```python
'''1. Create and transform vectors and matrices (the transpose vector (matrix) c
transpose of a vector (matrix))
'''
import numpy as np
a=np.array([[1,2,3],[4,5,6],[7,8,9]]) #Matrix
b=np.array([1,2,3]) #Vector

print('transpose of matrix\n',np.transpose(a)) #Transpose Of Matrix
print('transpose of matrix\n',np.transpose(b)) #Transpose Of Vector

print('conjugate transpose of a matrix\n',np.conj(a).T) #Conjugate Transpose Of
print('conjugate transpose of a vector\n',np.conj(b).T) #Conjugate Transpose Of
```

```
transpose of matrix
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
transpose of matrix
 [1 2 3]
conjugate transpose of a matrix
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
conjugate transpose of a vector
 [1 2 3]
```

```python
'''2. Generate the matrix into echelon form and find its rank.'''

import sympy as sp
import numpy as np

A=[[1,2,3],[4,5,6],[7,8,9]]
b=np.array(A)
c=sp.Matrix(A)


print('Row Reduced Echelon Form of Matrix\n', c.rref()[0]) #Reduced Row Echelon

print("Rank Of Matrix A is :\n" , np.linalg.matrix_rank(b)) #Rank Of Matrix
```

```
Row Reduced Echelon Form of Matrix
 Matrix([[1, 0, -1], [0, 1, 2], [0, 0, 0]])
Rank Of Matrix A is :
 2
```

```python
#3. Find cofactors, determinant, adjoint and inverse of a matrix.

import numpy as np
a=np.array([[1, 9, 3],[2, 5, 4],[3, 7, 8]])

b=print('determinant of a matrix\n',np.linalg.det(a))#Determinant Of Matrix
c=print('inverse of a matrix\n',np.linalg.inv(a))#Inverse Of Matrix
d=print('cofactors of a matrix\n',np.linalg.inv(a).T*np.linalg.det(a))#Cofactors
print('adjoint of matrix\n',np.linalg.inv(a)*np.linalg.det(a))#Adjoint Of Matrix
```

```
determinant of a matrix
 -27.0
inverse of a matrix
 [[-0.44444444  1.88888889 -0.77777778]
 [ 0.14814815  0.03703704 -0.07407407]
 [ 0.03703704 -0.74074074  0.48148148]]
cofactors of a matrix
 [[ 12.  -4.  -1.]
 [-51.  -1.  20.]
 [ 21.   2. -13.]]
adjoint of matrix
 [[ 12. -51.  21.]
 [ -4.  -1.   2.]
 [ -1.  20. -13.]]
```

```python
import numpy as np
import sys

n = int(input('Enter number of unknowns: '))
a = np.zeros((n,n+1))
x = np.zeros(n)

# Reading augmented matrix coefficients
print('Enter Augmented Matrix Coefficients:')
for i in range(n):
    for j in range(n+1):
        a[i][j] = float(input( 'a['+str(i)+']['+ str(j)+']='))

#Applying Gauss Elimination
for i in range(n):
    if a[i][i] == 0.0:
        sys.exit('Zero Division Error')

    for j in range(i+1, n):
        ratio = a[j][i]/a[i][i]

        for k in range(n+1):
            a[j][k] = a[j][k] - ratio * a[i][k]

#Back Substitution
x[n-1] = a[n-1][n]/a[n-1][n-1]

for i in range(n-2,-1,-1):
    x[i] = a[i][n]

    for j in range(i+1,n):
        x[i] = x[i] - a[i][j]*x[j]

    x[i] = x[i]/a[i][i]

# Displaying solution
for i in range(n):
    print('\n X%d = %0.2f' %(i,x[i]), end = '\t')
```

```
Enter number of unknowns: 2
Enter Augmented Matrix Coefficients:
a[0][0]=1
a[0][1]=2
a[0][2]=3
a[1][0]=4
a[1][1]=5
a[1][2]=6

 X0 = -1.00
 X1 = 2.00
```

```python
#5.gauss jordan elimination
import numpy as np
import sys


n=int(input('enter a number'))
a=np.zeros((n,n+1))
x=np.zeros(n)
print('enter augmented matrix coefficient')
for i in range(n):
  for j in range(n+1):
    a[i][j]=float(input('a['+str(i)+']['+str(j)+']='))
#applying gauss jordan elimination
for i in range(n):
  if a[i][j]==0.0:
    sys.exit('divide by zero')
for j in range(n):
  if i!=j:
    ratio=a[j][i]/a[i][i]
  for c in range(n+1):
      a[j][c]=a[j][c]
for i in range(n):
  x[i]=a[i][n]/a[i][i]
print('solution =')
for i in range(n):
  print('x%d=%0.2f'%(i,x[i]),end = '\t')
```

```
enter a number2
enter augmented matrix coefficient
a[0][0]=1
a[0][1]=2
a[0][2]=3
a[1][0]=4
a[1][1]=6
a[1][2]=9
solution =
x0=3.00 x1=1.50
```

```python
#6. Null space,Row space,Column space,LeftNull space
import sympy as sp

a=[[1, 0, 1, 3], [2, 3, 4, 7], [-1, -3, -3, -4]]

b=sp.Matrix(a)

print("Matrix a=\n",b)

def Null():
   print("\nNull Space Of Matrix a:-\n",b.
   nullspace())

def Row():
        print("\nRow Space Of Matrix a:-\n",b.rowspace())

def Col():
        print("\nColumn Space Of Matrix a:-\n",b.columnspace())

def LeftNull():
        c=b.T
        print("\nLeft Null Space Of Matrix a:-\n",c.nullspace())


Null()
Row()
Col()
LeftNull()
```

```
Matrix a=
 Matrix([[1, 0, 1, 3], [2, 3, 4, 7], [-1, -3, -3, -4]])

Null Space Of Matrix a:-
 [Matrix([
[  -1],
[-2/3],
[   1],
[   0]]), Matrix([
[  -3],
[-1/3],
[   0],
[   1]])]

Row Space Of Matrix a:-
 [Matrix([[1, 0, 1, 3]]), Matrix([[0, 3, 2, 1]])]

Column Space Of Matrix a:-
 [Matrix([
[ 1],
[ 2],
[-1]]), Matrix([
[ 0],
[ 3],
[-3]])]

Left Null Space Of Matrix a:-
 [Matrix([
[-1],
[ 1],
[ 1]])]
```

```python
#10. encryption & decryption of matrix
import numpy as np

def encrypt():
        a={1:'A',2:'B',3:'C',4:'D',5:'E',6:'F',7:'G',8:'H',9:'I',10:'J',11:'K',12:'L',13:'M',14:'N',15:'O',16:'P',17:'Q',18:'R',19:'S',20:'T',21:'U',22:'V',23:'W',24:'X',25:'Y',26:'Z',27:'

        b=[['L','I','N','E','A'],['R',' ','A','L','G'],['E','B','R','A',' '],['I','S',' ','F','U',],['N',' ',' ',' ',' ']]
        c=[]
        for x in range(5):
                for y in range(5):
                        for i in a:
                                if a[i]==str(b[x][y]):
                                        c.append(i)
        z=np.array([[c[0:5]],[c[5:10]],[c[10:15]],[c[15:20]],[c[20:25]]])
        print('\nOriginal Matrix is:-\n',np.array(b))
        print('\nEncrypted Matrix is:-\n',z)

def decrypt():
        a={1:'A',2:'B',3:'C',4:'D',5:'E',6:'F',7:'G',8:'H',9:'I',10:'J',11:'K',12:'L',13:'M',14:'N',15:'O',16:'P',17:'Q',18:'R',19:'S',20:'T',21:'U',22:'V',23:'W',24:'X',25:'Y',26:'Z',27:'

        b=[[12,9,14,5,1],[18,27,1,12,7],[5,2,18,1,27],[9,19,27,6,21],[14,27,27,27,27]]
        c=[]
        for i in range(5):
                for j in range(5):
                        x=b[i][j]
                        if x in a.keys():
                                c.append(a[x])

        z=np.array([[c[0:5]],[c[5:10]],[c[10:15]],[c[15:20]],[c[20:25]]])
        print('\nOriginal Matrix is:-\n',np.array(b))
        print('\nDecrypted Matrix is:-\n',z)

encrypt()
decrypt()
```

```
Original Matrix is:-
 [['L' 'I' 'N' 'E' 'A']
 ['R' ' ' 'A' 'L' 'G']
 ['E' 'B' 'R' 'A' ' ']
 ['I' 'S' ' ' 'F' 'U']
 ['N' ' ' ' ' ' ' ' ']]

Encrypted Matrix is:-
 [[[12  9 14  5  1]]

 [[18 27  1 12  7]]

 [[ 5  2 18  1 27]]

 [[ 9 19 27  6 21]]

 [[14 27 27 27 27]]]

Original Matrix is:-
 [[12  9 14  5  1]
 [18 27  1 12  7]
 [ 5  2 18  1 27]
 [ 9 19 27  6 21]
 [14 27 27 27 27]]

Decrypted Matrix is:-
 [[['L' 'I' 'N' 'E' 'A']]

 [['R' ' ' 'A' 'L' 'G']]

 [['E' 'B' 'R' 'A' ' ']]

 [['I' 'S' ' ' 'F' 'U']]
```

```python
#11. gradient
import numpy as np

def Grad():
        a=[1,2,4,7,11,16]
        b=np.array(a)
        print("\n Gradient Of Scalar Field Is:-\n",np.gradient(b))

Grad()
```

```
Gradient Of Scalar Field Is:-
[1.  1.5 2.5 3.5 4.5 5. ]
```

```python
#12. divergence
import numpy as np

def Diver(a):
        return np.ufunc.reduce(np.add,np.gradient(a))

c=[1,2,3,4,5,6,7,8,9]
b=np.array(c)
print("\n Divergence:-\n",Diver(b))
```

Divergence:-
9.0

```python
#13. curl
from sympy.physics.vector import ReferenceFrame
from sympy.physics.vector import curl


def Curl():
        R = ReferenceFrame('R')
        F = R[1]**2 * R[2] * R.x - R[0]*R[1] * R.y +R[2]**2 * R.z
        CURL=curl(F,R)
        print('\nCurl Will Be:-\n',CURL)

Curl()
```

Curl Will Be:-

```
 R_y**2*R.y + (-2*R_y*R_z - R_y)*R.z
```