal(a,b)

print(cQuestion)
print("Time required to execute: " + str(slow) + "s")
```

```
<class 'numpy.ndarray'>
Good to Go
[[262.02681889 250.35010895 255.20209698 ... 255.11959541 248.13659427
  246.60845719]
 [267.28382531 248.10413271 253.0417709  ... 255.93196541 253.71831423
  246.48668303]
 [268.78477885 252.60592752 268.07751687 ... 262.21855106 259.70257958
  254.42145539]
 ...
 [256.28928623 241.29542559 253.63761207 ... 255.58972957 248.44848129
  245.45765948]
 [261.245718   248.87670911 259.54001149 ... 257.58727381 252.51065398
  250.76435101]
 [256.55021563 242.88337935 249.70633094 ... 251.16608605 245.52951864
  237.41175273]]
[[262.02681889 250.35010895 255.20209698 ... 255.11959541 248.13659427
  246.60845719]
 [267.28382531 248.10413271 253.0417709  ... 255.93196541 253.71831423
  246.48668303]
 [268.78477885 252.60592752 268.07751687 ... 262.21855106 259.70257958
  254.42145539]
 ...
 [256.28928623 241.29542559 253.63761207 ... 255.58972957 248.44848129
  245.45765948]
 [261.245718   248.87670911 259.54001149 ... 257.58727381 252.51065398
  250.76435101]
 [256.55021563 242.88337935 249.70633094 ... 251.16608605 245.52951864
  237.41175273]]
Time required to execute: 4.353710174560547s
```

In [3]:

```python
np.random.seed(24787)
a = np.random.random((1000,1000))
b = np.random.random((1000,1000))

start = time.time()
cCheck = a@b
print(cCheck)
print("Time required to execute: " + str((time.time() - start)) + "s")
print(bool(np.sum(cQuestion) == np.sum(cCheck)))
```

```
[[262.02681889 250.35010895 255.20209698 ... 255.11959541 248.13659427
  246.60845719]
 [267.28382531 248.10413271 253.0417709  ... 255.93196541 253.71831423
  246.48668303]
 [268.78477885 252.60592752 268.07751687 ... 262.21855106 259.70257958
  254.42145539]
 ...
 [256.28928623 241.29542559 253.63761207 ... 255.58972957 248.44848129
  245.45765948]
 [261.245718   248.87670911 259.54001149 ... 257.58727381 252.51065398
  250.76435101]
 [256.55021563 242.88337935 249.70633094 ... 251.16608605 245.52951864
  237.41175273]]
Time required to execute: 0.017966508865356445s
True
```

In [4]:

```python
#The time required for the @ symbol is faster because numpy is written in
#C which is computationally faster then the code I have written in Python
#because the @ command is written in CPython
```