I.) Initialize matrix, transpose, determinant, size, inverse, power, add, minus,multiply, rank, nulity

In [1]:
```
import numpy as np
```

In [2]:
```
a1= np.matrix([['a','b','c'],['d','e','f'],['g','h','i']])
print('Initialize matrix :')
print(a1)
```

```
Initialize matrix :
[['a' 'b' 'c']
 ['d' 'e' 'f']
 ['g' 'h' 'i']]
```

In [3]:
```
print('After transpose on matrix : ')
print(a1.transpose())
```

```
After transpose on matrix :
[['a' 'd' 'g']
 ['b' 'e' 'h']
 ['c' 'f' 'i']]
```

In [4]:
```
print('#dimension of a1 : ',a1.ndim)
print('#shape of a1 : ',a1.shape)
print('#datatype of a1 : ',a1.dtype)
```

```
#dimension of a1 :  2
#shape of a1 :  (3, 3)
#datatype of a1 :  <U1
```

In [5]:
```
print('size of element : ',a1.itemsize,'bits')
print('total size of a1 : ',a1.nbytes,'bits')
```

```
size of element :  4 bits
total size of a1 :  36 bits
```

In [6]:
```
a2 = np.matrix([['d','a','e'],['a','t','w'],['s','e','y']])
print(a2)
```

```
[['d' 'a' 'e']
 ['a' 't' 'w']
 ['s' 'e' 'y']]
```

In [7]:
```
print('after transpose \n',a2.transpose())
```

```
after transpose
 [['d' 'a' 's']
 ['a' 't' 'e']
 ['e' 'w' 'y']]
```

In [8]:
```
print('after transpose\n',a2.T)
```

```
after transpose
 [['d' 'a' 's']
 ['a' 't' 'e']
 ['e' 'w' 'y']]
```

In [9]:
```
print('#dimension of a2 : ',a2.ndim)
print('#shape of a2 : ',a2.shape)
print('#size of element : ',a2.itemsize)
print('#total size of a2 : ',a2.nbytes)
```

```
#dimension of a2 :  2
#shape of a2 :  (3, 3)
#size of element :  4
#total size of a2 :  36
```

In [10]:
```
a = np.array([[1,2],[4,6]])
b = a.transpose()
print('after tranpose\n',b)
print("determinant(a)",np.linalg.det(a))
```

```
after tranpose
 [[1 4]
 [2 6]]
determinant(a) -2.0
```

In [11]:
```
c = np.array([[1,1,3],[-2,2,1],[4,7,4]])
print('transpose of c\n',c.T)
print('det(c) : ',np.linalg.det(c))
```

```
transpose of c
 [[ 1 -2  4]
 [ 1  2  7]
 [ 3  1  4]]
det(c) :  -53.00000000000001
```

Inverse of matrix

In [12]:
```
M = np.array([[1,1,1],[0,2,5],[2,5,-1]])
print(a)
```

```
[[1 2]
 [4 6]]
```

In [13]:
```
# Display the inverse of matrix M
print(np.linalg.inv(M))
```

```
[[ 1.28571429 -0.28571429 -0.14285714]
 [-0.47619048  0.14285714  0.23809524]
 [ 0.19047619  0.14285714 -0.0952381 ]]
```

In [14]:
```
A = np.array([[6, 1, 1, 3],
              [4, -2, 5, 1],
              [2, 8, 7, 6],
              [3, 1, 9, 7]])
print(np.linalg.inv(A))
```

```
[[ 0.13368984  0.10695187  0.02139037 -0.09090909]
 [-0.00229183  0.02673797  0.14820474 -0.12987013]
 [-0.12987013  0.18181818  0.06493506 -0.02597403]
 [ 0.11000764 -0.28342246 -0.11382735  0.23376623]]
```

In [15]:
```
N = np.array([[6, 1, 1],
              [4, -2, 5],
              [2, 8, 7]])
print(np.linalg.inv(N))
```

```
[[ 0.17647059 -0.00326797 -0.02287582]
 [ 0.05882353 -0.13071895  0.08496732]
 [-0.11764706  0.1503268   0.05228758]]
```

In [16]:
```
O = np.array([[[1,2],[3,4]],
              [[1,3],[3,5]]])
print(np.linalg.inv(O))
```

```
[[[-2.    1.  ]
  [ 1.5  -0.5 ]]

 [[-1.25  0.75]
  [ 0.75 -0.25]]]
```

Rank and Nulity of Matrix

In [17]:
```
print(a2)
```

```
[['d' 'a' 'e']
 ['a' 't' 'w']
 ['s' 'e' 'y']]
```

In [18]:
```
# Display the rank of matrix a
print(np.linalg.matrix_rank(a1))
```

```
---------------------------------------------------------------------------
UFuncTypeError                            Traceback (most recent call last)
Input In [18], in <cell line: 2>()
      1 # Display the rank of matrix a
----> 2 print(np.linalg.matrix_rank(a1))

File <__array_function__ internals>:5, in matrix_rank(*args, **kwargs)

File C:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py:1901, in matrix_rank(M, tol, hermitian)
   1899 if M.ndim < 2:
   1900     return int(not all(M==0))
-> 1901 S = svd(M, compute_uv=False, hermitian=hermitian)
   1902 if tol is None:
   1903     tol = S.max(axis=-1, keepdims=True) * max(M.shape[-2:]) * finfo(S.dtype).eps

File <__array_function__ internals>:5, in svd(*args, **kwargs)

File C:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py:1672, in svd(a, full_matrices, compute_uv, hermitian)
   1669     gufunc = _umath_linalg.svd_n
   1671 signature = 'D->d' if isComplexType(t) else 'd->d'
-> 1672 s = gufunc(a, signature=signature, extobj=extobj)
   1673 s = s.astype(_realType(result_t), copy=False)
   1674 return s

UFuncTypeError: Cannot cast ufunc 'svd_n' input from dtype('<U1') to dtype('float64') with casting rule 'same_kind'
```

In [ ]:
```
# Display the nulity of matrix a
nulity = a1.shape[1] - np.linalg.matrix_rank(a1)
print(nulity)
```

In [ ]:
```
print(A)
print(np.linalg.matrix_rank(A)) # is the rank of matrix A
```

In [ ]:
```
nulity2 = A.shape[1]-np.linalg.matrix_rank(A)
print(nulity2)
```

Eigenvalue of matrix

In [ ]:
```
print(f)
```

In [ ]:
```
X = np.matrix([[2,1,1],[1,2,1],[1,1,2]])
print(X)
u,v=np.linalg.eig(X)
print("Eigenvalue :",u)
print("Eigenvector : ",v)
```

In [19]:
```
print(c)
w,v = np.linalg.eig(c)
print("Eigenvalue : \n",w)
print("Eigenvector : \n",v)
```

```
[[ 1  1  3]
 [-2  2  1]
 [ 4  7  4]]
Eigenvalue :
 [-2.52202863  3.47554612  6.04648251]
Eigenvector :
 [[ 0.52636191  0.63158487  0.50527621]
 [ 0.39885396 -0.41019718 -0.03666742]
 [-0.75090523  0.65790488  0.86217832]]
```

In [20]:

```
m = np.array([[1, 2, 3],
              [2, 3, 4],
              [4, 5, 6]])
print("Printing the Original square array:\n",m)

# find eigenvalues and eigenvectors
w, v = np.linalg.eig(m)
print("Printing the Eigen values of the given square array:\n",w)
print("Printing Right eigenvectors of the given square array:\n",v)
```

```
Printing the Original square array:
 [[1 2 3]
 [2 3 4]
 [4 5 6]]
Printing the Eigen values of the given square array:
 [ 1.08309519e+01 -8.30951895e-01 -1.09042184e-16]
Printing Right eigenvectors of the given square array:
 [[ 0.34416959  0.72770285  0.40824829]
 [ 0.49532111  0.27580256 -0.81649658]
 [ 0.79762415 -0.62799801  0.40824829]]
```

Elementary row operation(row echelon form)

In [21]:

```
def fixRowTwo(A) :

    # Sets the sub-diagonal elements of row two to zero
    A[2] = A[2] - A[2,0] * A[1]
    A[2] = A[2] - A[2,1] * A[1]

    # Test if diagonal element is not zero.
    if A[2,2] == 0 :
        # Add a lower row to row two.
        A[2] = A[2] + A[3]

        # Sets the sub-diagonal elements to zero again ???
        A[2] = A[2] - A[2,0] * A[1]
        A[2] = A[2] - A[2,1] * A[1]

    if A[2,2] == 0 :
        print("S I N G U L A R")
        sys.Exit()

    # Set the diagonal element to one
    A[2] = A[2] / A[2,2]

    return A

def fixRowThree(A) :

    # Sets the sub-diagonal elements of row two to zero
    A[3] = A[3] - A[3,0] * A[2]
    A[3] = A[3] - A[3,1] * A[2]
    A[3] = A[3] - A[3,2] * A[2]

    # Test if diagonal element is not zero.
    if A[3,3] == 0:
        print("S I N G U L A R")
        sys.Exit()

    # Set the diagonal element to one
    A[3] = A[3] / A[3,3]

    return A

A = np.array([
        [1, 7, 4, 3],
        [0, 1, 2, 3],
        [3, 2, 0, 3],
        [1, 3, 1, 3]
    ], dtype=np.float_)

fixRowTwo(A)
print("")
print("Row Two:")
print(A)

fixRowThree(A)
print("")
print("Row Three:")
print(A)
```

```
Row Two:
[[ 1.    7.    4.    3.  ]
 [ 0.    1.    2.    3.  ]
 [-0.75 -0.    1.    0.75]
 [ 1.    3.    1.    3.  ]]

Row Three:
[[ 1.          7.          4.          3.        ]
 [ 0.          1.          2.          3.        ]
 [-0.75       -0.          1.          0.75      ]
 [ 0.77777778  1.33333333  0.          1.        ]]
```

In [22]:
```
def row_echelon(A):
    """ Return Row Echelon Form of matrix A """

    # if matrix A has no columns or rows,
    # it is already in REF, so we return itself
    r, c = A.shape
    if r == 0 or c == 0:
        return A

    # we search for non-zero element in the first column
    for i in range(len(A)):
        if A[i,0] != 0:
            break
    else:
        # if all elements in the first column is zero,
        # we perform REF on matrix from second column
        B = row_echelon(A[:,1:])
        # and then add the first zero-column back
        return np.hstack([A[:,:1], B])

    # if non-zero element happens not in the first row,
    # we switch rows
    if i > 0:
        ith_row = A[i].copy()
        A[i] = A[0]
        A[0] = ith_row

    # we divide first row by first element in it
    A[0] = A[0] / A[0,0]
    # we subtract all subsequent rows with first row (it has 1 now as first element)
    # multiplied by the corresponding element in the first column
    A[1:] -= A[0] * A[1:,0:1]

    # we perform REF on matrix from second row, from second column
    B = row_echelon(A[1:,1:])

    # we add first row and first (zero) column, and return
    return np.vstack([A[:1], np.hstack([A[1:,:1], B]) ])

A = np.array([[4, 7, 3, 8],
              [8, 3, 8, 7],
              [2, 9, 5, 3]], dtype='float')

print(row_echelon(A))
```

```
[[ 1.          1.75        0.75        2.        ]
 [ 0.          1.         -0.18181818  0.81818182]
 [ 0.          0.          1.         -1.22222222]]
```

In [ ]:
```

```

Summation, Multiplication, Substraction, Power

In [23]:
```
d = np.array([[5,4,6],[8,9,0],[3,4,5]])
print(c,'\n+\n',d,'\n=')
print(c+d)
```

```
[[ 1  1  3]
 [-2  2  1]
 [ 4  7  4]]
+
 [[5 4 6]
 [8 9 0]
 [3 4 5]]
=
[[ 6  5  9]
 [ 6 11  1]
 [ 7 11  9]]
```

In [24]:
```
print(c-d)
```

```
[[ -4  -3  -3]
 [-10  -7   1]
 [  1   3  -1]]
```

In [25]:
```
print('c^T-d^T = ')
print(c.T-d.T)
```

```
c^T-d^T =
[[ -4 -10   1]
 [ -3  -7   3]
 [ -3   1  -1]]
```

In [26]:
```
print('det(c-d) = ')
print(np.linalg.det(c-d))
```

```
det(c-d) =
79.99999999999997
```

In [27]:
```
print('det(c^T-d^T) = ')
print(np.linalg.det(c.T-d.T))
```

```
det(c^T-d^T) =
80.00000000000004
```

In [28]:
```
print('(c-d)^T = ')
print((c-d).T)
```

```
(c-d)^T =
[[ -4 -10   1]
 [ -3  -7   3]
 [ -3   1  -1]]
```

In [29]:
```
print('c*d = \n',c*d)
```

```
c*d =
 [[  5   4  18]
 [-16  18   0]
 [ 12  28  20]]
```

In [30]:
```
print(c.T*d.T)
print(np.linalg.det(c.T*d.T))
```

```
[[  5 -16  12]
 [  4  18  28]
 [ 18   0  20]]
-8871.999999999998
```

In [31]:
```
print('c^2 = \n',c**2)
```

```
c^2 =
 [[ 1  1  9]
 [ 4  4  1]
 [16 49 16]]
```

In [32]:
```
print('c^3 = \n',c**3)
```

```
c^3 =
 [[  1   1  27]
 [ -8   8   1]
 [ 64 343  64]]
```

In [33]:
```
print('Initialize matrix c \n',c)
for i in range(5):
    print("=> c^",i,"\n",c^i,"\n")
```

```
Initialize matrix c
 [[ 1  1  3]
 [-2  2  1]
 [ 4  7  4]]
=> c^ 0
 [[ 1  1  3]
 [-2  2  1]
 [ 4  7  4]]

=> c^ 1
 [[ 0  0  2]
 [-1  3  0]
 [ 5  6  5]]

=> c^ 2
 [[ 3  3  1]
 [-4  0  3]
 [ 6  5  6]]

=> c^ 3
 [[ 2  2  0]
 [-3  1  2]
 [ 7  4  7]]

=> c^ 4
 [[ 5  5  7]
 [-6  6  5]
 [ 0  3  0]]
```

In [34]:
```
print('Initialize matrix c \n',c)
for i in range(11):
    print("=> pow(c,"+str(i)+")\n",c**i,"\n")
```

```
Initialize matrix c
 [[ 1  1  3]
 [-2  2  1]
 [ 4  7  4]]
=> pow(c,0)
 [[1 1 1]
 [1 1 1]
 [1 1 1]]

=> pow(c,1)
 [[ 1  1  3]
 [-2  2  1]
 [ 4  7  4]]

=> pow(c,2)
 [[ 1  1  9]
 [ 4  4  1]
 [16 49 16]]

=> pow(c,3)
 [[  1   1  27]
 [ -8   8   1]
 [ 64 343  64]]

=> pow(c,4)
 [[   1    1   81]
 [  16   16    1]
 [ 256 2401  256]]

=> pow(c,5)
 [[    1     1   243]
 [  -32    32     1]
 [ 1024 16807  1024]]

=> pow(c,6)
 [[     1      1    729]
 [    64     64      1]
 [  4096 117649   4096]]

=> pow(c,7)
 [[     1      1   2187]
 [  -128    128      1]
 [ 16384 823543  16384]]

=> pow(c,8)
 [[      1       1    6561]
 [    256     256       1]
 [  65536 5764801   65536]]

=> pow(c,9)
 [[       1        1    19683]
 [    -512      512        1]
 [  262144 40353607   262144]]

=> pow(c,10)
 [[        1         1     59049]
 [     1024      1024         1]
 [  1048576 282475249   1048576]]
```

In [35]:

```
print('initialize matrix d')
print(d)
for i in range(31):
    print('pow(d,'+str(i)+')\n',d**i)
```

```
initialize matrix d
[[5 4 6]
 [8 9 0]
 [3 4 5]]
pow(d,0)
 [[1 1 1]
 [1 1 1]
 [1 1 1]]
pow(d,1)
 [[5 4 6]
 [8 9 0]
 [3 4 5]]
pow(d,2)
 [[25 16 36]
 [64 81  0]
 [ 9 16 25]]
pow(d,3)
 [[125  64 216]
 [512 729   0]
 [ 27  64 125]]
pow(d,4)
 [[ 625  256 1296]
 [4096 6561    0]
 [  81  256  625]]
pow(d,5)
 [[ 3125  1024  7776]
 [32768 59049     0]
 [  243  1024  3125]]
pow(d,6)
 [[ 15625   4096  46656]
 [262144 531441      0]
 [   729   4096  15625]]
pow(d,7)
 [[  78125   16384  279936]
 [2097152 4782969       0]
 [   2187   16384   78125]]
pow(d,8)
 [[  390625    65536  1679616]
 [16777216 43046721        0]
 [    6561    65536   390625]]
pow(d,9)
 [[  1953125    262144  10077696]
 [134217728 387420489         0]
 [    19683    262144   1953125]]
pow(d,10)
 [[   9765625    1048576   60466176]
 [1073741824 -808182895          0]
 [     59049    1048576    9765625]]
pow(d,11)
 [[  48828125    4194304  362797056]
 [         0 1316288537          0]
 [    177147    4194304   48828125]]
pow(d,12)
 [[  244140625    16777216 -2118184960]
 [          0 -1038305055           0]
 [     531441    16777216   244140625]]
pow(d,13)
 [[1220703125   67108864  175792128]
 [         0 -754810903          0]
 [   1594323   67108864 1220703125]]
pow(d,14)
 [[1808548329  268435456 1054752768]
 [         0 1796636465          0]
 [   4782969  268435456 1808548329]]
pow(d,15)
 [[  452807053  1073741824  2033549312]
 [          0 -1010140999           0]
 [   14348907  1073741824   452807053]]
pow(d,16)
 [[-2030932031           0  -683606016]
 [          0  -501334399           0]
 [   43046721           0 -2030932031]]
pow(d,17)
 [[-1564725563           0   193331200]
 [          0  -217042295           0]
 [  129140163           0 -1564725563]]
pow(d,18)
 [[  766306777           0  1159987200]
 [          0 -1953380655           0]
 [  387420489           0   766306777]]
pow(d,19)
 [[ -463433411           0 -1630011392]
 [          0  -400556711           0]
 [ 1162261467           0  -463433411]]
pow(d,20)
 [[ 1977800241           0 -1190133760]
```

```
 [          0   689956897           0]
 [ -808182895           0  1977800241]]
pow(d,21)
 [[1299066613          0 1449132032]
 [         0 1914644777          0]
 [1870418611          0 1299066613]]
pow(d,22)
 [[-2094601527           0   104857600]
 [          0    51933809           0]
 [ 1316288537           0 -2094601527]]
pow(d,23)
 [[-1883073043           0   629145600]
 [          0   467404281           0]
 [ -346101685           0 -1883073043]]
pow(d,24)
 [[ -825430623           0  -520093696]
 [          0   -88328767           0]
 [-1038305055           0  -825430623]]
pow(d,25)
 [[ 167814181          0 1174405120]
 [         0 -794958903          0]
 [1180052131          0  167814181]]
pow(d,26)
 [[  839070905           0 -1543503872]
 [          0  1435304465           0]
 [ -754810903           0   839070905]]
pow(d,27)
 [[ -99612771          0 -671088640]
 [         0   32838297          0]
 [2030534587          0  -99612771]]
pow(d,28)
 [[-498063855          0  268435456]
 [         0  295544673          0]
 [1796636465          0 -498063855]]
pow(d,29)
 [[ 1804648021           0  1610612736]
 [          0 -1635065239           0]
 [ 1094942099           0  1804648021]]
pow(d,30)
 [[  433305513           0  1073741824]
 [          0 -1830685263           0]
 [-1010140999           0   433305513]]
```

In [36]:

```
print('initialize matrix d')
print(d)
for i in range(31):
    print('\n==> d^'+str(i)+'\n',d^i)
```

```
initialize matrix d
[[5 4 6]
 [8 9 0]
 [3 4 5]]

==> d^0
 [[5 4 6]
 [8 9 0]
 [3 4 5]]

==> d^1
 [[4 5 7]
 [9 8 1]
 [2 5 4]]

==> d^2
 [[ 7  6  4]
 [10 11  2]
 [ 1  6  7]]

==> d^3
 [[ 6  7  5]
 [11 10  3]
 [ 0  7  6]]

==> d^4
 [[ 1  0  2]
 [12 13  4]
 [ 7  0  1]]

==> d^5
 [[ 0  1  3]
 [13 12  5]
 [ 6  1  0]]

==> d^6
 [[ 3  2  0]
 [14 15  6]
 [ 5  2  3]]

==> d^7
 [[ 2  3  1]
 [15 14  7]
 [ 4  3  2]]

==> d^8
 [[13 12 14]
 [ 0  1  8]
 [11 12 13]]

==> d^9
 [[12 13 15]
 [ 1  0  9]
 [10 13 12]]

==> d^10
 [[15 14 12]
 [ 2  3 10]
 [ 9 14 15]]

==> d^11
 [[14 15 13]
 [ 3  2 11]
 [ 8 15 14]]

==> d^12
 [[ 9  8 10]
 [ 4  5 12]
 [15  8  9]]

==> d^13
 [[ 8  9 11]
 [ 5  4 13]
 [14  9  8]]

==> d^14
 [[11 10  8]
 [ 6  7 14]
 [13 10 11]]

==> d^15
 [[10 11  9]
 [ 7  6 15]
 [12 11 10]]

==> d^16
```

```
 [[21 20 22]
 [24 25 16]
 [19 20 21]]

==> d^17
 [[20 21 23]
 [25 24 17]
 [18 21 20]]

==> d^18
 [[23 22 20]
 [26 27 18]
 [17 22 23]]

==> d^19
 [[22 23 21]
 [27 26 19]
 [16 23 22]]

==> d^20
 [[17 16 18]
 [28 29 20]
 [23 16 17]]

==> d^21
 [[16 17 19]
 [29 28 21]
 [22 17 16]]

==> d^22
 [[19 18 16]
 [30 31 22]
 [21 18 19]]

==> d^23
 [[18 19 17]
 [31 30 23]
 [20 19 18]]

==> d^24
 [[29 28 30]
 [16 17 24]
 [27 28 29]]

==> d^25
 [[28 29 31]
 [17 16 25]
 [26 29 28]]

==> d^26
 [[31 30 28]
 [18 19 26]
 [25 30 31]]

==> d^27
 [[30 31 29]
 [19 18 27]
 [24 31 30]]

==> d^28
 [[25 24 26]
 [20 21 28]
 [31 24 25]]

==> d^29
 [[24 25 27]
 [21 20 29]
 [30 25 24]]

==> d^30
 [[27 26 24]
 [22 23 30]
 [29 26 27]]
```

In [37]:
```
e = np.matrix([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[0,1,2,3,3],[4,6,7,8,9]])
f = np.matrix([[4,5,6,7,9],[3,4,65,3,2],[5,7,67,54,3],[0,0,98,8,9],[5,-3,-4,-5,2]])
print('transpose(e)\n',e.T)
print('transpose(f)\n',f.T)
print('sum(e+f) = \n'+str(e+f))
print('sum(e.T+f.T) = \n',e.T+f.T)
```

```
transpose(e)
 [[ 1  6 11  0  4]
 [ 2  7 12  1  6]
 [ 3  8 13  2  7]
 [ 4  9 14  3  8]
 [ 5 10 15  3  9]]
transpose(f)
 [[ 4  3  5  0  5]
 [ 5  4  7  0 -3]
 [ 6 65 67 98 -4]
 [ 7  3 54  8 -5]
 [ 9  2  3  9  2]]
sum(e+f) =
[[  5   7   9  11  14]
 [  9  11  73  12  12]
 [ 16  19  80  68  18]
 [  0   1 100  11  12]
 [  9   3   3   3  11]]
sum(e.T+f.T) =
 [[  5   9  16   0   9]
 [  7  11  19   1   3]
 [  9  73  80 100   3]
 [ 11  12  68  11   3]
 [ 14  12  18  12  11]]
```

In [38]:
```
print('det(e)   = ',np.linalg.det(e))
print('det(e.T) = ',np.linalg.det(e.T))
```

```
det(e)   =  2.5288413338684173e-15
det(e.T) =  3.330669073875456e-15
```

In [39]:
```
print('det(f) = ',np.linalg.det(f))
print('det(f.T) = ',np.linalg.det(f.T))
```

```
det(f) =  1895118.000000004
det(f.T) =  1895118.0000000005
```

In [40]:
```
print(e.ndim)
print(f.ndim)
```

```
2
2
```

In [41]:
```
g = np.array([[9.0,8.0,7.0],[6.0,5.0,4.0]])
h = np.array([[1,2,3,4,5,0],[-3,8,-5,0,8,8],[1,2,3,4,5,6]])
print('The 2x3 order of g is \n',g)
print('The 3x6 order of h is \n',h)
```

```
The 2x3 order of g is
 [[9. 8. 7.]
 [6. 5. 4.]]
The 3x6 order of h is
 [[ 1  2  3  4  5  0]
 [-3  8 -5  0  8  8]
 [ 1  2  3  4  5  6]]
```

In [51]:
```
a11 = int(input("Enter a11 : "))
a12 = int(input("Enter a12 : "))
a13 = int(input("Enter a13 : "))
a14 = int(input("Enter a14 : "))

a21 = int(input("Enter a21 : "))
a22 = int(input("Enter a22 : "))
a23 = int(input("Enter a23 : "))
a24 = int(input("Enter a24 : "))

a31 = int(input("Enter a31 : "))
a32 = int(input("Enter a32 : "))
a33 = int(input("Enter a33 : "))
a34 = int(input("Enter a34 : "))

a41 = int(input("Enter a41 : "))
a42 = int(input("Enter a42 : "))
a43 = int(input("Enter a43 : "))
a44 = int(input("Enter a44 : "))
i = np.array([[a11,a12,a13,a14],
              [a21,a22,a23,a24],
              [a31,a32,a33,a34],
              [a41,a42,a43,a44]])
print(i)
```

```
Enter a11 : 1
Enter a12 : 2
Enter a13 : 3
Enter a14 : 4
Enter a21 : 5
Enter a22 : 6
Enter a23 : 7
Enter a24 : 8
Enter a31 : 9
Enter a32 : 0
Enter a33 : 1
Enter a34 : 2
Enter a41 : 3
Enter a42 : 4
Enter a43 : 5
Enter a44 : 6
[[1 2 3 4]
 [5 6 7 8]
 [9 0 1 2]
 [3 4 5 6]]
```

In [52]:
```
print(i)
print(i.T)
print(np.linalg.det(i.T))
print(np.linalg.det(i))
```

```
[[1 2 3 4]
 [5 6 7 8]
 [9 0 1 2]
 [3 4 5 6]]
[[1 5 9 3]
 [2 6 0 4]
 [3 7 1 5]
 [4 8 2 6]]
0.0
4.17772812229319e-14
```

In [53]:
```
j = np.matrix([[1,2,3.9,4,5],[6,7,8,9,0],[-1,-3,4,5,6],[2,3,4,5,6],[0,8,6,5,4]])
print(j)
```

```
[[ 1.   2.   3.9  4.   5. ]
 [ 6.   7.   8.   9.   0. ]
 [-1.  -3.   4.   5.   6. ]
 [ 2.   3.   4.   5.   6. ]
 [ 0.   8.   6.   5.   4. ]]
```

In [54]:
```
print("After transpose(j)\n",j.T,"\n")
print(np.linalg.det(j),"\n")
print(j**2,"\n")
print(j**100)
```

```
After transpose(j)
 [[ 1.   6.  -1.   2.   0. ]
 [ 2.   7.  -3.   3.   8. ]
 [ 3.9  8.   4.   4.   6. ]
 [ 4.   9.   5.   5.   5. ]
 [ 5.   0.   6.   6.   4. ]]

888.5999999999988

[[ 17.1  56.3  81.5  86.5  72.4]
 [ 58.   64.  147.4 172.  132. ]
 [-13.   28.   44.1  44.   73. ]
 [ 26.   76.  103.8 110.   88. ]
 [ 52.   85.  132.  147.   82. ]]

[[1.46372009e+128 3.45731468e+128 5.56042406e+128 6.08504870e+128
  4.92050705e+128]
 [2.57952687e+128 6.09285628e+128 9.79918457e+128 1.07237352e+129
  8.67145315e+128]
 [8.74316042e+127 2.06513917e+128 3.32137818e+128 3.63474939e+128
  2.93913999e+128]
 [1.88349039e+128 4.44881438e+128 7.15506016e+128 7.83013832e+128
  6.33162571e+128]
 [2.25014271e+128 5.31484913e+128 8.54791007e+128 9.35440328e+128
  7.56418059e+128]]
```

In [55]:
```
print(j.trace())
print(j.conjugate())
print("sort j",j.argsort())
```

```
[[21.]]
[[ 1.   2.   3.9  4.   5. ]
 [ 6.   7.   8.   9.   0. ]
 [-1.  -3.   4.   5.   6. ]
 [ 2.   3.   4.   5.   6. ]
 [ 0.   8.   6.   5.   4. ]]
sort j [[0 1 2 3 4]
 [4 0 1 2 3]
 [1 0 2 3 4]
 [0 1 2 3 4]
 [0 4 3 2 1]]
```

In [56]:
```
k = np.array([[1,2,3,4,5,6,7,8,9,10],[4,1,5,5,6,0,-4,-2,-1,6],[1,8,0,7,8,9,-2,-1,3,-4],[2,4,5,6,4,3,8,9,0,7],
              [3,8,9,0,4,3,5,6,7,8],[1,2,3,4,5,6,7,8,9,0],[4,6,3,2,1,-9,-7,0,6,4],[1,4,5,6,7,5,4,3,2,1],
              [1,2,3,4,5,6,7,8,9,0],[0,0,7,0,5,4,3,9,0,1]])
print(k)
```

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

In [57]:
```
print('After transpose')
print(k.T)
```

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

In [58]:
```
print('det(k) = ',np.linalg.det(k))
print('det(k^T) = ',np.linalg.det(k.T))
```

```
det(k) =  0.0
det(k^T) =  1.3485847077899445e-08
```

In [59]:

```
for i in range(1,11):
    print('\npow(k,'+str(i)+") = \n",pow(k,i))
```

```
pow(k,1) =
 [[ 1  2  3  4  5  6  7  8  9 10]
 [ 4  1  5  5  6  0 -4 -2 -1  6]
 [ 1  8  0  7  8  9 -2 -1  3 -4]
 [ 2  4  5  6  4  3  8  9  0  7]
 [ 3  8  9  0  4  3  5  6  7  8]
 [ 1  2  3  4  5  6  7  8  9  0]
 [ 4  6  3  2  1 -9 -7  0  6  4]
 [ 1  4  5  6  7  5  4  3  2  1]
 [ 1  2  3  4  5  6  7  8  9  0]
 [ 0  0  7  0  5  4  3  9  0  1]]

pow(k,2) =
 [[  1   4   9  16  25  36  49  64  81 100]
 [ 16   1  25  25  36   0  16   4   1  36]
 [  1  64   0  49  64  81   4   1   9  16]
 [  4  16  25  36  16   9  64  81   0  49]
 [  9  64  81   0  16   9  25  36  49  64]
 [  1   4   9  16  25  36  49  64  81   0]
 [ 16  36   9   4   1  81  49   0  36  16]
 [  1  16  25  36  49  25  16   9   4   1]
 [  1   4   9  16  25  36  49  64  81   0]
 [  0   0  49   0  25  16   9  81   0   1]]

pow(k,3) =
 [[   1    8   27   64  125  216  343  512  729 1000]
 [  64    1  125  125  216    0  -64   -8   -1  216]
 [   1  512    0  343  512  729   -8   -1   27  -64]
 [   8   64  125  216   64   27  512  729    0  343]
 [  27  512  729    0   64   27  125  216  343  512]
 [   1    8   27   64  125  216  343  512  729    0]
 [  64  216   27    8    1 -729 -343    0  216   64]
 [   1   64  125  216  343  125   64   27    8    1]
 [   1    8   27   64  125  216  343  512  729    0]
 [   0    0  343    0  125   64   27  729    0    1]]

pow(k,4) =
 [[    1    16    81   256   625  1296  2401  4096  6561 10000]
 [  256     1   625   625  1296     0   256    16     1  1296]
 [    1  4096     0  2401  4096  6561    16     1    81   256]
 [   16   256   625  1296   256    81  4096  6561     0  2401]
 [   81  4096  6561     0   256    81   625  1296  2401  4096]
 [    1    16    81   256   625  1296  2401  4096  6561     0]
 [  256  1296    81    16     1  6561  2401     0  1296   256]
 [    1   256   625  1296  2401   625   256    81    16     1]
 [    1    16    81   256   625  1296  2401  4096  6561     0]
 [    0     0  2401     0   625   256    81  6561     0     1]]

pow(k,5) =
 [[     1     32    243   1024   3125   7776  16807  32768  59049 100000]
 [  1024      1   3125   3125   7776      0  -1024    -32     -1   7776]
 [     1  32768      0  16807  32768  59049    -32     -1    243  -1024]
 [    32   1024   3125   7776   1024    243  32768  59049      0  16807]
 [   243  32768  59049      0   1024    243   3125   7776  16807  32768]
 [     1     32    243   1024   3125   7776  16807  32768  59049      0]
 [  1024   7776    243     32      1 -59049 -16807      0   7776   1024]
 [     1   1024   3125   7776  16807   3125   1024    243     32      1]
 [     1     32    243   1024   3125   7776  16807  32768  59049      0]
 [     0      0  16807      0   3125   1024    243  59049      0      1]]

pow(k,6) =
 [[      1      64     729    4096   15625   46656  117649  262144  531441
  1000000]
 [   4096       1   15625   15625   46656       0    4096      64       1
    46656]
 [      1  262144       0  117649  262144  531441      64       1     729
     4096]
 [     64    4096   15625   46656    4096     729  262144  531441       0
   117649]
 [    729  262144  531441       0    4096     729   15625   46656  117649
   262144]
 [      1      64     729    4096   15625   46656  117649  262144  531441
        0]
 [   4096   46656     729      64       1  531441  117649       0   46656
     4096]
 [      1    4096   15625   46656  117649   15625    4096     729      64
        1]
 [      1      64     729    4096   15625   46656  117649  262144  531441
        0]
 [      0       0  117649       0   15625    4096     729  531441       0
        1]]

pow(k,7) =
 [[       1      128     2187    16384    78125   279936   823543  2097152
   4782969 10000000]
```

```
 [   16384       1    78125    78125   279936        0   -16384     -128
       -1   279936]
 [       1  2097152        0   823543  2097152  4782969     -128       -1
     2187   -16384]
 [     128    16384    78125   279936    16384     2187  2097152  4782969
        0   823543]
 [    2187  2097152  4782969        0    16384     2187    78125   279936
    823543  2097152]
 [       1      128     2187    16384    78125   279936   823543  2097152
   4782969        0]
 [   16384   279936     2187      128        1 -4782969  -823543        0
    279936    16384]
 [       1    16384    78125   279936   823543    78125    16384     2187
      128        1]
 [       1      128     2187    16384    78125   279936   823543  2097152
   4782969        0]
 [       0        0   823543        0    78125    16384     2187  4782969
         0        1]]

pow(k,8) =
 [[        1       256      6561     65536    390625   1679616   5764801
   16777216  43046721 100000000]
 [    65536         1    390625    390625   1679616         0     65536
       256         1   1679616]
 [        1  16777216         0   5764801  16777216  43046721       256
         1      6561     65536]
 [      256     65536    390625   1679616     65536      6561  16777216
   43046721         0   5764801]
 [     6561  16777216  43046721         0     65536      6561    390625
    1679616   5764801  16777216]
 [        1       256      6561     65536    390625   1679616   5764801
   16777216  43046721         0]
 [    65536   1679616      6561       256         1  43046721   5764801
         0   1679616     65536]
 [        1     65536    390625   1679616   5764801    390625     65536
      6561       256         1]
 [        1       256      6561     65536    390625   1679616   5764801
   16777216  43046721         0]
 [        0         0   5764801         0    390625     65536      6561
   43046721         0         1]]

pow(k,9) =
 [[         1        512      19683     262144    1953125   10077696
    40353607  134217728  387420489 1000000000]
 [    262144          1    1953125    1953125   10077696          0
     -262144       -512         -1   10077696]
 [         1  134217728          0   40353607  134217728  387420489
       -512         -1      19683    -262144]
 [       512     262144    1953125   10077696     262144      19683
   134217728  387420489          0   40353607]
 [     19683  134217728  387420489          0     262144      19683
     1953125   10077696   40353607  134217728]
 [         1        512      19683     262144    1953125   10077696
    40353607  134217728  387420489          0]
 [    262144   10077696      19683        512          1 -387420489
   -40353607          0   10077696     262144]
 [         1     262144    1953125   10077696   40353607    1953125
      262144      19683        512          1]
 [         1        512      19683     262144    1953125   10077696
    40353607  134217728  387420489          0]
 [         0          0   40353607          0    1953125     262144
       19683  387420489          0          1]]

pow(k,10) =
 [[         1       1024      59049    1048576    9765625   60466176
   282475249 1073741824 -808182895 1410065408]
 [   1048576          1    9765625    9765625   60466176          0
     1048576       1024          1   60466176]
 [         1 1073741824          0  282475249 1073741824 -808182895
        1024          1      59049    1048576]
 [      1024    1048576    9765625   60466176    1048576      59049
  1073741824 -808182895          0  282475249]
 [     59049 1073741824 -808182895          0    1048576      59049
     9765625   60466176  282475249 1073741824]
 [         1       1024      59049    1048576    9765625   60466176
   282475249 1073741824 -808182895          0]
 [   1048576   60466176      59049       1024          1 -808182895
   282475249          0   60466176    1048576]
 [         1    1048576    9765625   60466176  282475249    9765625
     1048576      59049       1024          1]
 [         1       1024      59049    1048576    9765625   60466176
   282475249 1073741824 -808182895          0]
 [         0          0  282475249          0    9765625    1048576
       59049 -808182895          0          1]]
```

In [60]:
```
1 print("power(k,0) = \n",pow(k,0))
```

```
power(k,0) =
 [[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 [61]:

```
for j in range(1,10):
    print("\npower(k,"+str(j)+")\n",pow(k,1/j))
```

```
power(k,1)
 [[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
 [ 4.  1.  5.  5.  6.  0. -4. -2. -1.  6.]
 [ 1.  8.  0.  7.  8.  9. -2. -1.  3. -4.]
 [ 2.  4.  5.  6.  4.  3.  8.  9.  0.  7.]
 [ 3.  8.  9.  0.  4.  3.  5.  6.  7.  8.]
 [ 1.  2.  3.  4.  5.  6.  7.  8.  9.  0.]
 [ 4.  6.  3.  2.  1. -9. -7.  0.  6.  4.]
 [ 1.  4.  5.  6.  7.  5.  4.  3.  2.  1.]
 [ 1.  2.  3.  4.  5.  6.  7.  8.  9.  0.]
 [ 0.  0.  7.  0.  5.  4.  3.  9.  0.  1.]]

power(k,2)
 [[1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
  2.64575131 2.82842712 3.         3.16227766]
 [2.         1.         2.23606798 2.23606798 2.44948974 0.
         nan        nan        nan 2.44948974]
 [1.         2.82842712 0.         2.64575131 2.82842712 3.
         nan        nan 1.73205081        nan]
 [1.41421356 2.         2.23606798 2.44948974 2.         1.73205081
  2.82842712 3.         0.         2.64575131]
 [1.73205081 2.82842712 3.         0.         2.         1.73205081
  2.23606798 2.44948974 2.64575131 2.82842712]
 [1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
  2.64575131 2.82842712 3.         0.        ]
 [2.         2.44948974 1.73205081 1.41421356 1.                nan
         nan 0.         2.44948974 2.        ]
 [1.         2.         2.23606798 2.44948974 2.64575131 2.23606798
  2.         1.73205081 1.41421356 1.        ]
 [1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
  2.64575131 2.82842712 3.         0.        ]
 [0.         0.         2.64575131 0.         2.23606798 2.
  1.73205081 3.         0.         1.        ]]

power(k,3)
 [[1.         1.25992105 1.44224957 1.58740105 1.70997595 1.81712059
  1.91293118 2.         2.08008382 2.15443469]
 [1.58740105 1.         1.70997595 1.70997595 1.81712059 0.
         nan        nan        nan 1.81712059]
 [1.         2.         0.         1.91293118 2.         2.08008382
         nan        nan 1.44224957        nan]
 [1.25992105 1.58740105 1.70997595 1.81712059 1.58740105 1.44224957
  2.         2.08008382 0.         1.91293118]
 [1.44224957 2.         2.08008382 0.         1.58740105 1.44224957
  1.70997595 1.81712059 1.91293118 2.        ]
 [1.         1.25992105 1.44224957 1.58740105 1.70997595 1.81712059
  1.91293118 2.         2.08008382 0.        ]
 [1.58740105 1.81712059 1.44224957 1.25992105 1.                nan
         nan 0.         1.81712059 1.58740105]
 [1.         1.58740105 1.70997595 1.81712059 1.91293118 1.70997595
  1.58740105 1.44224957 1.25992105 1.        ]
 [1.         1.25992105 1.44224957 1.58740105 1.70997595 1.81712059
  1.91293118 2.         2.08008382 0.        ]
 [0.         0.         1.91293118 0.         1.70997595 1.58740105
  1.44224957 2.08008382 0.         1.        ]]

power(k,4)
 [[1.         1.18920712 1.31607401 1.41421356 1.49534878 1.56508458
  1.62657656 1.68179283 1.73205081 1.77827941]
 [1.41421356 1.         1.49534878 1.49534878 1.56508458 0.
         nan        nan        nan 1.56508458]
 [1.         1.68179283 0.         1.62657656 1.68179283 1.73205081
         nan        nan 1.31607401        nan]
 [1.18920712 1.41421356 1.49534878 1.56508458 1.41421356 1.31607401
  1.68179283 1.73205081 0.         1.62657656]
 [1.31607401 1.68179283 1.73205081 0.         1.41421356 1.31607401
  1.49534878 1.56508458 1.62657656 1.68179283]
 [1.         1.18920712 1.31607401 1.41421356 1.49534878 1.56508458
  1.62657656 1.68179283 1.73205081 0.        ]
 [1.41421356 1.56508458 1.31607401 1.18920712 1.                nan
         nan 0.         1.56508458 1.41421356]
 [1.         1.41421356 1.49534878 1.56508458 1.62657656 1.49534878
  1.41421356 1.31607401 1.18920712 1.        ]
 [1.         1.18920712 1.31607401 1.41421356 1.49534878 1.56508458
  1.62657656 1.68179283 1.73205081 0.        ]
 [0.         0.         1.62657656 0.         1.49534878 1.41421356
  1.31607401 1.73205081 0.         1.        ]]

power(k,5)
 [[1.         1.14869835 1.24573094 1.31950791 1.37972966 1.43096908
  1.47577316 1.51571657 1.55184557 1.58489319]
 [1.31950791 1.         1.37972966 1.37972966 1.43096908 0.
         nan        nan        nan 1.43096908]
 [1.         1.51571657 0.         1.47577316 1.51571657 1.55184557
         nan        nan 1.24573094        nan]
```

```
 [1.14869835 1.31950791 1.37972966 1.43096908 1.31950791 1.24573094
  1.51571657 1.55184557 0.         1.47577316]
 [1.24573094 1.51571657 1.55184557 0.         1.31950791 1.24573094
  1.37972966 1.43096908 1.47577316 1.51571657]
 [1.         1.14869835 1.24573094 1.31950791 1.37972966 1.43096908
  1.47577316 1.51571657 1.55184557 0.        ]
 [1.31950791 1.43096908 1.24573094 1.14869835 1.                nan
         nan 0.         1.43096908 1.31950791]
 [1.         1.31950791 1.37972966 1.43096908 1.47577316 1.37972966
  1.31950791 1.24573094 1.14869835 1.        ]
 [1.         1.14869835 1.24573094 1.31950791 1.37972966 1.43096908
  1.47577316 1.51571657 1.55184557 0.        ]
 [0.         0.         1.47577316 0.         1.37972966 1.31950791
  1.24573094 1.55184557 0.         1.        ]]

power(k,6)
 [[1.         1.12246205 1.20093696 1.25992105 1.30766049 1.34800615
  1.38308755 1.41421356 1.44224957 1.46779927]
 [1.25992105 1.         1.30766049 1.30766049 1.34800615 0.
         nan        nan        nan 1.34800615]
 [1.         1.41421356 0.         1.38308755 1.41421356 1.44224957
         nan        nan 1.20093696        nan]
 [1.12246205 1.25992105 1.30766049 1.34800615 1.25992105 1.20093696
  1.41421356 1.44224957 0.         1.38308755]
 [1.20093696 1.41421356 1.44224957 0.         1.25992105 1.20093696
  1.30766049 1.34800615 1.38308755 1.41421356]
 [1.         1.12246205 1.20093696 1.25992105 1.30766049 1.34800615
  1.38308755 1.41421356 1.44224957 0.        ]
 [1.25992105 1.34800615 1.20093696 1.12246205 1.                nan
         nan 0.         1.34800615 1.25992105]
 [1.         1.25992105 1.30766049 1.34800615 1.38308755 1.30766049
  1.25992105 1.20093696 1.12246205 1.        ]
 [1.         1.12246205 1.20093696 1.25992105 1.30766049 1.34800615
  1.38308755 1.41421356 1.44224957 0.        ]
 [0.         0.         1.38308755 0.         1.30766049 1.25992105
  1.20093696 1.44224957 0.         1.        ]]

power(k,7)
 [[1.         1.10408951 1.16993081 1.21901365 1.25849895 1.29170834
  1.32046925 1.34590019 1.36873811 1.38949549]
 [1.21901365 1.         1.25849895 1.25849895 1.29170834 0.
         nan        nan        nan 1.29170834]
 [1.         1.34590019 0.         1.32046925 1.34590019 1.36873811
         nan        nan 1.16993081        nan]
 [1.10408951 1.21901365 1.25849895 1.29170834 1.21901365 1.16993081
  1.34590019 1.36873811 0.         1.32046925]
 [1.16993081 1.34590019 1.36873811 0.         1.21901365 1.16993081
  1.25849895 1.29170834 1.32046925 1.34590019]
 [1.         1.10408951 1.16993081 1.21901365 1.25849895 1.29170834
  1.32046925 1.34590019 1.36873811 0.        ]
 [1.21901365 1.29170834 1.16993081 1.10408951 1.                nan
         nan 0.         1.29170834 1.21901365]
 [1.         1.21901365 1.25849895 1.29170834 1.32046925 1.25849895
  1.21901365 1.16993081 1.10408951 1.        ]
 [1.         1.10408951 1.16993081 1.21901365 1.25849895 1.29170834
  1.32046925 1.34590019 1.36873811 0.        ]
 [0.         0.         1.32046925 0.         1.25849895 1.21901365
  1.16993081 1.36873811 0.         1.        ]]

power(k,8)
 [[1.         1.09050773 1.14720269 1.18920712 1.22284454 1.2510334
  1.27537311 1.29683955 1.31607401 1.33352143]
 [1.18920712 1.         1.22284454 1.22284454 1.2510334  0.
         nan        nan        nan 1.2510334 ]
 [1.         1.29683955 0.         1.27537311 1.29683955 1.31607401
         nan        nan 1.14720269        nan]
 [1.09050773 1.18920712 1.22284454 1.2510334  1.18920712 1.14720269
  1.29683955 1.31607401 0.         1.27537311]
 [1.14720269 1.29683955 1.31607401 0.         1.18920712 1.14720269
  1.22284454 1.2510334  1.27537311 1.29683955]
 [1.         1.09050773 1.14720269 1.18920712 1.22284454 1.2510334
  1.27537311 1.29683955 1.31607401 0.        ]
 [1.18920712 1.2510334  1.14720269 1.09050773 1.                nan
         nan 0.         1.2510334  1.18920712]
 [1.         1.18920712 1.22284454 1.2510334  1.27537311 1.22284454
  1.18920712 1.14720269 1.09050773 1.        ]
 [1.         1.09050773 1.14720269 1.18920712 1.22284454 1.2510334
  1.27537311 1.29683955 1.31607401 0.        ]
 [0.         0.         1.27537311 0.         1.22284454 1.18920712
  1.14720269 1.31607401 0.         1.        ]]

power(k,9)
 [[1.         1.08005974 1.12983096 1.16652904 1.19581317 1.22028494
  1.24136582 1.25992105 1.27651801 1.29154967]
 [1.16652904 1.         1.19581317 1.19581317 1.22028494 0.
         nan        nan        nan 1.22028494]
```

```
[1.         1.25992105 0.         1.24136582 1.25992105 1.27651801
        nan        nan 1.12983096        nan]
[1.08005974 1.16652904 1.19581317 1.22028494 1.16652904 1.12983096
 1.25992105 1.27651801 0.         1.24136582]
[1.12983096 1.25992105 1.27651801 0.         1.16652904 1.12983096
 1.19581317 1.22028494 1.24136582 1.25992105]
[1.         1.08005974 1.12983096 1.16652904 1.19581317 1.22028494
 1.24136582 1.25992105 1.27651801 0.        ]
[1.16652904 1.22028494 1.12983096 1.08005974 1.                nan
        nan 0.         1.22028494 1.16652904]
[1.         1.16652904 1.19581317 1.22028494 1.24136582 1.19581317
 1.16652904 1.12983096 1.08005974 1.        ]
[1.         1.08005974 1.12983096 1.16652904 1.19581317 1.22028494
 1.24136582 1.25992105 1.27651801 0.        ]
[0.         0.         1.24136582 0.         1.19581317 1.16652904
 1.12983096 1.27651801 0.         1.        ]]

C:\Users\LH\AppData\Local\Temp\ipykernel_7300\4261224221.py:2: RuntimeWarning: invalid value encountered in power
  print("\npower(k,"+str(j)+")\n",pow(k,1/j))
```

In [62]:
```
l = np.array([[1],[2],[1],[3],[5],[2],[0],[7],[-1],[-2]])
print(l)
```

```
[[ 1]
 [ 2]
 [ 1]
 [ 3]
 [ 5]
 [ 2]
 [ 0]
 [ 7]
 [-1]
 [-2]]
```

In [63]:
```
print('j*l = \n',j*l)
```

```
j*l =
 [[  9]
 [ 18]
 [  9]
 [ 27]
 [ 45]
 [ 18]
 [  0]
 [ 63]
 [ -9]
 [-18]]
```

In [64]:
```
m = np.array([[1,2,3,4,5,6,7,8,9,10,1,2,3,4,5],[4,1,5,5,6,0,-4,-2,-1,6,6,7,8,9,0],[1,8,0,7,8,9,-2,-1,3,-4,-8,-1,2,3,4],
              [2,4,5,6,4,3,8,9,0,7,2,3,4,5,6],[3,8,9,0,4,3,5,6,7,8,0,0,1,2,3],[1,2,3,4,5,6,7,2,3,4,5,6,8,9,0],
              [4,6,3,2,1,-9,-7,0,6,4,0,0,1,-2,3],[1,4,5,6,7,5,4,3,2,1,1,2,0,0,4],[1,2,3,4,5,6,7,8,9,0,9,9,5,4,8],
              [2,3,4,5,6,7,8,9,0,1,2,4,3,4,5],[0,0,7,0,5,4,3,9,0,1,0,0,8,-9,-2],[1,2,3,4,5,6,7,8,9,0,0,9,8,4,3],
              [1,2,3,4,5,6,7,8,9,0,9,7,5,4,3],[1,2,3,4,6,-9,-8,-6,-3,-1,2,3,5,6,7],[0,0,2,1,3,4,5,6,7,8,6,5,4,9,6]])
print("Initialize matrix m")
print(m)
```

```
Initialize matrix m
[[ 1  2  3  4  5  6  7  8  9 10  1  2  3  4  5]
 [ 4  1  5  5  6  0 -4 -2 -1  6  6  7  8  9  0]
 [ 1  8  0  7  8  9 -2 -1  3 -4 -8 -1  2  3  4]
 [ 2  4  5  6  4  3  8  9  0  7  2  3  4  5  6]
 [ 3  8  9  0  4  3  5  6  7  8  0  0  1  2  3]
 [ 1  2  3  4  5  6  7  2  3  4  5  6  8  9  0]
 [ 4  6  3  2  1 -9 -7  0  6  4  0  0  1 -2  3]
 [ 1  4  5  6  7  5  4  3  2  1  1  2  0  0  4]
 [ 1  2  3  4  5  6  7  8  9  0  9  9  5  4  8]
 [ 2  3  4  5  6  7  8  9  0  1  2  4  3  4  5]
 [ 0  0  7  0  5  4  3  9  0  1  0  0  8 -9 -2]
 [ 1  2  3  4  5  6  7  8  9  0  0  9  8  4  3]
 [ 1  2  3  4  5  6  7  8  9  0  9  7  5  4  3]
 [ 1  2  3  4  6 -9 -8 -6 -3 -1  2  3  5  6  7]
 [ 0  0  2  1  3  4  5  6  7  8  6  5  4  9  6]]
```

In [65]:
```
print(m.T)
```

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

In [67]:
```
print(np.linalg.det(m.T))
print(np.linalg.det(m))
```

```
5787723947628.011
5787723947627.99
```

In [68]:
```
print(m.size) # number of element
print(m.nbytes) # size of total elements in array
print(m.shape) # the order of matrix
```

```
225
900
(15, 15)
```

II.) Accessing and Changing specific elements, row and columns in matrix

In [69]:
```
a = np.matrix([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14],[2,4,7,8,9,6,4],[0,9,0,7,5,4,3]])
print(a)
```

```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]
 [ 2  4  7  8  9  6  4]
 [ 0  9  0  7  5  4  3]]
```

In [70]:
```
# Get specific element[r,c]
print("a[0,0] =",a[0,0])
print("a[0,1] =",a[0,1])
print("a[0,2] =",a[0,2])
print("a[0,3] =",a[0,3])
print("a[0,4] =",a[0,4])
print("a[0,5] =",a[0,5])
print("a[0,6] =",a[0,6])
```

```
a[0,0] = 1
a[0,1] = 2
a[0,2] = 3
a[0,3] = 4
a[0,4] = 5
a[0,5] = 6
a[0,6] = 7
```

In [71]:
```
for i in range(0,7):
    print("a[0,"+str(i)+"] =",a[0,i])
```

```
a[0,0] = 1
a[0,1] = 2
a[0,2] = 3
a[0,3] = 4
a[0,4] = 5
a[0,5] = 6
a[0,6] = 7
```

In [72]:
```
# Get a specific row
print('0th row : ',a[0,:])
print('1st row : ',a[1,:])
print('2nd row : ',a[2,:])
print("3rd row : ",a[3,:])
```

```
0th row :  [[1 2 3 4 5 6 7]]
1st row :  [[ 8  9 10 11 12 13 14]]
2nd row :  [[2 4 7 8 9 6 4]]
3rd row :  [[0 9 0 7 5 4 3]]
```

In [73]:
```
# Get a specific column
print('0th column : \n',a[:,0])
print('1st column : \n',a[:,1])
print('2nd column : \n',a[:,2])
print('3rd column : \n',a[:,3])
print('3rd column : \n',a[:,3])
print('4th column : \n',a[:,4])
print('5st column : \n',a[:,5])
print('6nd column : \n',a[:,6])
```

```
0th column :
 [[1]
 [8]
 [2]
 [0]]
1st column :
 [[2]
 [9]
 [4]
 [9]]
2nd column :
 [[ 3]
 [10]
 [ 7]
 [ 0]]
3rd column :
 [[ 4]
 [11]
 [ 8]
 [ 7]]
3rd column :
 [[ 4]
 [11]
 [ 8]
 [ 7]]
4th column :
 [[ 5]
 [12]
 [ 9]
 [ 5]]
5st column :
 [[ 6]
 [13]
 [ 6]
 [ 4]]
6nd column :
 [[ 7]
 [14]
 [ 4]
 [ 3]]
```

In [74]:
```
print(k)
for i in range(10):
    print("row("+str(i)+") : ",k[i,:])
```

```
[[ 1  2  3  4  5  6  7  8  9 10]
 [ 4  1  5  5  6  0 -4 -2 -1  6]
 [ 1  8  0  7  8  9 -2 -1  3 -4]
 [ 2  4  5  6  4  3  8  9  0  7]
 [ 3  8  9  0  4  3  5  6  7  8]
 [ 1  2  3  4  5  6  7  8  9  0]
 [ 4  6  3  2  1 -9 -7  0  6  4]
 [ 1  4  5  6  7  5  4  3  2  1]
 [ 1  2  3  4  5  6  7  8  9  0]
 [ 0  0  7  0  5  4  3  9  0  1]]
row(0) :  [ 1  2  3  4  5  6  7  8  9 10]
row(1) :  [ 4  1  5  5  6  0 -4 -2 -1  6]
row(2) :  [ 1  8  0  7  8  9 -2 -1  3 -4]
row(3) :  [2 4 5 6 4 3 8 9 0 7]
row(4) :  [3 8 9 0 4 3 5 6 7 8]
row(5) :  [1 2 3 4 5 6 7 8 9 0]
row(6) :  [ 4  6  3  2  1 -9 -7  0  6  4]
row(7) :  [1 4 5 6 7 5 4 3 2 1]
row(8) :  [1 2 3 4 5 6 7 8 9 0]
row(9) :  [0 0 7 0 5 4 3 9 0 1]
```

In [75]:
```
for j in range(10):
    print("column("+str(j)+") : ",k[:,j])
```

```
column(0) :  [1 4 1 2 3 1 4 1 1 0]
column(1) :  [2 1 8 4 8 2 6 4 2 0]
column(2) :  [3 5 0 5 9 3 3 5 3 7]
column(3) :  [4 5 7 6 0 4 2 6 4 0]
column(4) :  [5 6 8 4 4 5 1 7 5 5]
column(5) :  [ 6  0  9  3  3  6 -9  5  6  4]
column(6) :  [ 7 -4 -2  8  5  7 -7  4  7  3]
column(7) :  [ 8 -2 -1  9  6  8  0  3  8  9]
column(8) :  [ 9 -1  3  0  7  9  6  2  9  0]
column(9) :  [10  6 -4  7  8  0  4  1  0  1]
```

In [76]:
```
print(k.trace()) # sum the diagonal values in matrix k
```

```
24
```

In [77]:
```
print(a)
```

```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]
 [ 2  4  7  8  9  6  4]
 [ 0  9  0  7  5  4  3]]
```

In [78]:
```
# Get little more fancy[row-th,startIndex:endIndex:stepSize] where stepsize is a step of order forward.
print('The fancy location of a is such that : ')
print(a[0,1:6:2])
print(a[1,0:6:1])
print(a[3,3:5:1])
print(a[2,2:-1:1])
```

```
The fancy location of a is such that :
[[2 4 6]]
[[ 8  9 10 11 12 13]]
[[7 5]]
[[7 8 9 6]]
```

In [79]:
```
# Change element using specific index
b = np.matrix([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
print(b)
```

```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]]
```

In [80]:
```
# change b15
b[1,5] = 200
print(b)
```

```
[[  1   2   3   4   5   6   7]
 [  8   9  10  11  12 200  14]]
```

In [81]:
```
# change b12 and b[column(6) with the same value]
b[1,2] = 1200
b[:,6] = 3000
print(b)
```

```
[[   1    2    3    4    5    6 3000]
 [   8    9 1200   11   12  200 3000]]
```

In [82]:
```
print(k)
```

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

In [83]:
```
k[:,0] = 200
k[:,2] = -6
print(k)
```

```
[[200   2  -6   4   5   6   7   8   9  10]
 [200   1  -6   5   6   0  -4  -2  -1   6]
 [200   8  -6   7   8   9  -2  -1   3  -4]
 [200   4  -6   6   4   3   8   9   0   7]
 [200   8  -6   0   4   3   5   6   7   8]
 [200   2  -6   4   5   6   7   8   9   0]
 [200   6  -6   2   1  -9  -7   0   6   4]
 [200   4  -6   6   7   5   4   3   2   1]
 [200   2  -6   4   5   6   7   8   9   0]
 [200   0  -6   0   5   4   3   9   0   1]]
```

In [84]:
```
k[0,:] = 100
# change value at all rows that start from index(1) to index(9) with step size = 2 with the same values.
k[:,9] = 400
k[0,9] = 100
k[1,1:9:1] = 0 # change value at 1st row with index1 to 9 about stepsize=2
k[9,:] = 100
print(k)
```

```
[[100 100 100 100 100 100 100 100 100 100]
 [200   0   0   0   0   0   0   0   0 400]
 [200   8  -6   7   8   9  -2  -1   3 400]
 [200   4  -6   6   4   3   8   9   0 400]
 [200   8  -6   0   4   3   5   6   7 400]
 [200   2  -6   4   5   6   7   8   9 400]
 [200   6  -6   2   1  -9  -7   0   6 400]
 [200   4  -6   6   7   5   4   3   2 400]
 [200   2  -6   4   5   6   7   8   9 400]
 [100 100 100 100 100 100 100 100 100 100]]
```

In [85]:
```
c = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print(c)

# Get specific element(work outside in)
print('element[0,0]:'+str(c[0,1,1]))  # c[order of matrix,row-th,column-th]
print('element[0,:] of two matrices :\n'+str(c[:,1,:]))
print('element[9,1] of two matrices :\n'+str(c[:,0,1]))
c[:,1,:] = [[9,9],[8,8]]
print('element[1:] of two matrix after change value :\n'+str(c))
```

```
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
element[0,0]:4
element[0,:] of two matrices :
[[3 4]
 [7 8]]
element[9,1] of two matrices :
[2 6]
element[1:] of two matrix after change value :
[[[1 2]
  [9 9]]

 [[5 6]
  [8 8]]]
```

III.) Initialize different types of matrix

a.) Zero Matrix:

In [86]:
```
a = np.zeros(5) # 1 row and 5 columns
print(a)
```

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

In [87]:
```
b = np.zeros(20)
print(b)
```

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

In [88]:
```python
c = np.zeros(100)
print(c)
```

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

In [89]:
```python
d = np.zeros((4,6)) # 4 rows and 6 columns
print(d)
```

```
[[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 [90]:
```python
e = np.zeros((20,20))
print(e)
```

```
[[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.]
 [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.]
 [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.]
 [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.]
 [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.]
 [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.]
 [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.]
 [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.]
 [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.]
 [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 [91]:
```python
# zero matrix with 15 rows and 10 columns
f = np.zeros((15,10))
print(f)
```

```
[[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.]
 [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.]
 [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.]
 [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 [92]:
```python
f[0,:] = 1
f[14,:] = 2
f[:,0] = 4
f[:,9] = 5
print(f)
```

```
[[4. 1. 1. 1. 1. 1. 1. 1. 1. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 0. 0. 0. 0. 0. 0. 0. 0. 5.]
 [4. 2. 2. 2. 2. 2. 2. 2. 2. 5.]]
```

In [93]:
```
# 3 zero matrices with 5 rows and 7 columns
g = np.zeros((3,5,7))
print(g)
```

```
[[[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. 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. 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 [94]:

```
# 20 zero matrices with 7 rows and 3 columns
h = np.zeros((4,5,7,3))
print(h)
```

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

b.) One Matrix

In [95]:

```python
d = np.ones(20)
print(d)
```

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

In [96]:
```
e = np.ones(100)
print(e)
```

```
[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 [97]:
```
# one matrix with 2 rows and 3 columns
f = np.ones((2,3))
print(f)
```

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

In [98]:
```
# 2 one matrices with 3 rows and 5 columns
g = np.ones((2,3,5))
print(g)
```

```
[[[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 [99]:
```
h = np.ones((4,6,10))
print(h)
```

```
[[[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. 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 [100]:
```
1 i = np.ones((10,10,10))
2 print(i)
```

```
[[[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. 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. 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. 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. 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.]
  [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. 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 [101]:

```
1 j = np.ones((4,2,2))
2 print(j)
3 print(j.dtype)
```

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

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]]
float64
```

In [102]:

```
1 j = np.ones((4,2,2),dtype="int32")
2 print(j)
3 print(j.dtype)
```

```
[[[1 1]
  [1 1]]

 [[1 1]
  [1 1]]

 [[1 1]
  [1 1]]

 [[1 1]
  [1 1]]]
int32
```

c.) Any other number of matrix

In [103]:

```
1 x=np.full((2,2),99,dtype="float32")
2 print(x)
```

```
[[99. 99.]
 [99. 99.]]
```

In [104]:
```
y = np.full((10,10,10),3)
print(y)
```

```
[[[3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]]

 [[3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]]

 [[3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]]

 [[3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]]

 [[3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]]

 [[3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]]

 [[3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]]

 [[3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
  [3 3 3 3 3 3 3 3 3 3]
```

```
 [3 3 3 3 3 3 3 3 3 3]]

[[3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]]

[[3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]
 [3 3 3 3 3 3 3 3 3 3]]]
```

In [105]:

```
z = np.full((20,20),-1)
print(z)
```

```
[[-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 -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 -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 [106]:

```
m = np.full((30,30),4)
print(m)
```

```
[[4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]]
```

In [107]:
```
n = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14],[0,9,0,8,6,5,-3]])
print(np.full_like(n,4))
```

```
[[4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4]]
```

In [108]:
```
print(np.full(n.shape,4))
```

```
[[4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4]
 [4 4 4 4 4 4 4]]
```

In [109]:
```
print(np.full_like(n,5))
```

```
[[5 5 5 5 5 5 5]
 [5 5 5 5 5 5 5]
 [5 5 5 5 5 5 5]]
```

In [110]:
```
# np.full_like(n,a) = np.full(n.shape,a), where a is constant
```

d.) Random number in matrix

In [111]:
```
# random number on(0,1) with 4 rows and 2 columns
print(np.random.rand(4,2))
```

```
[[0.29836468 0.95491516]
 [0.81032453 0.09527103]
 [0.71370772 0.158331  ]
 [0.85871714 0.06759747]]
```

In [112]:
```
# random number on(0,1) with 2 rows and 3 columns for 4 matrices.
print(np.random.rand(4,2,3))
```

```
[[[0.53259878 0.23340412 0.13684431]
  [0.14139531 0.9771577  0.84943649]]

 [[0.06199424 0.23966948 0.26959797]
  [0.44991392 0.65597107 0.90445552]]

 [[0.83702931 0.27365017 0.44604798]
  [0.40768061 0.69318887 0.75366238]]

 [[0.11617681 0.48146572 0.50454002]
  [0.78640471 0.28507069 0.9655753 ]]]
```

In [113]:
```
# random on matrix n on the interval(0,1)
print(n)
print(np.random.random_sample(n.shape))
```

```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]
 [ 0  9  0  8  6  5 -3]]
[[0.37375404 0.37634546 0.68646076 0.35779359 0.06100001 0.5302908
  0.25139477]
 [0.79861591 0.00634484 0.84074891 0.64250249 0.29214985 0.04423007
  0.92291453]
 [0.26271885 0.61875452 0.49268848 0.37511463 0.9206648  0.39932682
  0.34285574]]
```

In [114]:
```
# random integer value on the interval(-4,8) with 3 matrices about 3 rows and 3 columns
print(np.random.randint(-4,8,size=(3,3,3)))
```

```
[[[ 1  0 -3]
  [ 4  3 -1]
  [-4 -4  7]]

 [[-4 -2  6]
  [-2  0 -1]
  [ 0  0  2]]

 [[ 2  4  5]
  [-3 -4  4]
  [-1 -3  3]]]
```

In [118]:
```
# random integer value on the interval(1,100)
print(np.random.randint(1,100,size=(3,4,5)))
```

```
[[[94 59 98 48 40]
  [35 36 60 92 39]
  [78 95 12 40 55]
  [47 15 17  5 62]]

 [[79 66 30 44 68]
  [68 27 23 43 66]
  [45 27 95 43 11]
  [43  5 95 41  2]]

 [[35 57 94 33  2]
  [ 6  3  6 32  3]
  [44 36 74 80 24]
  [86 27 28 63 28]]]
```

In [119]:
```
print(np.random.randint(1,1000,size=(15,15)))
print("============================================================")
print(np.linalg.det(np.random.randint(1,100,size=(15,15))))
```

```
[[689 594 383 760 850 841 255 891 931 143 801 573 693 109  11]
 [750  34  81 945 346 294 714  33 746 721 654 337 834 458 506]
 [126 640 367 571 780 617 906 610 276 868 268 752 795 791   3]
 [781 626 579 496 491 665 350 687 336 178 546 668 533 169 318]
 [834 334 604 930 931 681 869 201 278  80 265 171 271 803 585]
 [916 365 519 746 748 787 953 342 840 465  82 580 811  89  51]
 [813 725 655 877 216 738 654 805 171 393 795   5 664 855 334]
 [902 811 858 953 142 216 672 480 731 235 736 876 777 435 792]
 [421 976 720 114 757 898 579 782 986 462 610 217 328 601 354]
 [868 479 340 674 233 477 363 207 986 390 848 968  20 927 168]
 [ 21 383 252 281 476 622 803 115 893 828 544 529 176 435  32]
 [678 289 719 121 737 990 953 312 390 988 222 139 225 223 689]
 [261 467 793 348 226 330 514 138 111 987 716 460 391 857 932]
 [602 904 252 330 208 645 541 201 308 496 446  14 445 428 767]
 [240 559 115 383 612 355 403 725 281 823 891 726 849 896 143]]
============================================================
-3.043883811156688e+28
```

In [120]:
```
# np.random.rand(number of matrix,row-th,column-th), if no input number of matrix it will display just a matrix default.
# np.random.random_sample(m.shape) : this case we need input m.shape such the form (a,b) where a,b are row-th and column-th
# respectively in order.
# np.random.randint(startValue,endValue,size(#matrix,row-th,column-th))
```

e.) Identity Matrix

In [121]:
```
print(np.identity(3))
```

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

In [122]:
```
print(np.identity(10))
```

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

In [123]:
```
print(np.identity(20))
```

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

In [124]:
```
print(np.identity(20).trace())
```

```
20.0
```

In [125]:
```
print(np.identity(100).trace())
```

```
100.0
```

In [126]:
```
print(pow(np.identity(20),2))
```

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

In [127]:
```
print(np.identity(20)**100)
```

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

In [128]:
```
print(np.identity(2))
```

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

In [129]:
```
# Repeat an array
a = np.array([1,2,3])
print(np.repeat(a,10,0))
# 0 is default value for the axis where set of axis = {-1,0,1}
```

```
[1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3]
```

In [130]:
```
# np.repeat(matrix,#repetition,axis) : it displays that has the repetition.
print(x)
print("\n")
print(np.repeat(x,7,0))
```

```
[[99. 99.]
 [99. 99.]]


[[99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]
 [99. 99.]]
```

In [131]:
```
table = np.ones((5,5))
print(table)
```

```
[[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 [132]:
```
z = np.zeros((3,3))
z[1,1] = 9
print(z)
```

```
[[0. 0. 0.]
 [0. 9. 0.]
 [0. 0. 0.]]
```

In [133]:
```
table[1:4,1:4] = z
print(table)
```

```
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 9. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

In [134]:
```
w = np.identity(4)
v = np.identity(3)
x = np.ones((10,10))
y = np.zeros((8,8))
y[4,4] = 2
x[1:9,1:9] = y
x[1:5,1:5] = w
x[6:9,6:9] = v
print(x)
print('\n')
print(x.T)
print('\n')
print(np.linalg.det(x))
```

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 1. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 2. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]


[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 1. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 1. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 2. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 1. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]


0.0
```

f.) Copying matrix

In [135]:
```
a = np.array([1,2,3])
b = a
b[0] = 100
print(a)
```

```
[100   2   3]
```

In [136]:
```
b = a.copy()
b[0] = 99
print(b)
```

```
[99  2  3]
```

In [137]:
```
c = np.matrix([[1,2,3],[9,8,7],[0,7,6]])
print(c)
print("\n")
c = b.copy()
print(c)
```

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


[99  2  3]
```

IV.) Mathematics

In [138]:
```
y = np.array([3,4,5,6])
print(y)
```

```
[3 4 5 6]
```

In [139]:
```
y= y+2
print(y)
```

```
[5 6 7 8]
```

In [140]:
```
m = y-14
print(m)
```

```
[-9 -8 -7 -6]
```

In [141]:
```
w = y*2
print(w)
```

```
[10 12 14 16]
```

In [142]:
```
x = y/2
print(x)
```

```
[2.5 3.  3.5 4. ]
```

In [143]:
```
from math import sqrt
a = np.matrix([1,-1,-1/2,1/2,sqrt(2)/2,sqrt(3)/2])
print(np.arccos(a))
print(np.arcsin(a))
print(np.arctan(a))
print(1/np.arctan(a))
```

```
[[0.         3.14159265 2.0943951  1.04719755 0.78539816 0.52359878]]
[[ 1.57079633 -1.57079633 -0.52359878  0.52359878  0.78539816  1.04719755]]
[[ 0.78539816 -0.78539816 -0.46364761  0.46364761  0.61547971  0.71372438]]
[[ 1.27323954 -1.27323954 -2.15681043  2.15681043  1.62474893  1.40110108]]
```

V.) Linear Algebra

a.) Matrix multiplication

In [144]:
```
a = np.ones((2,3))
print(a)
```

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

In [145]:
```
b = np.full((3,2),2)
print(b)
```

```
[[2 2]
 [2 2]
 [2 2]]
```

In [146]:
```
result = np.matmul(a,b)
print(result)
```

```
[[6. 6.]
 [6. 6.]]
```

In [149]:
```
c = np.array([[1,2,3,4,5,6,7,8,9,10,1,2,3,4,5],[4,1,5,5,6,0,-4,-2,-1,6,6,7,8,9,0],[1,8,0,7,8,9,-2,-1,3,-4,-8,-1,2,3,4],
              [2,4,5,6,4,3,8,9,0,7,2,3,4,5,6],[3,8,9,0,4,3,5,6,7,8,0,0,1,2,3],[1,2,3,4,5,6,7,2,3,4,5,6,8,9,0],
              [4,6,3,2,1,-9,-7,0,6,4,0,0,1,-2,3],[1,4,5,6,7,5,4,3,2,1,1,2,0,0,4],[1,2,3,4,5,6,7,8,9,0,9,9,5,4,8],
              [2,3,4,5,6,7,8,9,0,1,2,4,3,4,5],[0,0,7,0,5,4,3,9,0,1,0,0,8,-9,-2],[1,2,3,4,5,6,7,8,9,0,0,9,8,4,3],
              [1,2,3,4,5,6,7,8,9,0,9,7,5,4,3],[1,2,3,4,6,-9,-8,-6,-3,-1,2,3,5,6,7],[0,0,2,1,3,4,5,6,7,8,6,5,4,9,6]])
d = np.array([[1,2,3,4,5,6,7,8,9,10,1,2,3,4,5]])
result = np.matmul(c,d.T)
print(c,"\n")
print(d,"\n")
print(result)
```

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

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

[[440]
 [158]
 [132]
 [355]
 [336]
 [300]
 [ 39]
 [218]
 [383]
 [310]
 [151]
 [358]
 [354]
 [-53]
 [369]]
```

In [150]:
```
e = np.array([[1,2,3,4,5,6,7,8,9,10,1,2,3,4,5],[4,1,5,5,6,0,-4,-2,-1,6,6,7,8,9,0]])
result = np.matmul(c,e.transpose())
print(result)
```

```
[[440 158]
 [158 390]
 [132  66]
 [355 193]
 [336 124]
 [300 277]
 [ 39  89]
 [218 107]
 [383 211]
 [310 148]
 [151  24]
 [358 181]
 [354 197]
 [-53 245]
 [369 226]]
```

In [151]:
```
g = np.array([[1,2,3,4,5,6,7,8,9,10,1,2,3,4,5],
              [1,2,3,4,5,6,7,2,3,4,5,6,8,9,0],
              [4,6,3,2,1,-9,-7,0,6,4,0,0,1,-2,3],
              [4,1,5,5,6,0,-4,-2,-1,6,6,7,8,9,0]])
result = np.matmul(c,g.transpose())
print(result)
```

```
[[440 300  39 158]
 [158 277  89 390]
 [132 113  17  66]
 [355 294  20 193]
 [336 210 109 124]
 [300 375 -41 277]
 [ 39 -41 262  89]
 [218 174  17 107]
 [383 358  10 211]
 [310 284 -51 148]
 [151  96  -7  24]
 [358 337  -2 181]
 [354 346  -5 197]
 [-53  47 168 245]
 [369 322  18 226]]
```

In [152]:
```
h = np.array([[1,2,3,4,5,6,7,8,9,10,1,2,3,4,5],[1,2,3,4,5,6,7,2,3,4,5,6,8,9,0],[4,6,3,2,1,-9,-7,0,6,4,0,0,1,-2,3],
              [4,1,5,5,6,0,-4,-2,-1,6,6,7,8,9,0],[0,0,7,0,5,4,3,9,0,1,0,0,8,-9,-2],[1,2,3,4,5,6,7,8,9,0,0,9,8,4,3]])
result = np.matmul(c,h.transpose())
print(result)
```

```
[[440 300  39 158 151 358]
 [158 277  89 390  24 181]
 [132 113  17  66  38 175]
 [355 294  20 193 154 312]
 [336 210 109 124 156 255]
 [300 375 -41 277  96 337]
 [ 39 -41 262  89  -7  -2]
 [218 174  17 107 122 213]
 [383 358  10 211 151 446]
 [310 284 -51 148 170 331]
 [151  96  -7  24 330 185]
 [358 337  -2 181 185 455]
 [354 346  -5 197 161 413]
 [-53  47 168 245 -92 -13]
 [369 322  18 226  61 326]]
```

b.) Statistic

In [158]:
```
a = np.matrix([[1,2,3],[4,5,6],[7,8,9]])
print(a)
```

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

In [159]:
```
print(np.min(a[0,:])) # show minimum value at zero(th) row and any column
print(np.min(a[1,:]))
print(np.min(a[2,:]))
```

```
1
4
7
```

In [160]:
```
print(np.min(a[:,0]))
print(np.min(a[:,1]))
print(np.min(a[:,2]))
```

```
1
2
3
```

In [161]:
```
print(np.min(a))
print(np.max(a))
print(np.average(a))
```

```
1
9
5.0
```

In [162]:
```
print(a)
```

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

In [163]:
```
print(np.min(a,axis=-1))
```

```
[[1]
 [4]
 [7]]
```

In [164]:
```
print(np.min(a,axis=0))
```

```
[[1 2 3]]
```

In [165]:
```
print(np.min(a,axis=1))
```

```
[[1]
 [4]
 [7]]
```

In [166]:
```
print(np.max(a,axis=0))
```

```
[[7 8 9]]
```

In [167]:
```
print(np.average(a,axis=0))
```

```
[[4. 5. 6.]]
```

In [168]:
```
print(np.average(a,axis=1))
```

```
[[2.]
 [5.]
 [8.]]
```

In [169]:
```
Sum = np.sum(a)
print("sum all element in a = ",Sum)
```

```
sum all element in a =  45
```

In [170]:
```
S = np.sum(a,axis=0)
print(S) # show the total value for each column
```

```
[[12 15 18]]
```

c.) Reorganizing array

In [171]:
```
a1 = np.matrix([[1,2,3,4],[5,6,7,8]])
print(a1)
```

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

In [172]:
```
a2 = a1.reshape((8,1))
print(a2)
```

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

In [173]:
```
a3 = a1.reshape((4,2))
print(a3)
```

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

In [174]:
```
a4 = a1.reshape((1,8))
print(a4)
```

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

In [175]:
```
# vertically Stacking Vectors(Combine together)
print("vertical stack")
v1 = np.matrix([1,2,3,4])
v2 = np.matrix([5,6,7,8])
x = np.vstack([v1,v2])
print(x)
```

```
vertical stack
[[1 2 3 4]
 [5 6 7 8]]
```

In [176]:
```
y = np.stack([v1,v2,v1,v2])
print(y)
```

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

In [177]:
```
# Horizontal Stack
print('Horizontal Stack')
h1 = np.ones((2,4))
print(h1)
```

```
Horizontal Stack
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

In [178]:
```
h2 = np.zeros((2,2))
print(h2)
```

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

In [179]:
```
# After horizontal stack
print(np.hstack((h1,h2)))
```

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

e.) Boolean masking and advanced indexing

In [180]:
```
print(k)
```

```
[[100 100 100 100 100 100 100 100 100 100]
 [200   0   0   0   0   0   0   0   0 400]
 [200   8  -6   7   8   9  -2  -1   3 400]
 [200   4  -6   6   4   3   8   9   0 400]
 [200   8  -6   0   4   3   5   6   7 400]
 [200   2  -6   4   5   6   7   8   9 400]
 [200   6  -6   2   1  -9  -7   0   6 400]
 [200   4  -6   6   7   5   4   3   2 400]
 [200   2  -6   4   5   6   7   8   9 400]
 [100 100 100 100 100 100 100 100 100 100]]
```

In [181]:
```
a = k[k<10]
print(a)
```

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

In [182]:
```
print(k[k>100])
```

```
[200 400 200 400 200 400 200 400 200 400 200 400 200 400 200 400]
```

In [183]:
```
print(k[k>200])
```

```
[400 400 400 400 400 400 400 400]
```

In [184]:
```
# load data from file
file = np.genfromtxt('data2.txt',delimiter=',')
file.astype('int32')
print(file)
```

```
[  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.  38.  40.  78.  90.
   9.  87.  56.  78.  99. 900. 987. 865. 453. 100.]
```

In [185]:
```
print(file>50)
```

```
[False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False  True  True False  True  True  True  True  True
  True  True  True  True]
```

In [186]:
```
# test condition whether it is true or false with value < 59
print(file<50)
```

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True False False  True False False False False False
 False False False False]
```

In [187]:
```
# Show the elements that are smaller than 10
print(file[file<10])
```

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

In [188]:
```
# show the element that are smaller than 50.
print(file[file<50])
```

```
[ 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. 38. 40.  9.]
```

In [189]:
```
print(file[file>100])
```

```
[900. 987. 865. 453.]
```

In [190]:
```
x = np.all(file>50,axis=0)
print(x)
```

```
False
```

In [191]:
```
# Show the reality of element that are neither larger than 50 and less than 100.
y = (~(file>50)&(file<100))
print(y)
```

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True False False  True False False False False False
 False False False False]
```

In [192]:
```
# ~ = not. For example a & b = true then, ~(a & b) = false.
z = ~y
print(z)
```

```
[False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False  True  True False  True  True  True  True  True
  True  True  True  True]
```

f.) Index with a list

In [193]:
```
a = np.array([1,2,3,4,5,6,7,8,9])
print(a)
# display the value at index[1], index[2] and index[8]
print(a[[1,2,8]])
```

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

In [195]:

```
y = np.array([1,2,3,4,5])
for i in range(1,6):
    if(i==1):
        x = np.array([i,i+1,i+2,i+3,i+4])
    elif(i==2):
        x = np.array([i+4,i+5,i+6,i+7,i+8])
    elif(i==3):
        x = np.array([i+8,i+9,i+10,i+11,i+12])
    elif(i==4):
        x = np.array([i+13,i+14,i+15,i+16,i+17])
    elif(i==5):
        x = np.array([i+17,i+18,i+19,i+20,i+21])
    print(x)
```

```
[1 2 3 4 5]
[ 6  7  8  9 10]
[11 12 13 14 15]
[17 18 19 20 21]
[22 23 24 25 26]
```