In [1]:

```
###1 Create a null vector (zeros) of size 40
##(a) create in one dimension, as array A1
import numpy as np
A1 = np.zeros(40)
print(A1)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [2]:

```
##(b) create in two dimension with 8 rows and 5 columns, as array A2
A2 = np.zeros((8, 5))
print(A2)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [3]:

```
###(c) create a new array as A3, which is a copy of A2, except that all values o
A3 = A2.copy()
A3[:,2] = 1
print(A3)
```

```
[[0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]]
```

In [4]:

```
###2. Create a vector with values ranging from 1 to 40
##(a) create in one dimension, as array B1

B1 = np.arange(1,41)
print("Array B1:")
print(B1)
```

```
Array B1:
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40]
```

In [5]:

```
###(b) create in two dimenion with 8 rows and 5 columns, as array B2
B2 = B1.reshape(8,5)
print("Array B2:")
print(B2)
```

```
Array B2:
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]
 [26 27 28 29 30]
 [31 32 33 34 35]
 [36 37 38 39 40]]
```

In [6]:

```
###(c) create a new array B3 by adding B2 with A3 (,or increase all values of fo
B3 = A3 + B2
print("Array B3:")
print(B3)
```

```
Array B3:
[[ 1.  2.  4.  4.  5.]
 [ 6.  7.  9.  9. 10.]
 [11. 12. 14. 14. 15.]
 [16. 17. 19. 19. 20.]
 [21. 22. 24. 24. 25.]
 [26. 27. 29. 29. 30.]
 [31. 32. 34. 34. 35.]
 [36. 37. 39. 39. 40.]]
```

In [7]:

```
###(d) create a new array B4 with the values from B3 subtracted by 11
B4 = B3-11
print("Array B4:")
print(B4)
```

```
Array B4:
[[-10.  -9.  -7.  -7.  -6.]
 [ -5.  -4.  -2.  -2.  -1.]
 [  0.   1.   3.   3.   4.]
 [  5.   6.   8.   8.   9.]
 [ 10.  11.  13.  13.  14.]
 [ 15.  16.  18.  18.  19.]
 [ 20.  21.  23.  23.  24.]
 [ 25.  26.  28.  28.  29.]]
```

In [8]:

```
###3. Create a 12x12 matrix and fill it with a checkerboard pattern
##(a) first row and first column starts from 0
x1 = np.zeros((12,12),dtype=int)
x1[1::2,::2] = 1
x1[::2,1::2] = 1
print("Checkerboard pattern starts from 0:")
print(x1)
```

```
Checkerboard pattern starts from 0:
[[0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]]
```

In [9]:

```
##(b) first row and first column starts from 1
x2 = np.ones((12,12),dtype=int)
x2[1::2,::2] = 0
```

```
x2[::2,1::2] = 0
print("Checkerboard pattern starts from 1:")
print(x2)
```

```
Checkerboard pattern starts from 1:
[[1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1 0 1]]
```

In [10]:

```
##(c) with all numbers are 1 (integer)
x3 = np.ones((12,12),dtype=int)
print("All integer ones:")
print(x3)
```

```
All integer ones:
[[1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1]]
```

In [11]:

```
##(d) with all numbers are 1.0 (floating-point)
Y = np.ones((12,12),dtype=float)
print("All floating-point ones:")
print(Y)
```

```
All floating-point ones:
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
```

In [12]:

```
###4. Create 7x7 matrixes
##(a) P matrix with row values ranging from 0 to 6
arr = np.array([[0,1,2,3,4,5,6]],dtype=float)
P = np.repeat(arr,6, axis=0)
```

```
print("Matrix P:")
print(P)
```

```
Matrix P:
[[0. 1. 2. 3. 4. 5. 6.]
 [0. 1. 2. 3. 4. 5. 6.]
 [0. 1. 2. 3. 4. 5. 6.]
 [0. 1. 2. 3. 4. 5. 6.]
 [0. 1. 2. 3. 4. 5. 6.]
 [0. 1. 2. 3. 4. 5. 6.]]
```

In [13]:

```
##(b) Q matrix with column values ranging from 0 to 6

v1 = np.array([0,0,0,0,0,0,0], dtype=float)
v2 = np.array([1,1,1,1,1,1,1], dtype=float)
v3 = np.array([2,2,2,2,2,2,2], dtype=float)
v4 = np.array([3,3,3,3,3,3,3], dtype=float)
v5 = np.array([4,4,4,4,4,4,4], dtype=float)
v6 = np.array([5,5,5,5,5,5,5], dtype=float)
v7 = np.array([6,6,6,6,6,6,6], dtype=float)

Q = np.vstack([v1,v2,v3,v4,v5,v6,v7])
print("Matrix Q:")
print(Q)

##(c) R matrix with cumulative sum from previous row of matrix Q
R = Q.cumsum(axis=0)
print("Matrix R:")
print(R)

##(d) S matrix with cumulative sum from previous column of matrix Q
S = Q.cumsum(axis=1)
print("Matrix S:")
print(S)

##(e) T list with cumulative sun from previous value rolled out in order of row
T=np.cumsum(Q)
print("This is List T:")
print(T)
```

```
Matrix Q:
[[0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1. 1.]
 [2. 2. 2. 2. 2. 2. 2.]
 [3. 3. 3. 3. 3. 3. 3.]
 [4. 4. 4. 4. 4. 4. 4.]
 [5. 5. 5. 5. 5. 5. 5.]
 [6. 6. 6. 6. 6. 6. 6.]]
Matrix R:
[[ 0.  0.  0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.  1.  1.]
 [ 3.  3.  3.  3.  3.  3.  3.]
 [ 6.  6.  6.  6.  6.  6.  6.]
 [10. 10. 10. 10. 10. 10. 10.]
 [15. 15. 15. 15. 15. 15. 15.]
 [21. 21. 21. 21. 21. 21. 21.]]
Matrix S:
[[ 0.  0.  0.  0.  0.  0.  0.]
 [ 1.  2.  3.  4.  5.  6.  7.]
 [ 2.  4.  6.  8. 10. 12. 14.]
 [ 3.  6.  9. 12. 15. 18. 21.]
 [ 4.  8. 12. 16. 20. 24. 28.]
 [ 5. 10. 15. 20. 25. 30. 35.]
 [ 6. 12. 18. 24. 30. 36. 42.]]
This is List T:
[   0.    0.    0.    0.    0.    0.    0.    1.    2.    3.    4.    5.    6.    7.
    9.   11.   13.   15.   17.   19.   21.   24.   27.   30.   33.   36.   39.   42.
   46.   50.   54.   58.   62.   66.   70.   75.   80.   85.   90.   95.  100.  105.
  111.  117.  123.  129.  135.  141.  147.]
```

In [14]:

```
###5. Create a matrix, block segments, and integrations
##(a) create matrix X of 3 rows and 12 columns of random integers range [0,100]
X = np.random.randint(0,100,(3,12))
print("Matrix X:")
print(X)
```

```
Matrix X:
[[44 15 15 58 68 26 54 34  6 52 27 78]
 [44 81 99 25 39 70 44 99 95 90 88 37]
 [98 16 69 10 61 30 13 93 24  9 59 62]]
```

In [15]:

```
##(b) create matrix XA of 3 rows and 3 columns of subset matrix of X with all ro
XA = X[0:, :3]
print("Matrix XA:")
print(XA)
```

```
Matrix XA:
[[44 15 15]
 [44 81 99]
 [98 16 69]]
```

In [16]:

```
##(c) create matrix XB of 3 rows and 3 columns of subset matrix of X with all ro
XB = X[:3,3:6]
print("Matrix XB:")
print(XB)
```

```
Matrix XB:
[[58 68 26]
 [25 39 70]
 [10 61 30]]
```

In [17]:

```
##(d)create matrix XC of 3 rows and 3 columns of subset matrix of X with all row
XC = X[:3,6:9]
print("Matrix XC:")
print(XC)
```

```
Matrix XC:
[[54 34  6]
 [44 99 95]
 [13 93 24]]
```

In [18]:

```
##(e)create matrix XD of 3 rows and 3 columns of subset matrix of X with all row
XD = X[:3,9:12]
print("Matrix XD:")
print(XD)
```

```
Matrix XD:
[[52 27 78]
 [90 88 37]
 [ 9 59 62]]
```

In [19]:

```
##(f)create matrix Y of 3 rows and 12 columns by putting together in the order o
Y = np.concatenate((XD, XA, XB, XC),  axis=1 )
print("Matrix Y:")
print(Y)
```

```
Matrix Y:
[[52 27 78 44 15 15 58 68 26 54 34  6]
 [90 88 37 44 81 99 25 39 70 44 99 95]
 [ 9 59 62 98 16 69 10 61 30 13 93 24]]
```

In [20]:

```
##(g)create matrix Z of 12 rows and 3 columns by putting together vertically in
Z = np.concatenate((XD, XA, XB, XC),  axis=0 )
print("Matrix Z:")
print(Z)
```

```
Matrix Z:
[[52 27 78]
 [90 88 37]
 [ 9 59 62]
 [44 15 15]
 [44 81 99]
 [98 16 69]
 [58 68 26]
 [25 39 70]
 [10 61 30]
 [54 34  6]
 [44 99 95]
 [13 93 24]]
```

In [21]:

```
###6. Create the following two matrixes in the program
##and do the matrix multiplication where C = AB
import numpy as np
A = np.arange(1,9).reshape((4,2))
print("Matrix A:")
print(A)

B = np.arange(9,17).reshape((2,4))
print("Matrix B:")
print(B)

C = np.dot(A,B)
print("Matrix C = AB:")
print(C)
```

```
Matrix A:
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
Matrix B:
[[ 9 10 11 12]
 [13 14 15 16]]
Matrix C = AB:
[[ 35  38  41  44]
 [ 79  86  93 100]
 [123 134 145 156]
 [167 182 197 212]]
```

In [ ]:

In [ ]: