In [1]:
```
#Question 1a
import numpy as np
np.random.seed(0)
a=np.reshape(np.random.randint(0, 11, size=360, dtype='int'),(3,12,10))
print("a array:"+str(a))
print("Shape of a:"+str(a.shape))
```

```
a array:[[[ 5  0  3  3  7  9  3  5  2  4]
  [ 7  6  8  8 10  1  6  7  7  8]
  [ 1  5  9  8  9  4  3  0  3  5]
  [ 0  2  3  8  1  3  3  3  7  0]
  [ 1  9  9  0 10  4  7  3  2  7]
  [ 2  0  0  4  5  5  6  8  4  1]
  [ 4  9 10 10  8  1  1  7  9  9]
  [ 3  6  7  2  0  3  5  9 10  4]
  [ 4  6  4  4  3  4  4  8  4  3]
  [10  7  5  5  0  1  5  9  3  0]
  [ 5  0  1  2  4  2  0  3  2 10]
  [ 0  7  5  9  0 10  2 10  7  2]]

 [[ 9  2  3  3  2  3  4  1  2  9]
  [10  1  4 10  6  8  2  3  0  0]
  [ 6  0  6  3 10  3  8  8  8  2]
  [ 3  2  0  8  8  3  8 10  2  8]
  [ 4  3  0  4  3  6  9  8  0  8]
  [ 5  9  0  9  6  5  3  1  8  0]
  [ 4  9  6  5  7  8  8  9  2  8]
  [ 6  6  9  1  6  8  8  3  2  3]
  [10  6  3  6  5  7  0  8  4 10]
  [ 6  5  8  2  3  9  7  5  3  4]
  [ 5  3  3  7  9  9  9  7  3 10]
  [ 2  3  9 10  7  7  5  1  2  2]]

 [[ 8  1  5  8  4  0  2  5  5  0]
  [ 8  1  1  0  3  8  8  4  4  0]
  [ 9  3  7  3  2  1  1  2  1  4]
```

```
  [ 2  5  5  5  2  5  7  7  6  1]
  [ 6  7  2  3  1  9  5  9  9  2]
  [ 0  9  1  9 10  0  6  0 10  4]
  [ 8  4  3  3  8  8  7  0  3  8]
  [ 7  7 10  1  8  4  7  0  4  9]
  [ 0 10  6  4  2  4  6  3 10  3]
  [ 7  8  5  0  8  5  4  7 10 10]
  [ 4  1  3  3  9  2  5  2  3  5]
  [10  7  2  7  1  6 10  5  0  0]]]
Shape of a:(3, 12, 10)
```

In [2]:

```
#Question 1b
b=a[0:3,2:10,1:9]
print("b array:"+str(b))
print("Shape of b:"+str(b.shape))
```

```
b array:[[[ 5  9  8  9  4  3  0  3]
  [ 2  3  8  1  3  3  3  7]
  [ 9  9  0 10  4  7  3  2]
  [ 0  0  4  5  5  6  8  4]
  [ 9 10 10  8  1  1  7  9]
  [ 6  7  2  0  3  5  9 10]
  [ 6  4  4  3  4  4  8  4]
  [ 7  5  5  0  1  5  9  3]]

 [[ 0  6  3 10  3  8  8  8]
  [ 2  0  8  8  3  8 10  2]
  [ 3  0  4  3  6  9  8  0]
  [ 9  0  9  6  5  3  1  8]
  [ 9  6  5  7  8  8  9  2]
  [ 6  9  1  6  8  8  3  2]
  [ 6  3  6  5  7  0  8  4]
  [ 5  8  2  3  9  7  5  3]]

 [[ 3  7  3  2  1  1  2  1]
  [ 5  5  5  2  5  7  7  6]
  [ 7  2  3  1  9  5  9  9]
  [ 9  1  9 10  0  6  0 10]
  [ 4  3  3  8  8  7  0  3]
```

```
  [ 7 10  1  8  4  7  0  4]
  [10  6  4  2  4  6  3 10]
  [ 8  5  0  8  5  4  7 10]]]
Shape of b:(3, 8, 8)
```

In [9]:
```
#Question 1c
c=np.sum(b,axis=2)
print("c array:"+str(c))
print("Shape of c:"+str(c.shape))
```

```
c array:[[41 30 44 32 55 42 37 35]
 [46 41 33 41 54 43 39 42]
 [20 42 45 45 36 41 45 47]]
Shape of c:(3, 8)
```

In [4]:
```
#Question 1d

d=np.reshape(b,(3*8*8,1))
print("d array:"+str(d))
print("Shape of array:"+str(d.shape))
print("Size of array:"+str(np.size(d)))
```

```
d array:[[ 5]
 [ 9]
 [ 8]
 [ 9]
 [ 4]
 [ 3]
 [ 0]
 [ 3]
 [ 2]
 [ 3]
 [ 8]
 [ 1]
 [ 3]
 [ 3]
 [ 3]
 [ 7]
 [ 9]
```

```
[ 9]
[ 0]
[10]
[ 4]
[ 7]
[ 3]
[ 2]
[ 0]
[ 0]
[ 4]
[ 5]
[ 5]
[ 6]
[ 8]
[ 4]
[ 9]
[10]
[10]
[ 8]
[ 1]
[ 1]
[ 7]
[ 9]
[ 6]
[ 7]
[ 2]
[ 0]
[ 3]
[ 5]
[ 9]
[10]
[ 6]
[ 4]
[ 4]
[ 3]
[ 4]
[ 4]
[ 8]
[ 4]
```

```
[ 7]
[ 5]
[ 5]
[ 0]
[ 1]
[ 5]
[ 9]
[ 3]
[ 0]
[ 6]
[ 3]
[10]
[ 3]
[ 8]
[ 8]
[ 8]
[ 2]
[ 0]
[ 8]
[ 8]
[ 3]
[ 8]
[10]
[ 2]
[ 3]
[ 0]
[ 4]
[ 3]
[ 6]
[ 9]
[ 8]
[ 0]
[ 9]
[ 0]
[ 9]
[ 6]
[ 5]
[ 3]
[ 1]
```

```
[ 8]
[ 9]
[ 6]
[ 5]
[ 7]
[ 8]
[ 8]
[ 9]
[ 2]
[ 6]
[ 9]
[ 1]
[ 6]
[ 8]
[ 8]
[ 3]
[ 2]
[ 6]
[ 3]
[ 6]
[ 5]
[ 7]
[ 0]
[ 8]
[ 4]
[ 5]
[ 8]
[ 2]
[ 3]
[ 9]
[ 7]
[ 5]
[ 3]
[ 3]
[ 7]
[ 3]
[ 2]
[ 1]
[ 1]
```

```
[ 2]
[ 1]
[ 5]
[ 5]
[ 5]
[ 2]
[ 5]
[ 7]
[ 7]
[ 6]
[ 7]
[ 2]
[ 3]
[ 1]
[ 9]
[ 5]
[ 9]
[ 9]
[ 9]
[ 1]
[ 9]
[10]
[ 0]
[ 6]
[ 0]
[10]
[ 4]
[ 3]
[ 3]
[ 8]
[ 8]
[ 7]
[ 0]
[ 3]
[ 7]
[10]
[ 1]
[ 8]
[ 4]
```

```
 [ 7]
 [ 0]
 [ 4]
 [10]
 [ 6]
 [ 4]
 [ 2]
 [ 4]
 [ 6]
 [ 3]
 [10]
 [ 8]
 [ 5]
 [ 0]
 [ 8]
 [ 5]
 [ 4]
 [ 7]
 [10]]
Shape of array:(192, 1)
Size of array:192
```

In [5]:

```
#Question 1e
np.random.seed(0)
import time
e=np.reshape(np.random.randint(0, 11, size=1000*1000, dtype='int'),(1000,1000))
f=np.reshape(np.random.randint(0, 11, size=1000*1000, dtype='int'),(1000,1000))
start_time=time.time()
M=e@f
end_time=time.time()
print("Execution time using @ operator:",end_time-start_time)
```

```
Execution time using @ operator: 3.729900598526001
```

In [6]:

```
def matmul(a,b):
    M=np.zeros((len(a),len(a)))
    for i in range(np.size(a,1)):
```

```
        for j in range(np.size(b,0)):
            for k in range(len(a)):
                M[i][j] +=a[i][k]*b[k][j]
    return M
start_time=time.time()
M=matmul(e,f)
end_time=time.time()
print("Execution time using matmul:",end_time-start_time)
```

```
Execution time using matmul: 1301.0255846977234
```

The built in function is faster than the maltiplication of matrices using matmul function. The performance of Python reduces when there are inner loops in the function.The loops take more time in memory allocation thus reducing the speed of hand-built functions.Built-in functions are often implemented using the best memory usage practices.

In [ ]:

In [ ]:

In [ ]: