TITLE OF THE MINI PROJECT

2D TRANSFORMATION OF OBJECTS

DEPARTMENT NAME: COMPUTER SCIENCE AND ENGINEERING

COLLEGE NAME: ST. THOMAS' COLLEGE OF ENGINEERING & TECHNOLOGY

ADVISOR/SUPERVISOR: Dr. MOUSUMI DUTT

GROUP DETAILS:

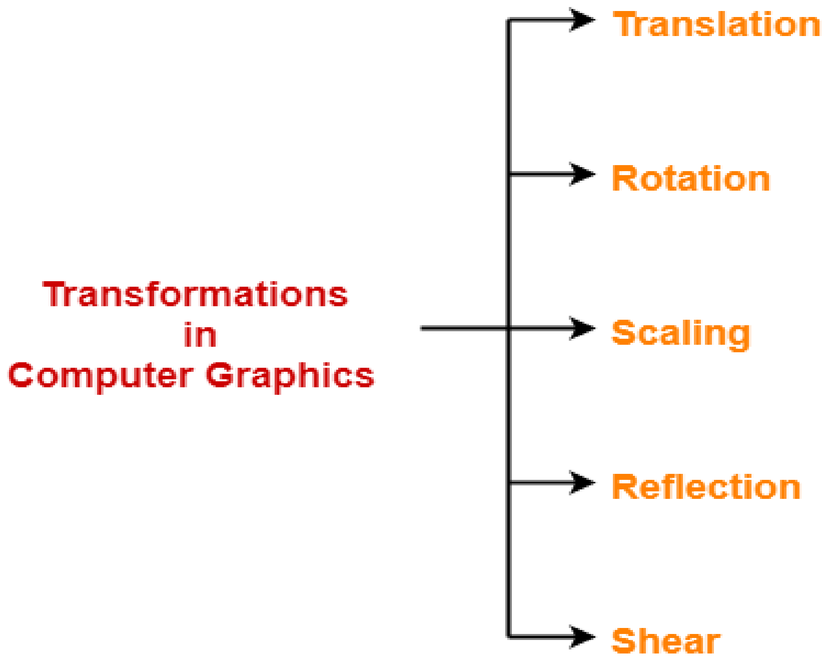
NAME	**ROLL NUMBER**
1) RUPAK DEB	21
2) SUBHANKAR GHOSH	32
3) CHITRANGADA SARKAR	63
4) SWARNALI SAHA	61
5) DEBARGHYA NANDI	52
6) ARPITA ROY	57

CONTENTS **PAGE NO.**

OBJECTIVES:-

- Transformation is a process of modifying and re-positioning the existing graphics. 2D Transformations take place in a two-dimensional plane.
- Transformations are helpful in changing the position, size, orientation, shape etc. of the object.
- 2D Translation is a process of moving an object from one position to another in a two-dimensional plane.

1. TRANSLATION
2. ROTATION
3. SCALING
4. REFLECTION
5. SHEARING



MOTIVATION:-

In Computer Graphics, transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation. Transformation means changing some graphics into something else by applying rules. When a transformation takes place on a 2D plane, it is called 2D transformation. Transformation of an object in computer graphics is based on mathematical theory and uses an important concept related to matrix theory. In general, there are two ways to represent the transformation of an object -

(i) Transformations are done by using the rules of co-ordinates axes
(ii) Transformations are done by representing in the matrix form.

2D Transformation in Computer Graphics could be applied in several projects, with a variety of goals, such as training, treatments, aid in learning, digital inclusion, simulations, among others. We realized the importance of visual programming and were motivated to do projects in this area. Computer Graphics is beyond beautiful and fun designs on screen. We can use all its potential for various projects. It just depends on our ability to create.

<u>**BRESENHAM'S LINE DRAWING ALGORITHM**</u>:-

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly. In this method, next pixel selected is that one who has the least distance from true line.

Given the starting and ending coordinates of a line, Bresenham Line Drawing Algorithm attempts to generate the points between the starting and ending coordinates.

Given-

Starting coordinates = (X_0, Y_0)
Ending coordinates = (X_n, Y_n)
The points generation using Bresenham Line Drawing Algorithm involves the following steps-

Step 1:

Calculate ΔX and ΔY from the given input.
These parameters are calculated as-
$\Delta X = X_n - X_0$
$\Delta Y = Y_n - Y_0$

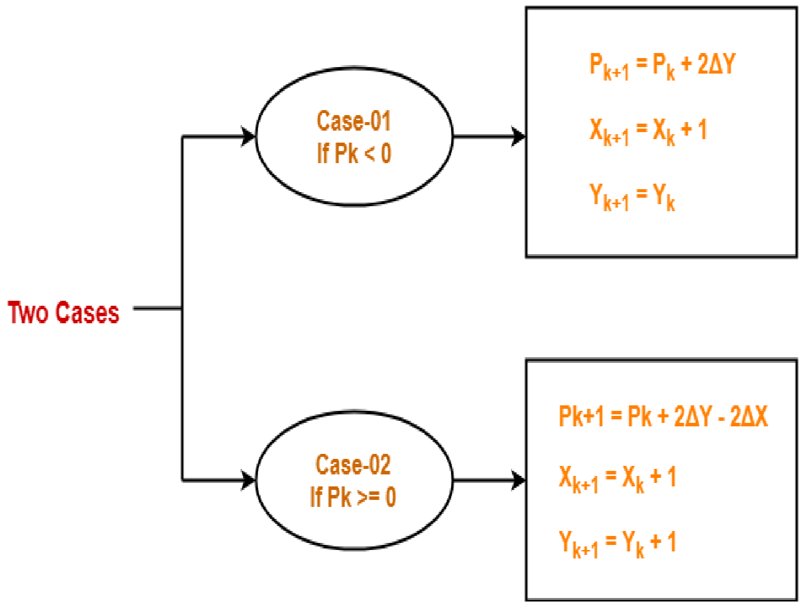
Step 2:

Calculate the decision parameter P_k.
It is calculated as-
$P_k = 2\Delta Y - \Delta X$

Step 3:

Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}). Find the next point depending on the value of decision parameter Pk.

Follow the below two cases-



Step 4:

Keep repeating Step-03 until the end point is reached or number of iterations equals to (ΔX-1) times.

<u>MIDPOINT CIRCLE DRAWING ALGORITHM:-</u>

Given the center point and radius of circle, Mid-Point Circle Drawing Algorithm attempts to generate the points of one octant. The points for other octants are generated using the eight-symmetry property.
Given-

Centre point of Circle = (X_0, Y_0)
Radius of Circle = R
The points generation using Mid-Point Circle Drawing Algorithm involves the following steps-

<u>Step 1</u>:

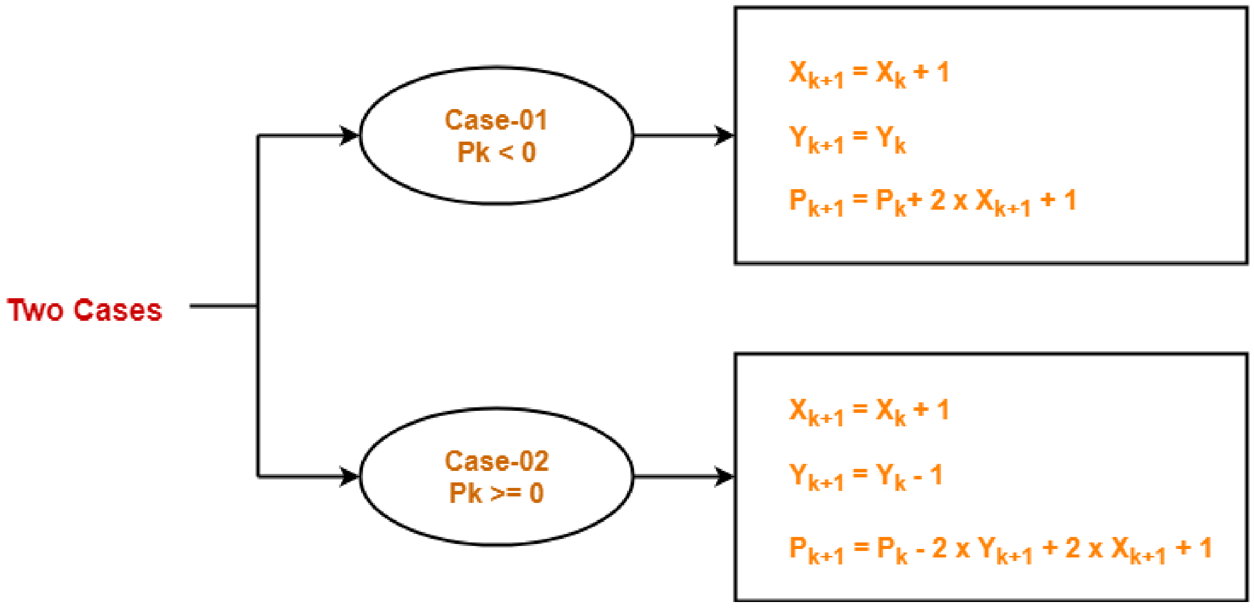
Assign the starting point coordinates (X_0, Y_0) as-
$X_0 = 0$
$Y_0 = R$

<u>Step 2</u>:

Calculate the value of initial decision parameter P_0 as-
$P_0 = 1 – R$

<u>Step 3</u>:

Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}).
Find the next point of the first octant depending on the value of decision parameter P_k.
Follow the below two cases-



<u>Step 4</u>:

If the given centre point (X_0, Y_0) is not (0, 0), then do the following and plot the point-

$X_{plot} = X_c + X_0$
$Y_{plot} = Y_c + Y_0$
Here, (X_c, Y_c) denotes the current value of X and Y coordinates.

<u>Step 5</u>:

Keep repeating Step-03 and Step-04 until $X_{plot} >= Y_{plot}$.

<u>Step 6</u>:

Step 5 generates all the points for one octant.
To find the points for other seven octants, follow the eight-symmetry property of circle.

MIDPOINT ELLIPSE DRAWING ALGORITHM:-

This is an incremental method for scan converting an ellipse that is cantered at the origin in standard position i.e., with the major and minor axis parallel to coordinate system axis. It is very similar to the midpoint circle algorithm.
Because of the four-way symmetry property we need to consider the entire elliptical curve in the first quadrant.
Let's first rewrite the ellipse equation and define the function of that can be used to decide if the midpoint between two candidate pixels is inside or outside the ellipse:

Input (r_x,r_y) cantered at origin and the first point is $(x_0,y_0)=(0,r_y)$

Region 1:

$P1_0=r_y^2-r_x^2r_y+1/4r_x^2$
If $p1_k<0$ then (X_{k+1},Y_k) and
$p1_{k+1}=p1_k+2r_y^2x_{k+1}+r_y^2$
Else (X_{k+1},Y_{k-1})
$P1_{k+1}=p1_k+2r_y^2x_{k+1}-2r_x^2y_{k+1}+r_y^2$
And continue till $2r_y^2x>=2r_x^2y$

Region 2:

Here (x_0,y_0) is the last point calculated in region 1
$P2_0=r_y^2(x_0+1/2)^2+r_x^2(y_0-1)^2-r_x^2r_y^2$
If $p2_k>0$ then (x_k,y_{k-1}) and
$P2_{k+1}=p2_k-2r_x^2y_{k+1}+r_x^2$
Else (x_{k+1},y_{k-1})
$P2_{k+1}=p2_k+2r_y^2x_{k+1}-2r_x^2y_{k+1}+r_x^2$
Till y=0, i.e., last point is $(r_x,0)$

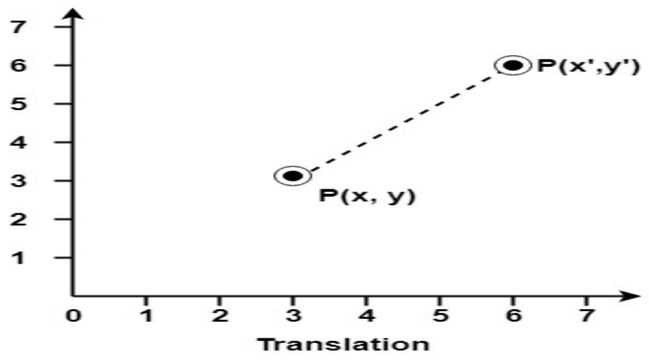
TRANSLATION:-

It is the straight-line movement of an object from one position to another is called Translation. Here the object is positioned from one coordinate location to another.
To translate a point from coordinate position (x, y) to another (x1 y1), we add algebraically the translation distances Tx and Ty to original coordinate.

$x1 = x + T_x$

$y1 = y + T_y$

The translation pair (T_x, Ty) is called as shift vector.
Translation is a movement of objects without deformation. Every position or point is translated by the same amount. When the straight line is translated, then it will be drawn using endpoints.
For translating polygon, each vertex of the polygon is converted to a new position. Similarly, curved objects are translated. To change the position of the circle or ellipse its centre coordinates are transformed, then the object is drawn using new coordinates.
Let P is a point with coordinates (x, y). It will be translated as (x1, y1).



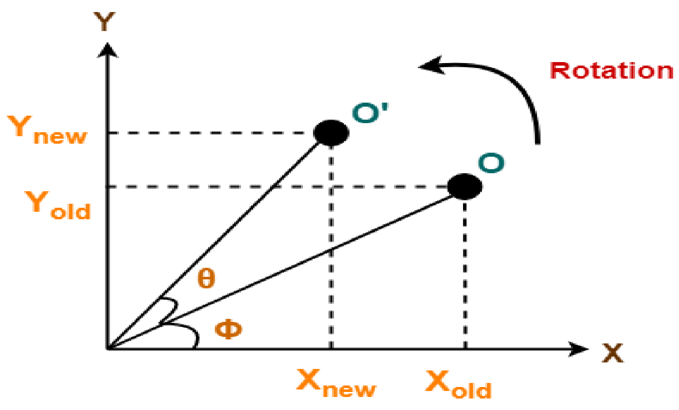
Matrix for Translation:

$$\begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix} \quad \text{Or} \quad \begin{vmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{vmatrix}$$

ROTATION:-

2D Rotation is a process of rotating an object with respect to an angle in a two-dimensional plane.

Consider a point object O has to be rotated from one angle to another in a 2D plane. Now let, (1) Initial coordinates of the object O = (X_{old}, Y_{old}) (2) Initial angle of the object O with respect to origin = Φ (3) Rotation angle = θ (4) New coordinates of the object O after rotation = (X_{new}, Y_{new})



2D Rotation in Computer Graphics

This rotation is achieved by using the following rotation equations-
$X_{new} = X_{old} \times \cos\theta - Y_{old} \times \sin\theta$, $Ynew = X_{old} \times \sin\theta + Y_{old} \times \cos\theta$

In Matrix form, the above rotation equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Rotation Matrix

For homogeneous coordinates, the above rotation matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Rotation Matrix

(Homogeneous Coordinates Representation)

Matrix for rotation:

Clockwise:

Matrix for homogeneous co-ordinate rotation

$$R = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Anti-clockwise:

Matrix for homogeneous co-ordinate rotation

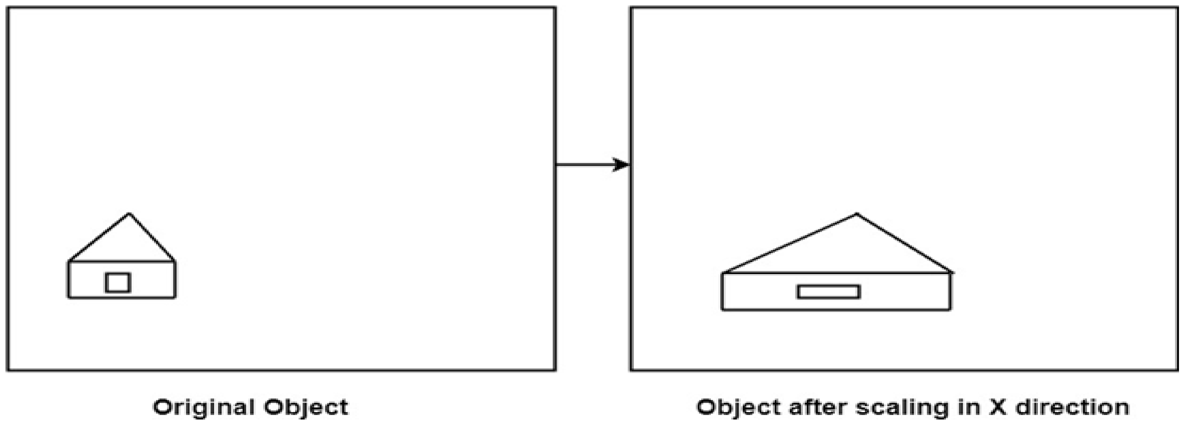
$$R = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

SCALING:-

It is used to alter or change the size of objects.
The change is done using scaling factors.
There are two scaling factors, i.e., Sx in x direction Sy in y-direction.
If the original position is x and y, scaling factors are Sx and Sy then the value of coordinates after scaling will be x1 and y1.
If the picture to be enlarged to twice its original size then Sx = Sy = 2.
If Sx and Sy are not equal then scaling will occur but it will elongate or distort the picture.
If scaling factors are less than one, then the size of the object will be reduced.
If scaling factors are higher than one, then the size of the object will be enlarged.
If S_x and S_y are equal it is also called as Uniform Scaling. If not equal then called as Differential Scaling. If scaling factors with values less than one will move the object closer to coordinate origin, while a value higher than one will move coordinate position farther from origin.

Enlargement: If $T_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$, If $(x_1\ y_1)$ is original position and T_1 is translation vector then $(x_2\ y_2)$ are coordinated after scaling

$$[x_2 y_2] = [\ x_1 y_1] \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = [2x_1 2y_1]$$



Matrix for scaling:

$$S = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

REFLECTION:-

It is a transformation which produces a mirror image of an object. The mirror image can be either about x-axis or y-axis. The object is rotated by180°.

1) Reflection about the x-axis

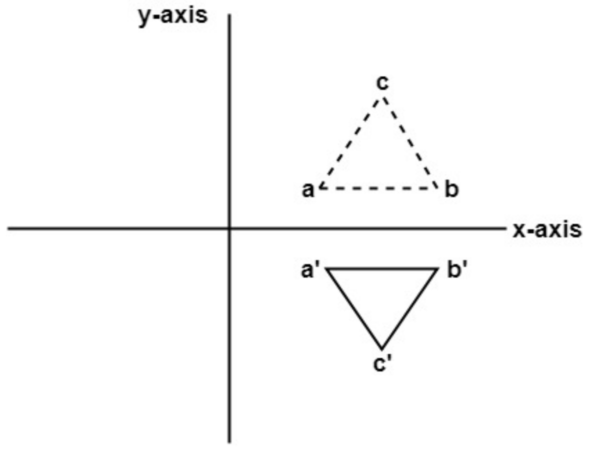
2) Reflection about the y-axis

3) Reflection about an axis perpendicular to xy plane and passing through the origin

4) Reflection about line y=x

1. Reflection about x-axis: The object can be reflected about x-axis with the help of the following matrix -

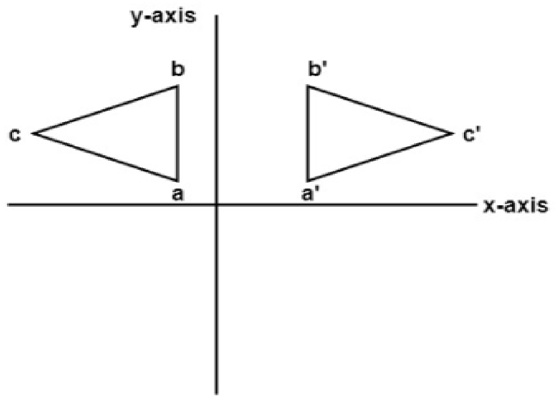
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



In this transformation value of x will remain same whereas the value of y will become negative. Following figures shows the reflection of the object axis. The object will lie another side of the x-axis.

2. Reflection about y-axis: The object can be reflected about y-axis with the help of following transformation matrix -

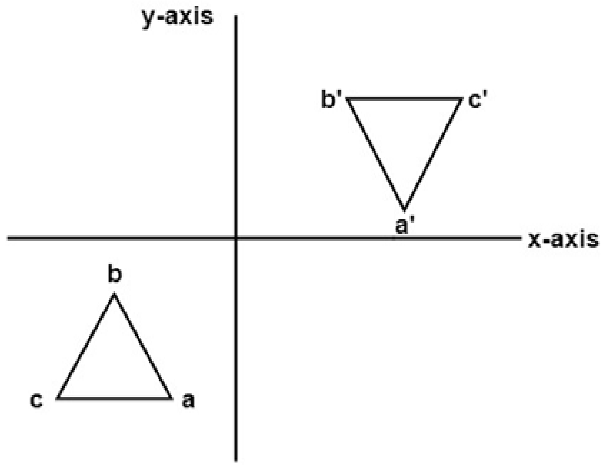
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Here the values of x will be reversed, whereas the value of y will remain the same. The object will lie another side of the y-axis. The above figure shows the reflection about the y-axis.

3. Reflection about an axis perpendicular to xy plane and passing through origin:
The matrix of this transformation is given below -

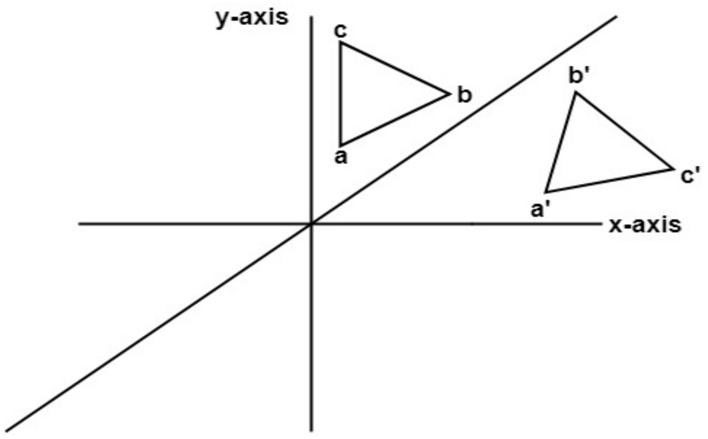
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



In this value of x and y both will be reversed. This is also called as half revolution about the origin.

4. Reflection about line y = x: The object may be reflected about line y = x with the help of following transformation matrix -

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



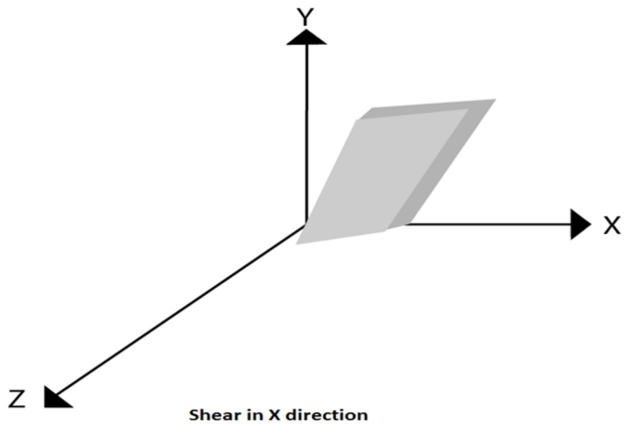
First of all, the object is rotated at 45°. The direction of rotation is clockwise. After its reflection is done concerning x-axis. The last step is the rotation of y=x back to its original position that is counterclockwise at 45°.

SHEARING:-

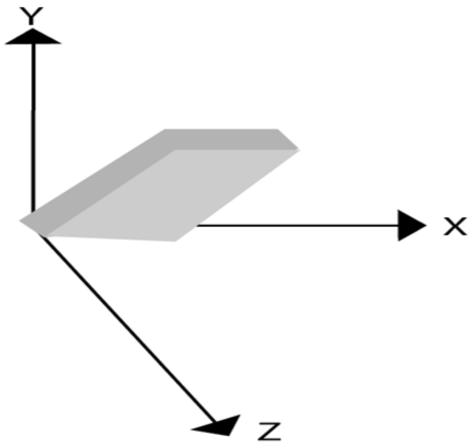
It is change in the shape of the object. It is also called as deformation. Change can be in the x-direction or y-direction or both directions in case of 2D. If shear occurs in both directions, the object will be distorted. But in 3D shear can occur in three directions.

Matrix for shear:

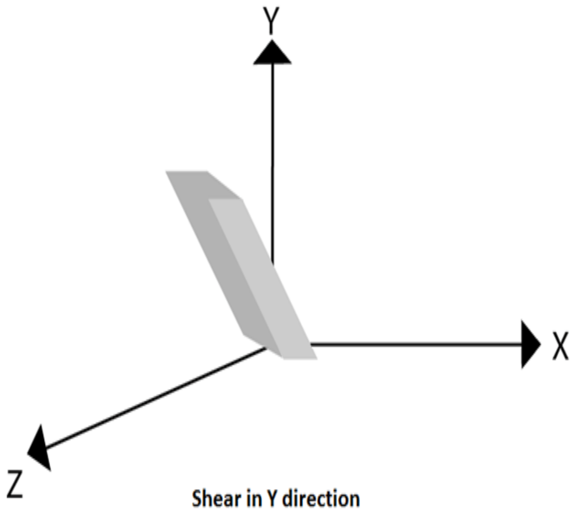
$$\begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Shear in X direction
(a)



Shear in X, Y direction
(b)



Shear in Y direction
(c)

SOURCE CODE:-

MATRIX MULTIPLICATION:

```
import numpy as np
def matrix_mul(matrix1,matrix2):
     new_matrix = [[0,0,0],[0,0,0],[0,0,0]]
     for i in range(len(matrix1)):
         for j in range(len(matrix2[0])):
     for k in range(len(matrix2)):
     new_matrix[i][j] += matrix1[i][k]*matrix2[k][j]
     return np.around(new_matrix,decimals=1)
```

BRESENHAM’S LINE DRAWING ALGORITHM:

```
def drawBresenham(x1, y1, x2, y2):
x_data = []
y_data = []
x_data.append(x1)
y_data.append(y1)
    dx = abs(x2 - x1)
dy = abs(y2 - y1)
    if (x2 > x1):
xs = 1
    else:
xs = -1
    if (y2 > y1):
ys = 1
    else:
ys = -1

    if (dx >dy):
        p1 = 2 * dy - dx
        while (x1 != x2):
            x1 += xs
            if (p1 >= 0):
                y1 += ys
                p1 -= 2 * dx
            p1 += 2 * dy
x_data.append(x1)
y_data.append(y1)

elif (dy>= dx):
        p1 = 2 * dx - dy
        while (y1 != y2):
            y1 += ys
            if (p1 >= 0):
                x1 += xs
                p1 -= 2 * dy
            p1 += 2 * dx
x_data.append(x1)
y_data.append(y1)

general_matrix = [[x_data],[y_data]]
    return general_matrix
```

MIDPOINT CIRCLE DRAWING ALGORITHM:

```
import numpy as np
def midpoint_circle(xc,yc,r):
     X=[0]
     Y=[r]
     x,y=0,r
     p_c=(5/4)-r
     while x<=y:
          if p_c<0:
               p_c=p_c+(2*x)+3
               x=x+1
               if(x>y):
                    break
               else:
                    X.append(x)
                    Y.append(y)
          else:
               p_c=p_c+(2*(x-y))+5
               y=y-1
               x=x+1
               if(x>y):
                    break
               else:
                    X.append(x)
                    Y.append(y)
     X_1=[]
     Y_1=[]
     X_rev = []
     Y_rev = []
     X_rev.extend(Y)
     X_neg = list(map(lambda x:-x, X))
     X_rev_neg = list(map(lambda x:-x,X_rev))
     Y_rev.extend(X)
     Y_neg = list(map(lambda x:-x, Y))
     Y_rev_neg = list(map(lambda x:-x,Y_rev))
     X_new = [X,X_rev[::-1],X_rev,X[::
1],X_neg,X_rev_neg[::-1],X_rev_neg,X_neg[::-1]]
     Y_new = [Y,Y_rev[::-1],Y_rev_neg,Y_neg[::-
1],Y_neg,Y_rev_neg[::-1],Y_rev,Y[::-1]]
     for i in X_new :
          X_1.extend(i)
     for i in Y_new :
          Y_1.extend(i)
     if xc==0 and yc==0 :
          general_matrix = [[X_1],[Y_1]]
          return general_matrix
     elif xc!=0 or yc!=0 :
          X1 = list(map(lambda x:x+xc,X_1))
          Y1 = list(map(lambda x:x+yc,Y_1))
          general_matrix = [[X1],[Y1]]
     return general_matrix
```

MIDPOINT ELLIPSE DRAWING ALGORITHM:

```
import numpy as np
import math

def midpoint_ellipse(xc,yc,rx_1,ry_1):
     rx=rx_1/2
     ry=ry_1/2
     X=[0]
     Y=[ry]
     x=0
     y=ry
     dx=2*pow(ry,2)*x
     dy=2*pow(rx,2)*y
     p1=pow(ry,2)-(pow(rx,2)*ry)+(pow(rx,2)/4)
     while dx<dy:
          if p1<0:
               x=x+1
               dx=2*x*pow(ry,2)
               p1=p1+dx+pow(ry,2)
               X.append(x)
               Y.append(y)
          else:
               x=x+1
               y=y-1
               dx=2*x*pow(ry,2)
               dy=2*y*pow(rx,2)
               p1=p1+dx+pow(ry,2)-dy
               X.append(x)
               Y.append(y)
     p2=(pow(ry,2)*pow((X[1]+0.5),2))+(pow(rx,2)*pow((Y[-
1]-1),2))-(pow(rx,2)*pow(ry,2))
     while(y>0):
          if p2>0:
               y=y-1
               dy=2*y*pow(rx,2)
```

```
            p2=p2-dy+pow(rx,2)
            if(y<0):
                break
            else:
                X.append(x)
                Y.append(y)
        else:
            x=x+1
            y=y-1
            dx=2*x*pow(ry,2)
            dy=2*y*pow(rx,2)
            p2=p2+dx-dy+pow(rx,2)
            if(y<0):
                break
            else:
                X.append(x)
                Y.append(y)
    X_1=[]
    Y_1=[]
    X_neg = list(map(lambda x:-x, X))
    Y_neg = list(map(lambda x:-x, Y))
    X_new = [X,X,X_neg,X_neg]
    Y_new = [Y,Y_neg,Y_neg,Y]
    for i in X_new :
        X_1.extend(i)
    for i in Y_new :
        Y_1.extend(i)
    if xc==0 and yc==0 :
        general_matrix = [[X_1],[Y_1]]
        return general_matrix
    elif xc!=0 or yc!=0 :
        X1 = list(map(lambda x:x+xc,X_1))
        Y1 = list(map(lambda x:x+yc,Y_1))
        general_matrix = [[X1],[Y1]]
        return general_matrix
```

TRANSLATION:

```
def translate_x(tx):
trans_matrix = [[1, 0, tx],[0, 1, 0],[0, 0, 1]]
    return trans_matrix

def translate_y(ty):
trans_matrix = [[1, 0, 0], [0, 1, ty], [0, 0, 1]]
    return trans_matrix

def translate_xy(tx, ty):
trans_matrix = [[1, 0, tx],[0, 1, ty],[0, 0, 1]]
    return trans_matrix
```

ROTATION:

```
import math
import numpy as np
import matrix_multiply as mm

def clockwise_rotation(m_c,m_t,m_rt):
     m1 = mm.matrix_mul(m_rt,m_c)
     m2 = mm.matrix_mul(m1,m_t)
     return m2

def anticlockwise_rotation(m_ac,m_t,m_rt):
     m1 = mm.matrix_mul(m_rt,m_ac)
     m2 = mm.matrix_mul(m1,m_t)
     return m2

def rotation(count,x1,y1,degree):
     a=math.sin(math.radians(degree))
     s=np.around(a,decimals=1)
     b=math.cos(math.radians(degree))
     c=np.around(b,decimals=1)
     m_c = ([c,s,0],[-s,c,0],[0,0,1])
     m_ac = ([c,-s,0],[s,c,0],[0,0,1])
     m_t = ([1,0,-x1],[0,1,-y1],[0,0,1])
     m_rt = ([1,0,x1],[0,1,y1],[0,0,1])
     if count==1 :
          mf=clockwise_rotation(m_c,m_t,m_rt)
     elif count==2 :
          mf=anticlockwise_rotation(m_ac,m_t,m_rt)
     return mf
```

SCALING:

```
def scale_x(Sx):
trans_matrix = [[Sx, 0, 0],[0, 1, 0],[0, 0, 1]]
    return trans_matrix

def scale_y(Sy):
trans_matrix = [[1, 0, 0], [0, Sy, 0], [0, 0, 1]]
    return trans_matrix

def scale_xy(Sx, Sy):
trans_matrix = [[Sx, 0, 0],[0, Sy, 0],[0, 0, 1]]
    return trans_matrix
```

REFLECTION:

```
import math
import numpy as np
import matrix_multiply as mm

def x_axis():
     m1 = ([1,0,0],[0,-1,0],[0,0,1])
     return m1
def y_axis():
     m1 = ([-1,0,0],[0,1,0],[0,0,1])
     return m1

def x_parallel(cst):
     if cst>0 :
          m1 = ([1,0,0],[0,1,cst],[0,0,1])
          m2 = ([1,0,0],[0,-1,0],[0,0,1])
          m3 = ([1,0,0],[0,1,-cst],[0,0,1])
          m4 = np.dot(m1,m2)
          m5 = np.dot(m4,m3)
     else :
          m1 = ([1,0,0],[0,1,-cst],[0,0,1])
          m2 = ([1,0,0],[0,-1,0],[0,0,1])
          m3 = ([1,0,0],[0,1,cst],[0,0,1])
          m4 = np.dot(m1,m2)
          m5 = np.dot(m4,m3)
     return m5

def y_parallel(cst):
     if cst>0 :
          m1 = ([1,0,cst],[0,1,0],[0,0,1])
          m2 = ([-1,0,0],[0,1,0],[0,0,1])
          m3 = ([1,0,-cst],[0,1,0],[0,0,1])
          m4 = np.dot(m1,m2)
          m5 = np.dot(m4,m3)
     else :
          m1 = ([1,0,-cst],[0,1,0],[0,0,1])
          m2 = ([-1,0,0],[0,1,0],[0,0,1])
          m3 = ([1,0,cst],[0,1,0],[0,0,1])
          m4 = np.dot(m1,m2)
          m5 = np.dot(m4,m3)
     return m5

def others(cst, m):
    degree = math.degrees(math.atan(m))
    a = math.sin(math.radians(degree))
    s = np.around(a, decimals=1)
    a = math.cos(math.radians(degree))
    c = np.around(a, decimals=1)
m_c = ([c, s, 0], [-s, c, 0], [0, 0, 1])
```

```
m_x = ([1, 0, 0], [0, -1, 0], [0, 0, 1])
m_ac = ([c, -s, 0], [s, c, 0], [0, 0, 1])
    m_t1 = ([1, 0, -cst], [0, 1, 0], [0, 0, 1])
    m_t2 = ([1, 0, cst], [0, 1, 0], [0, 0, 1])
    if cst == 0:
        m1 = mm.matrix_mul(m_ac, m_x)
        mf = mm.matrix_mul(m1, m_c)
    elif cst> 0:
        m1 = mm.matrix_mul(m_t2, m_ac)
        m2 = mm.matrix_mul(m1, m_x)
        m3 = mm.matrix_mul(m2, m_c)
        mf = mm.matrix_mul(m3, m_t1)
    else:
        m1 = mm.matrix_mul(m_t1, m_ac)
        m2 = mm.matrix_mul(m1, m_x)
        m3 = mm.matrix_mul(m2, m_c)
        mf = mm.matrix_mul(m3, m_t2)
    return mf
```

SHEARING:

```
def shear_x(shx):
trans_matrix = [[1, shx, 0],[0, 1, 0],[0, 0, 1]]
    return trans_matrix

def shear_y(shy):
trans_matrix = [[1, 0, 0], [shy, 1, 0], [0, 0, 1]]
    return trans_matrix

def shear_xy(shx, shy):
trans_matrix = [[1, shx, 0],[shy, 1, 0],[0, 0, 1]]
    return trans_matrix
```

MAIN FILE:

```
import matrix_multiply as mm
import Bresenham1 as bresen
import mcda as midcirc
import meda as midellip
import translate as tr
import rotation as rot
import scaling as sc
import reflection as ref
import shearing as sh
from matplotlib import pyplot as plt

x_list, y_list = [], []
shape = int(input("\nEnter the number of shapes you want
to create: "))
while(shape != 0):
ch = int(input("\nEnter 1 for drawing a line\nEnter 2 for
drawing a circle\nEnter 3 for drawing an ellipse\n\nINPUT
: "))
if(ch == 1):
        line = int(input("\nEnter the number of lines you
want to draw: "))
while(line != 0):
            x1 = float(input("\nEnter x coordinate of
starting point: "))
            y1 = float(input("Enter y coordinate of
starting point: "))
            x2 = float(input("Enter x coordinate of
ending point: "))
            y2 = float(input("Enter y coordinate of
ending point: "))
            returned_list1 = bresen.drawBresenham(x1, y1,
x2, y2)
            for i in returned_list1[0]:
              x_list.extend(i)
            for i in returned_list1[1]:
              y_list.extend(i)
            line = line - 1
        shape = shape - 1
elif(ch == 2):
        circ = int(input("\nEnter the number of circles
you want to draw: "))
        temp = circ
        while (circ != 0):
            cx = float(input("\nEnter x coordinate of
centre point: "))
            cy = float(input("Enter y coordinate of
centre point: "))
            r = float(input("Enter radius: "))
            returned_list2 = midcirc.midpoint_circle(cx,
cy, r)
            for i in returned_list2[0]:
              x_list.extend(i)
            for i in returned_list2[1]:
              y_list.extend(i)
            circ = circ - 1
        shape = shape - temp
elif(ch == 3):
elip = int(input("\nEnter the number of ellipse you want
to draw: "))
        temp = elip
        while (elip != 0):
          xe = float(input("\nEnter x center coordinate :
"))
          ye = float(input("Enter y center coordinate :
"))
            rxe_1 = float(input("Enter major axis of the
ellipse : "))
            rye_1 = float(input("Enter minor axis of the
ellipse : "))
```

```
            returned_list3 =
midellip.midpoint_ellipse(xe, ye, rxe_1, rye_1)
            for i in returned_list3[0]:
x_list.extend(i)
            for i in returned_list3[1]:
y_list.extend(i)
elip = elip - 1
        shape = shape - temp
    else:
print("\nWrong choice. Please enter correct details.")

plt.rc('grid', linestyle="-", color='black')
plt.scatter(x_list, y_list, c='r')
plt.xlabel(" X axis --->")
plt.ylabel(" Y axis --->")
plt.title('Plotting Shapes For Transformations')
plt.grid()
plt.show()




new_matrix, old_matrix = [], []
temp_list = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
nt = int(input("\nEnter the number of transformations:
"))
while(nt != 0):
    choice = int(input("\nEnter 1 for translation\nEnter
2 for rotation\nEnter 3 for scaling\nEnter 4 for
reflection\nEnter 5 for shearing\n\nINPUT : "))
if(choice == 1):
        c1 = int(input("\nEnter 1 for translating along
x-axis\nEnter 2 for translating along y-axis\nEnter 3 for
translating along both axes\n\nINPUT : "))
        if (c1 == 1):
tx = float(input("\nEnter the translating factor along x-
axis: "))
```

```
temp_list = mm.matrix_mul(tr.translate_x(tx), temp_list)
elif (c1 == 2):
            ty = float(input("\nEnter the translating
factor along y-axis: "))
temp_list = mm.matrix_mul(tr.translate_y(ty), temp_list)
elif (c1 == 3):
tx = float(input("\nEnter the translating factor along x-
axis: "))
            ty = float(input("Enter the translating
factor along y-axis: "))
temp_list = mm.matrix_mul(tr.translate_xy(tx, ty),
temp_list)
        else:
print("\nWrong choice. Please enter correct details.")
nt = nt - 1

elif(choice == 2):
        c2 = int(input("\nEnter 1 for clockwise
rotation\nEnter 2 for anticlockwise rotation\n\nINPUT :
"))
        if (c2 == 1):
            theta = float(input("\nEnter the angle of
clockwise rotation: "))
x_fixed = float(input("Enter the abscissa of the pivot
point(Enter 0 in case of origin): "))
y_fixed = float(input("Enter the ordinate of the pivot
point(Enter 0 in case of origin): "))
temp_list = mm.matrix_mul(rot.rotation(c2, x_fixed,
y_fixed, theta), temp_list)
elif (c2 == 2):
            theta = float(input("\nEnter the angle of
anticlockwise rotation: "))
x_fixed = float(input("Enter the abscissa of the pivot
point(Enter 0 in case of origin): "))
y_fixed = float(input("Enter the ordinate of the pivot
point(Enter 0 in case of origin): "))
```

```
temp_list = mm.matrix_mul(rot.rotation(c2, x_fixed,
y_fixed, theta), temp_list)
        else:
print("\nWrong choice. Please enter correct details.")
nt = nt - 1

elif(choice == 3):
        c3 = int(input("\nEnter 1 for scaling along x-
axis\nEnter 2 for scaling along y-axis\nEnter 3 for
scaling along both axes\n\nINPUT : "))
        if (c3 == 1):
            Sx = float(input("\nEnter the scaling factor
along x-axis: "))
temp_list = mm.matrix_mul(sc.scale_x(Sx), temp_list)
elif (c3 == 2):
            Sy = float(input("\nEnter the scaling factor
along y-axis: "))
temp_list = mm.matrix_mul(sc.scale_y(Sy), temp_list)
elif (c3 == 3):
            Sx = float(input("\nEnter the scaling factor
along x-axis: "))
            Sy = float(input("Enter the scaling factor
along y-axis: "))
temp_list = mm.matrix_mul(sc.scale_xy(Sx, Sy), temp_list)
        else:
print("\nWrong choice. Please enter correct details.")
nt = nt - 1

elif (choice == 4):
        c4 = int(input("Enter 1 for reflection about x-
axis\nEnter 2 for reflection about y-axis\nEnter 3 for
reflection about a line parallel to x-axis\nEnter 4 for
reflection about a line parallel to y-axis\nEnter 5 for
reflection about any other line\n\nINPUT : "))
        if (c4 == 1):
temp_list = mm.matrix_mul(ref.x_axis(), temp_list)
```

```
elif (c4 == 2):
temp_list = mm.matrix_mul(ref.y_axis(), temp_list)
elif (c4 == 3):
            constant = float(input("\nEnter the value of
constant (as in y = constant): "))
temp_list = mm.matrix_mul(ref.x_parallel(constant),
temp_list)
elif (c4 == 4):
            constant = float(input("\nEnter the value of
constant (as in x = constant): "))
temp_list = mm.matrix_mul(ref.y_parallel(constant),
temp_list)
elif (c4 == 5):
            slope = float(input("\nEnter the value of
slope(m) [as in y = mx + c]: "))
            constant = float(input("Enter the value of
constant(c) [as in y = mx + c]: "))
temp_list = mm.matrix_mul(ref.others(constant, slope),
temp_list)
        else:
print("\nWrong choice. Please enter correct details.")
nt = nt - 1

elif(choice == 5):
        c5 = int(input("Enter 1 for shearing along x-
axis\nEnter 2 for shearing along y-axis\nEnter 3 for
shearing along both axes\n\nINPUT : "))
        if (c5 == 1):
shx = float(input("\nEnter the shearing factor along x-
axis: "))
temp_list = mm.matrix_mul(sh.shear_x(shx), temp_list)
elif (c5 == 2):
            shy = float(input("\nEnter the shearing
factor along y-axis: "))
temp_list = mm.matrix_mul(sh.shear_y(shy), temp_list)
elif (c5 == 3):
```

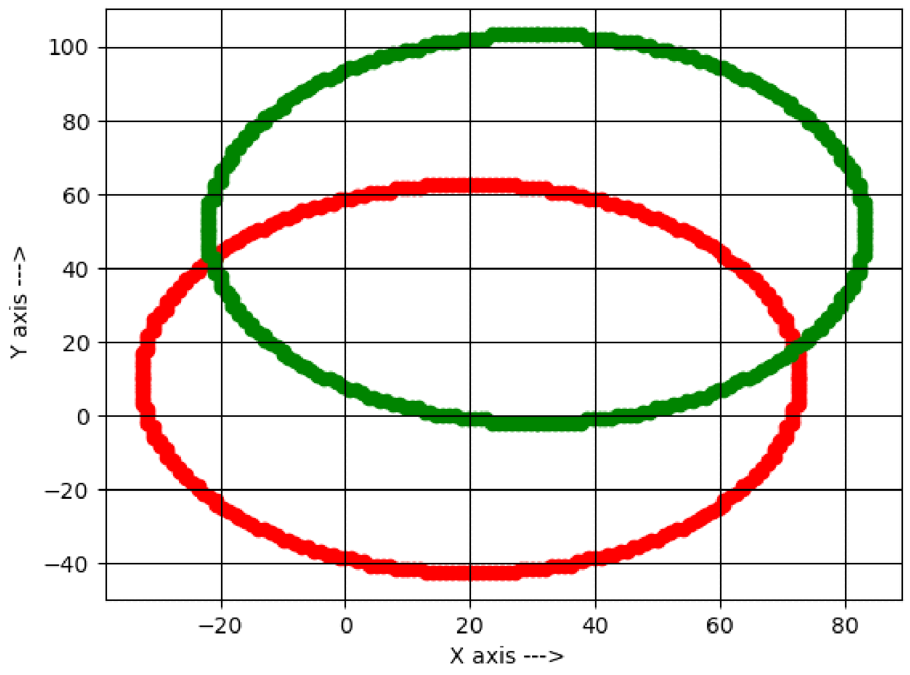
```
shx = float(input("\nEnter the shearing factor along x-
axis: "))
            shy = float(input("Enter the shearing factor
along y-axis: "))
temp_list = mm.matrix_mul(sh.shear_xy(shx, shy),
temp_list)
        else:
print("\nWrong choice. Please enter correct details.")
nt = nt - 1
    else:
print("\nWrong choice. Please enter correct details.")

x_list_new,y_list_new = [], []
#print("The corresponding coordinates obtained after
transformations are: ")
for i in range(len(x_list)):
old_matrix = [[x_list[i]], [y_list[i]], [1]]
new_matrix = mm.matrix_mul(temp_list, old_matrix)
x_list_new.append(new_matrix[0][0])
y_list_new.append(new_matrix[1][0])
    #print("x = %s, y = %s" % (new_matrix[0][0],
new_matrix[1][0]))

plt.rc('grid', linestyle="-", color='black')
plt.scatter(x_list,y_list, c='r')
plt.scatter(x_list_new,y_list_new, c='green')
plt.xlabel(" X axis --->")
plt.ylabel(" Y axis --->")
plt.grid()
plt.show()
```

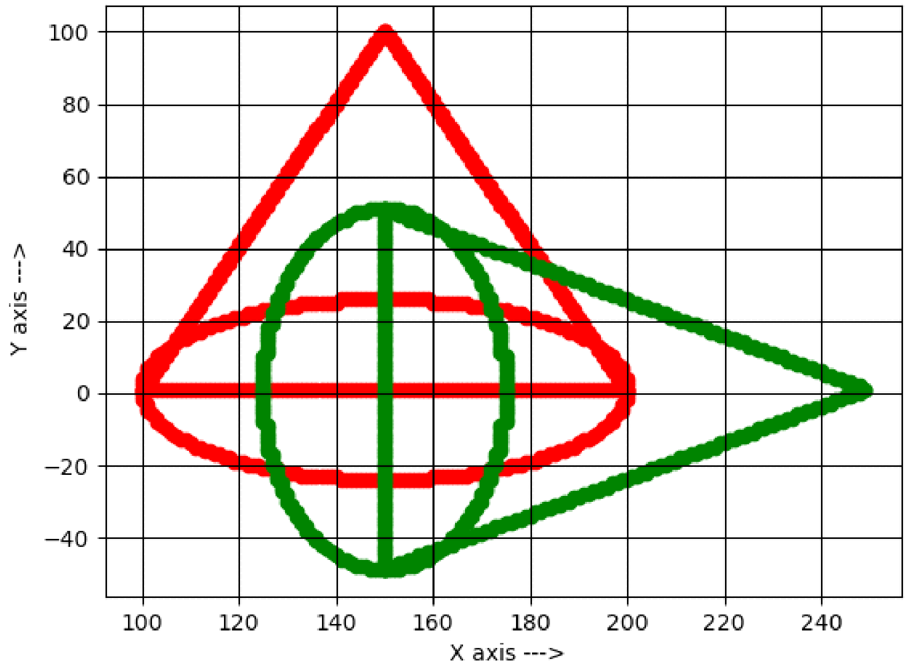
OUTPUTS:

OUTPUT-1



Creating a circle having center (20, 10) and with a radius = 52.6 and then translating ((Translating Factor) $_{x=}$10.6; (Translating Factor) $_{y=}$40.7).

OUTPUT-2



Creating two shapes: -

TRIANGLE-Line 1((x=100; y=1), (x=200; y=1)),

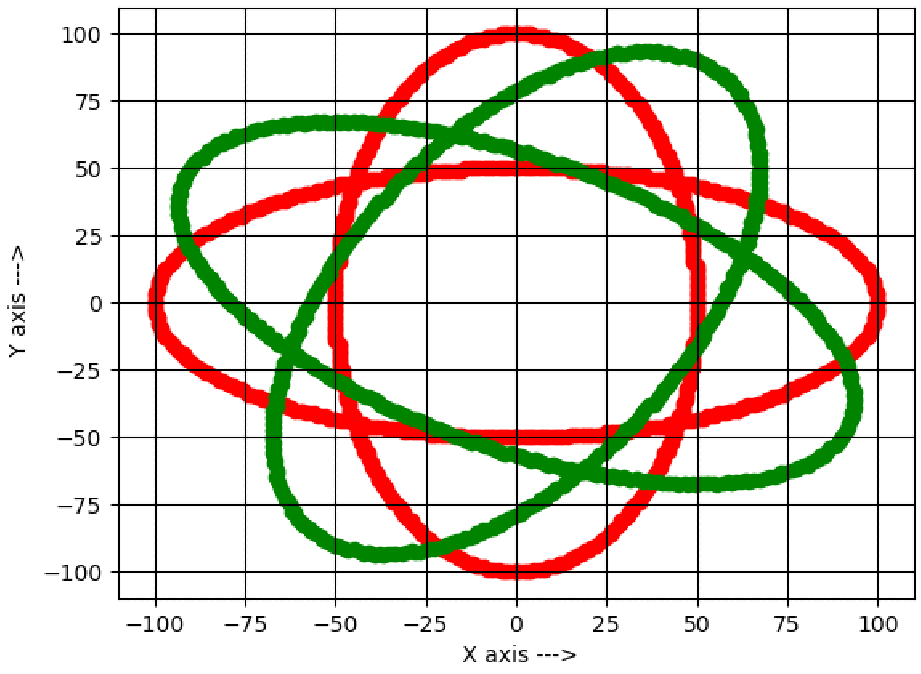
Line 2((x=200; y=1), (x=150; y=100)),

Lines 3((x=100; y=1), (x=150;100))

ELLIPSE (x=150; y=1)-Major axis=100,Minor axis=50 and,

Rotating them upto 90degrees in clockwise direction about the pivot point= (150,1)

OUTPUT-3



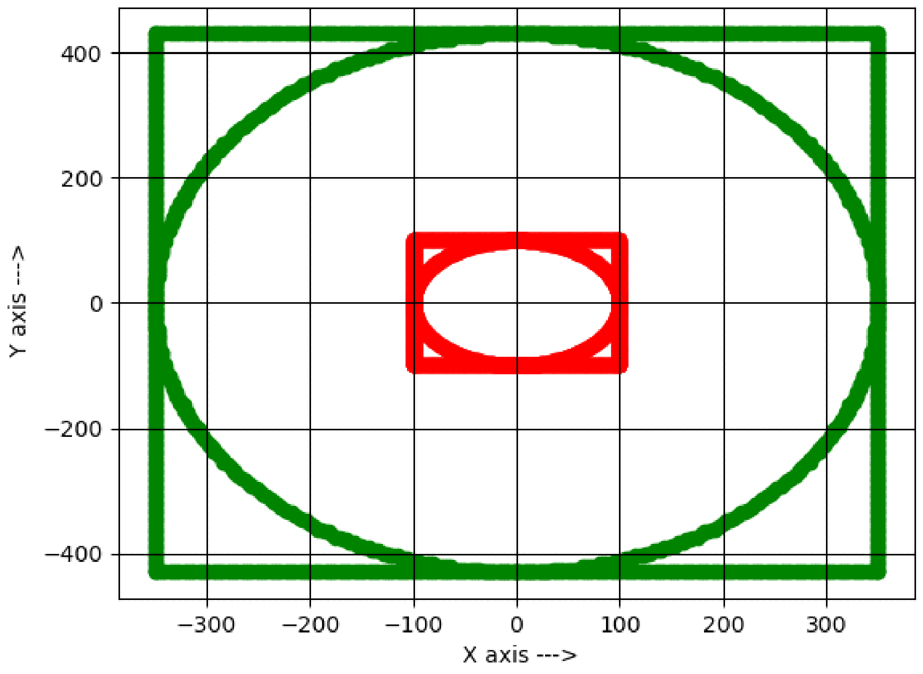
Creating two ellipse both centered at the origin (0,0)

ELLIPSE 1(Major axis=200 and Minor axis=100)

ELLIPSE 2(Major axis=100 and Minor axis=200) and

Rotating them upto 60 $^{\text{degrees}}$ in anti-clockwise direction about the pivot point= (0,0)

OUTPUT-4



Creating two shapes: -

SQUARE-Line 1((x=-100; y=-100), (x=100; y=-100)),

Line 2((x=100; y=-100), (x=100; y=100)),

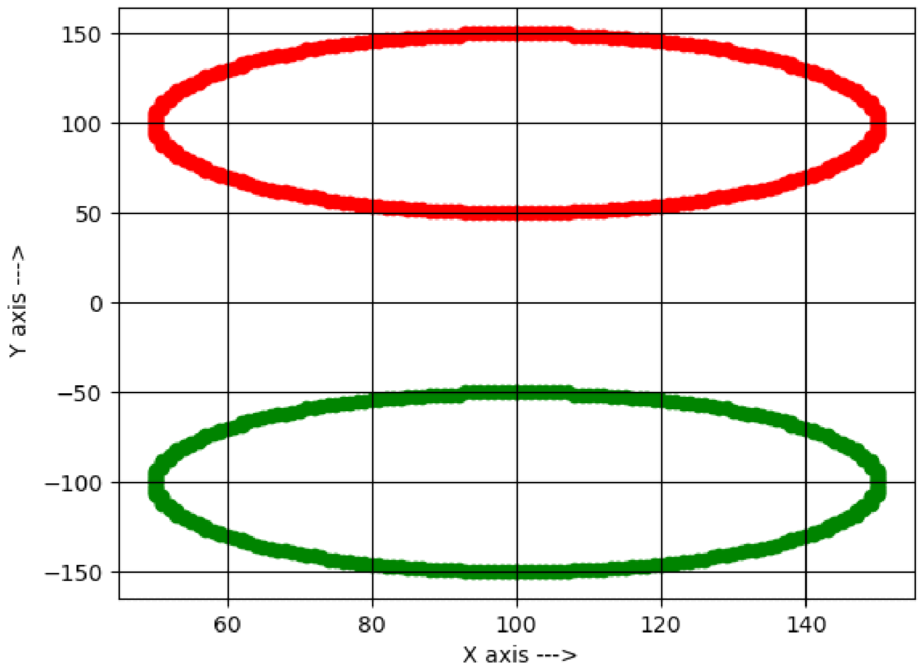
Line 3((x=-100; y=100), (x=100; y=100)),

Line 4((x=-100; y=100), (x=-100; y=-100)) and

CIRCLES centered at origin (0,0) with radius=100.

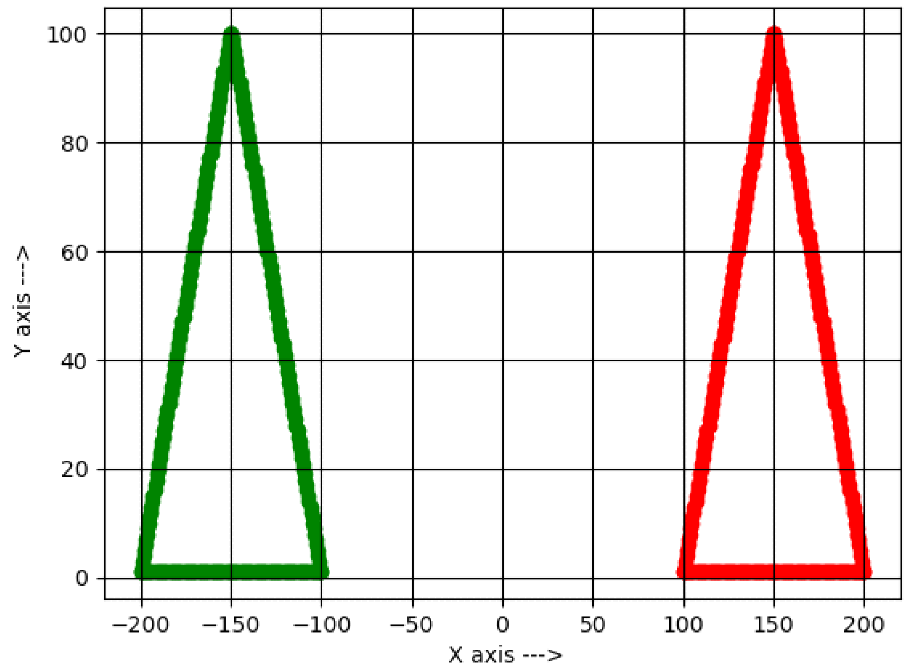
Now, on Scaling them along both axes ((Scaling Factor) $_{x=}$3.5, (Scaling Factor) $_{y}$=4.3)) we are getting the above output.

OUTPUT-5



Creating a CIRCLE centered at (100,100) with radius=50 then reflecting it about x-axis.

OUTPUT-6



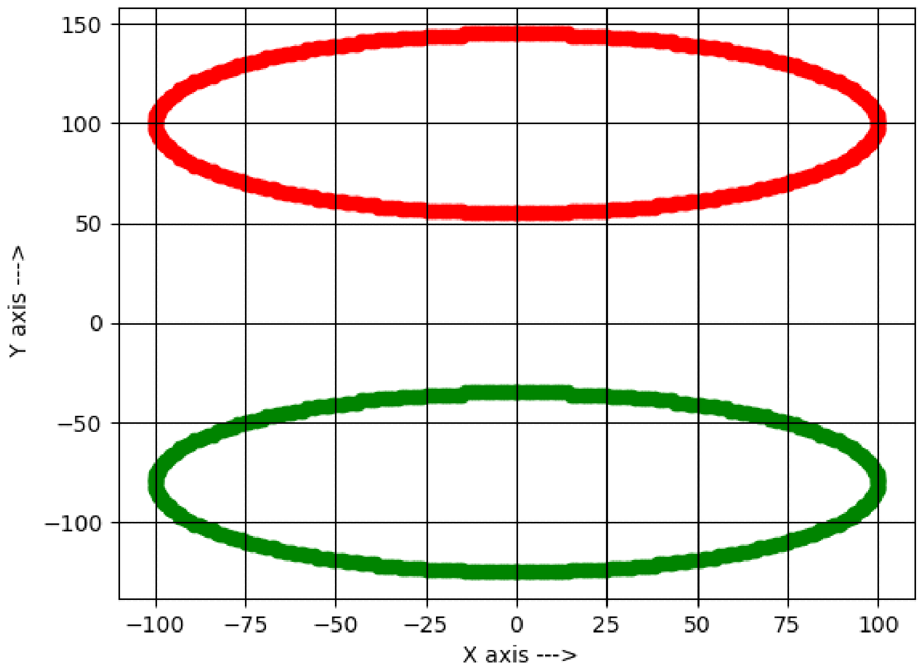
Creating a TRIANGLE- [Line 1((x=100; y=1), (x=200; y=1)),

Line 2((x=200; y=1), (x=150; y=100)),

Line 3((x=150; y=100), (x=100; y=1))]

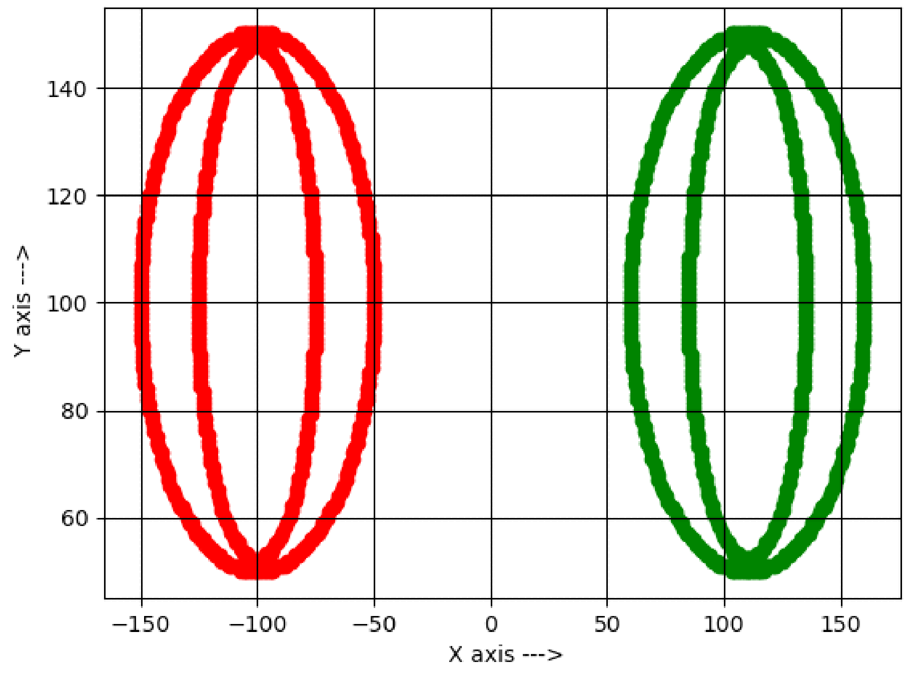
and reflecting it along y-axis.

OUTPUT-7



Creating an ELLIPSE centered at (0,100) with Major axis=200; Minor axis=90 and reflecting it about y=-10

OUTPUT-8



Creating two shapes: -

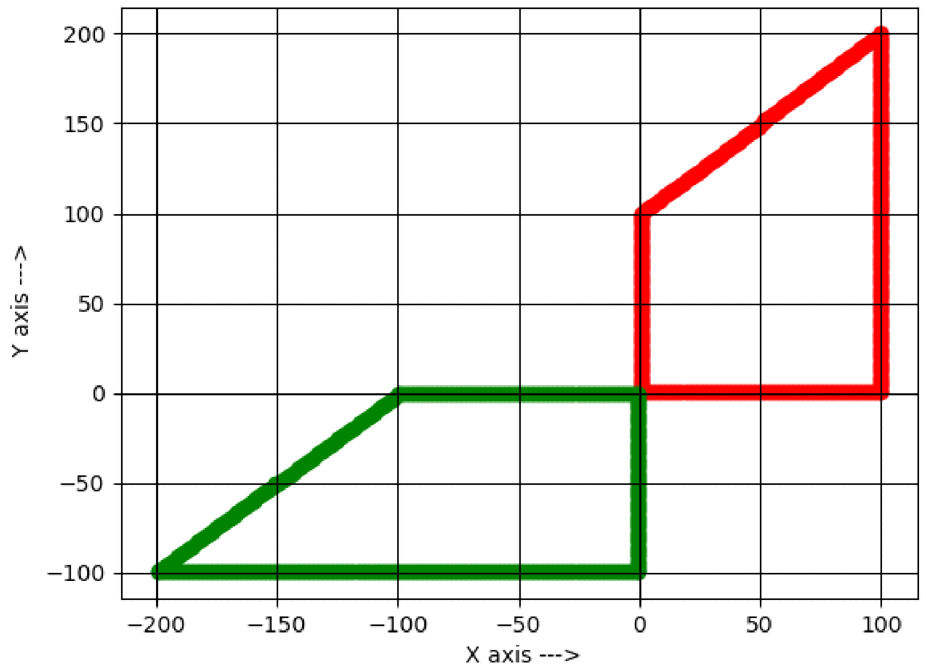
CIRCLE centered at (-100,100) with radius=50 and

ELLIPSE centered at (-100,100) with Major axis=50;

Minor axis=95.3 and

Reflecting it about a line x=5

OUTPUT-9



Creating a shape with 4 sides-

[Line 1((x=1; y=1), (x=100; y=1)),

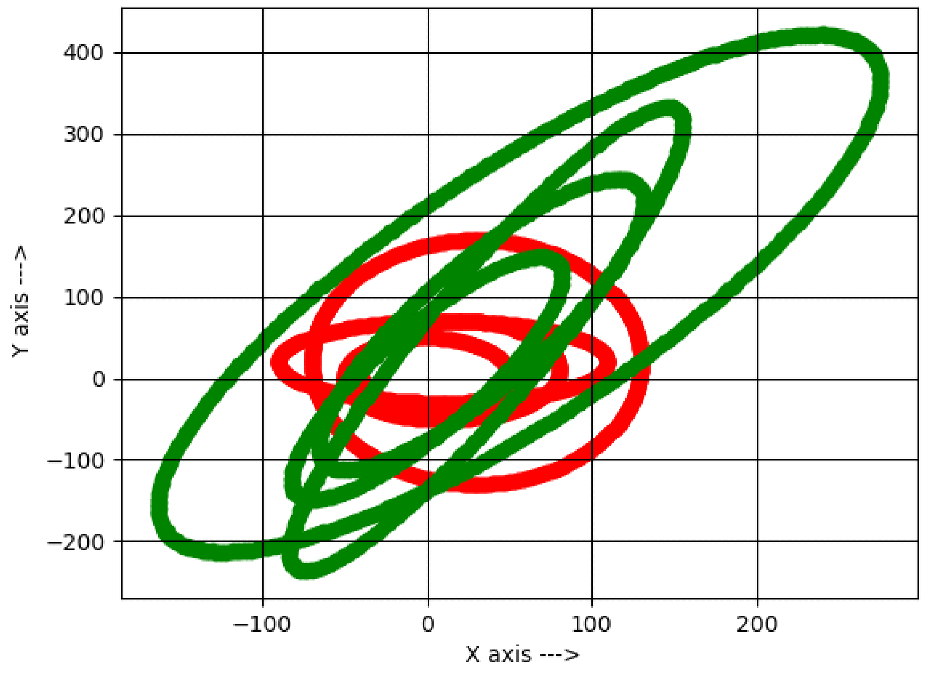
Line 2((x=100; y=1), (x=100; y=200)),

Line 3((x=1; y=100), (x=100; y=200)),

Line 4((x=1; y=1), (x=1; y=100))] and

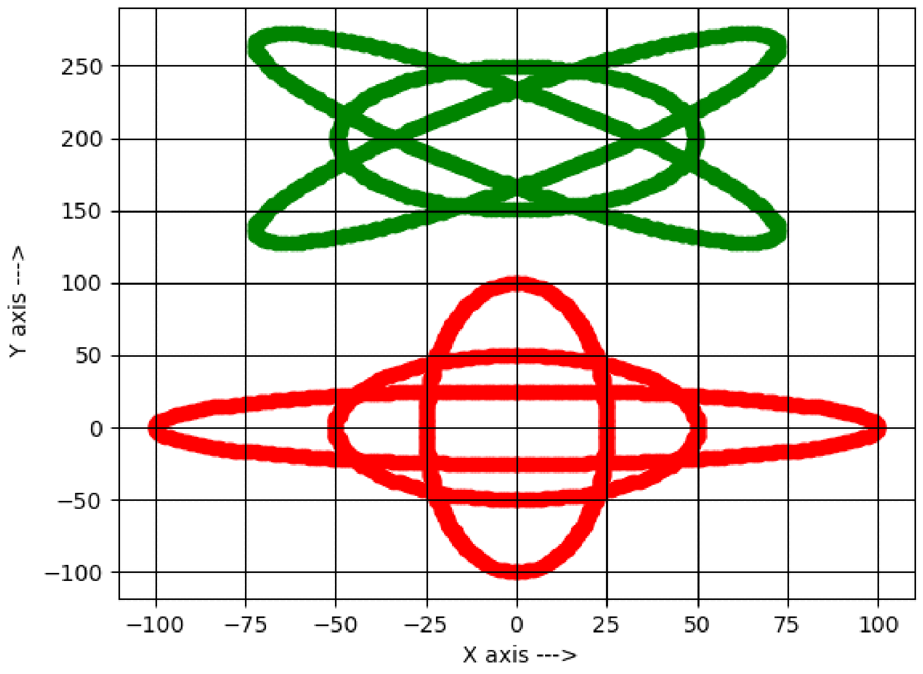
Reflecting it about a Line y=-x+0.5

OUTPUT-10



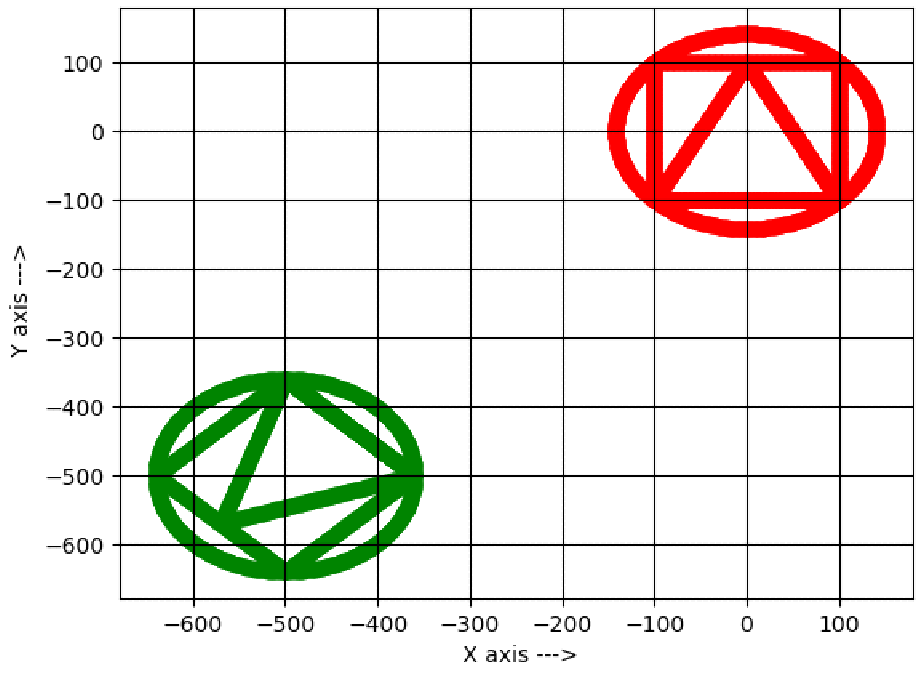
Creating 4 shapes: -
CIRCLE 1 centered at (0,0) with radius=50,
CIRCLE 2 centered at (20,10) with radius=60,
ELLIPSE 1 centered at (10, 20) with Major axis=200; Minor axis=100,
ELLIPSE 2 centered at (30,20) with Major axis=200; Minor axis=300
and
Shearing them along both axes ((Shearing Factor) $_x$=1.3; (Shearing Factor) $_y$=2.8).

OUTPUT-11



Creating three shapes: -
CIRCLE centered at (0, 0) with radius=50,
ELLIPSE 1 centered at (0, 0) with Major axis=200;
Minor axis-50,
ELLIPSE 2 centered at (0, 0) with Major axis=50;
Minor axis=200, then
Translating ((TranslatingFactor)$_y$=200) and
Rotating them upto 45 degrees about the pivot point= (0, 200)

OUTPUT-12



Creating 3 shapes: -

SQUARE- [Line 1((x=-100; y=-100), (x=100; y=-100)),

Line 2((x=100; y=-100), (x=100; y=100)),

Line 3((x=-100; y=100), (x=100; y=100)),

Line 4((x=-100; y=-100), (x=-100; y=100))]

TRIANGLE- [Line 1((x=-100; y=-100), (x=100; y=-100)),

Line 2((x=100; y=-100), (x=-0; y=100)),

Line 3((x=-100; y=-100), (x=-0; y=100))]

CIRCLE centered at (0,0) with radius=141.4, then

Translating along both axes ((Translating Factor) $_x$=500;(Translating Factor) $_y$=500),

Reflecting about a line y=-x, and, Rotating them upto an angle of 45 degrees about the pivot point= (-500, -500)

<u>**CONCLUSION:-**</u>

“Transformations are the operations applied to geometrical description of an object to change its position, orientation, or size and are called geometric transformations”.

To manipulate the initially created object and to display the modified object without having to redraw it, we use transformations.

Bibliography:

References:

1) Hearn, Baker – “Computer Graphics (C version 2nd Ed.)” – Pearson education
2) https://www.javatpoint.com/computer-graphics-programs
3) https://www.gatevidyalay.com/computer-graphics/

GitHub link:

https://github.com/subhankarghosh2000/2-D-Transformation