Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

LAB-

[Course : COMP 342]

Submitted by

Sudip Bhattarai

Roll no:11

Submitted to

Mr.Dhiraj Shrestha

Department of Computer Science and Engineering

May 28, 2024,

2D Transformation

The below shown matrixes represent the transformation matrix for each case and its corresponding coordinates calculated in a homogeneous coordinate system.

1. Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2. Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3. Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

4. Reflection

- Along X-Axis (Y=0 Line)

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

- Along Y-Axis (X=0 Line)

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

- Along X=Y Line

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

- Along Origin

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

5. Shearing

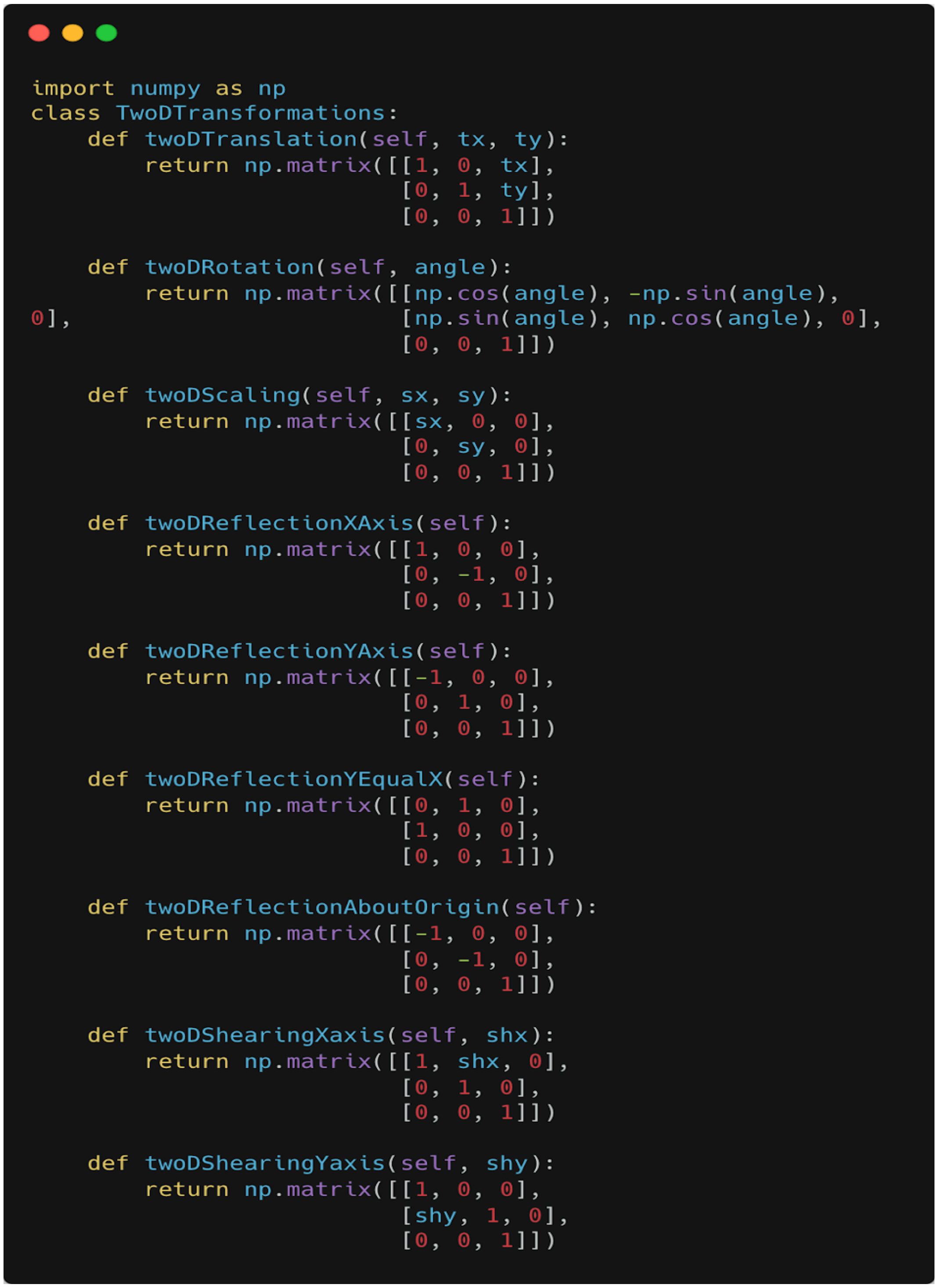
- Shearing Along X-axis

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

- Shearing Along Y-aixs

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

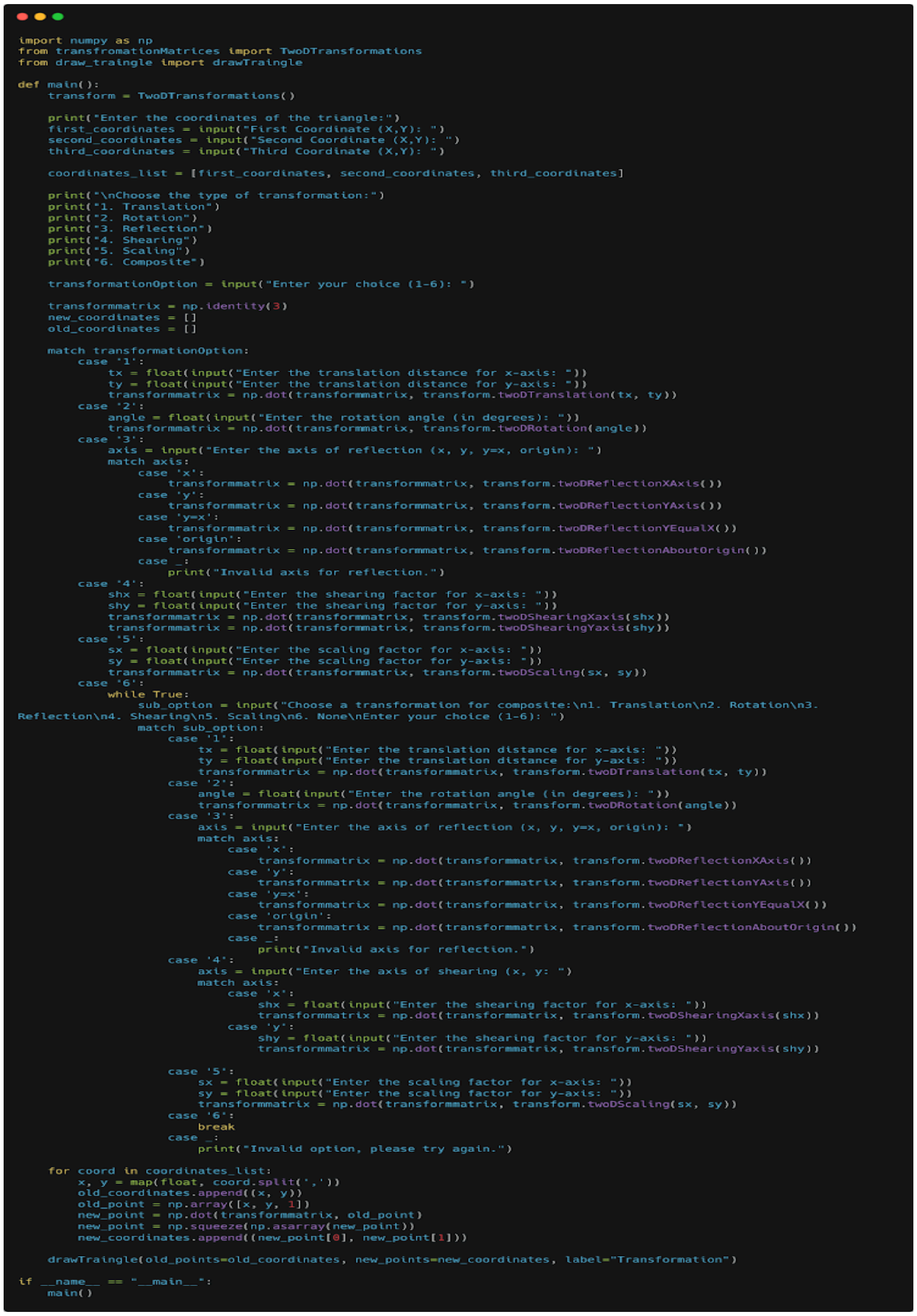
Now, to implement this transformation matrix, we defined a class and methods to return the transformation matrix depending on the user's choice. The code implementation can be shown below:



```
import numpy as np
class TwoDTransformations:
    def twoDTranslation(self, tx, ty):
        return np.matrix([[1, 0, tx],
                          [0, 1, ty],
                          [0, 0, 1]])

    def twoDRotation(self, angle):
        return np.matrix([[np.cos(angle), -np.sin(angle),
0],                       [np.sin(angle), np.cos(angle), 0],
                          [0, 0, 1]])

    def twoDScaling(self, sx, sy):
        return np.matrix([[sx, 0, 0],
                          [0, sy, 0],
                          [0, 0, 1]])

    def twoDReflectionXAxis(self):
        return np.matrix([[1, 0, 0],
                          [0, -1, 0],
                          [0, 0, 1]])

    def twoDReflectionYAxis(self):
        return np.matrix([[-1, 0, 0],
                          [0, 1, 0],
                          [0, 0, 1]])

    def twoDReflectionYEqualX(self):
        return np.matrix([[0, 1, 0],
                          [1, 0, 0],
                          [0, 0, 1]])

    def twoDReflectionAboutOrigin(self):
        return np.matrix([[-1, 0, 0],
                          [0, -1, 0],
                          [0, 0, 1]])

    def twoDShearingXaxis(self, shx):
        return np.matrix([[1, shx, 0],
                          [0, 1, 0],
                          [0, 0, 1]])

    def twoDShearingYaxis(self, shy):
        return np.matrix([[1, 0, 0],
                          [shy, 1, 0],
                          [0, 0, 1]])
```

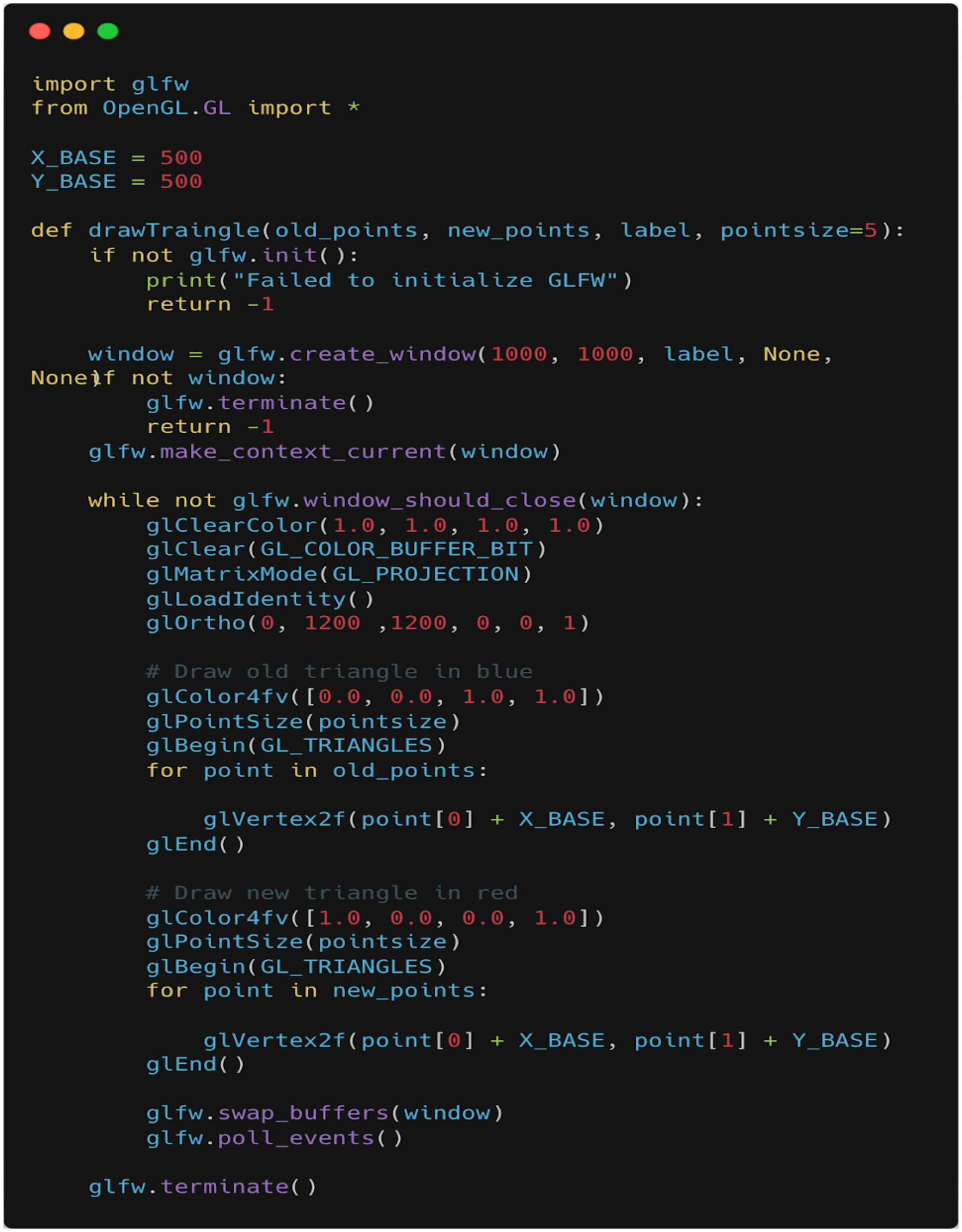
Implementation

To implement above mentioned transformation as well as the composite transformation, we will be taking an option from the user along with the coordinates to plot the Python implementation, which is shown below:



```python
import numpy as np
from transfromationMatrices import TwoDTransformations
from draw_traingle import drawTraingle

def main():
    transform = TwoDTransformations()

    print("Enter the coordinates of the triangle:")
    first_coordinates = input("First Coordinate (X,Y): ")
    second_coordinates = input("Second Coordinate (X,Y): ")
    third_coordinates = input("Third Coordinate (X,Y): ")

    coordinates_list = [first_coordinates, second_coordinates, third_coordinates]

    print("\nChoose the type of transformation:")
    print("1. Translation")
    print("2. Rotation")
    print("3. Reflection")
    print("4. Shearing")
    print("5. Scaling")
    print("6. Composite")

    transformationOption = input("Enter your choice (1-6): ")

    transformmatrix = np.identity(3)
    new_coordinates = []
    old_coordinates = []

    match transformationOption:
        case '1':
            tx = float(input("Enter the translation distance for x-axis: "))
            ty = float(input("Enter the translation distance for y-axis: "))
            transformmatrix = np.dot(transformmatrix, transform.twoDTranslation(tx, ty))
        case '2':
            angle = float(input("Enter the rotation angle (in degrees): "))
            transformmatrix = np.dot(transformmatrix, transform.twoDRotation(angle))
        case '3':
            axis = input("Enter the axis of reflection (x, y, y=x, origin): ")
            match axis:
                case 'x':
                    transformmatrix = np.dot(transformmatrix, transform.twoDReflectionXAxis())
                case 'y':
                    transformmatrix = np.dot(transformmatrix, transform.twoDReflectionYAxis())
                case 'y=x':
                    transformmatrix = np.dot(transformmatrix, transform.twoDReflectionYEqualX())
                case 'origin':
                    transformmatrix = np.dot(transformmatrix, transform.twoDReflectionAboutOrigin())
                case _:
                    print("Invalid axis for reflection.")
        case '4':
            shx = float(input("Enter the shearing factor for x-axis: "))
            shy = float(input("Enter the shearing factor for y-axis: "))
            transformmatrix = np.dot(transformmatrix, transform.twoDShearingXaxis(shx))
            transformmatrix = np.dot(transformmatrix, transform.twoDShearingYaxis(shy))
        case '5':
            sx = float(input("Enter the scaling factor for x-axis: "))
            sy = float(input("Enter the scaling factor for y-axis: "))
            transformmatrix = np.dot(transformmatrix, transform.twoDScaling(sx, sy))
        case '6':
            while True:
                sub_option = input("Choose a transformation for composite:\n1. Translation\n2. Rotation\n3.
Reflection\n4. Shearing\n5. Scaling\n6. None\nEnter your choice (1-6): ")
                match sub_option:
                    case '1':
                        tx = float(input("Enter the translation distance for x-axis: "))
                        ty = float(input("Enter the translation distance for y-axis: "))
                        transformmatrix = np.dot(transformmatrix, transform.twoDTranslation(tx, ty))
                    case '2':
                        angle = float(input("Enter the rotation angle (in degrees): "))
                        transformmatrix = np.dot(transformmatrix, transform.twoDRotation(angle))
                    case '3':
                        axis = input("Enter the axis of reflection (x, y, y=x, origin): ")
                        match axis:
                            case 'x':
                                transformmatrix = np.dot(transformmatrix, transform.twoDReflectionXAxis())
                            case 'y':
                                transformmatrix = np.dot(transformmatrix, transform.twoDReflectionYAxis())
                            case 'y=x':
                                transformmatrix = np.dot(transformmatrix, transform.twoDReflectionYEqualX())
                            case 'origin':
                                transformmatrix = np.dot(transformmatrix, transform.twoDReflectionAboutOrigin())
                            case _:
                                print("Invalid axis for reflection.")
                    case '4':
                        axis = input("Enter the axis of shearing (x, y: ")
                        match axis:
                            case 'x':
                                shx = float(input("Enter the shearing factor for x-axis: "))
                                transformmatrix = np.dot(transformmatrix, transform.twoDShearingXaxis(shx))
                            case 'y':
                                shy = float(input("Enter the shearing factor for y-axis: "))
                                transformmatrix = np.dot(transformmatrix, transform.twoDShearingYaxis(shy))

                    case '5':
                        sx = float(input("Enter the scaling factor for x-axis: "))
                        sy = float(input("Enter the scaling factor for y-axis: "))
                        transformmatrix = np.dot(transformmatrix, transform.twoDScaling(sx, sy))
                    case '6':
                        break
                    case _:
                        print("Invalid option, please try again.")

    for coord in coordinates_list:
        x, y = map(float, coord.split(','))
        old_coordinates.append((x, y))
        old_point = np.array([x, y, 1])
        new_point = np.dot(transformmatrix, old_point)
        new_point = np.squeeze(np.asarray(new_point))
        new_coordinates.append((new_point[0], new_point[1]))

    drawTraingle(old_points=old_coordinates, new_points=new_coordinates, label="Transformation")

if __name__ == "__main__":
    main()
```

Here we plan to draw a triangle using OpenGL for visualizing the 2D transformation, whose code implementation is shown below:



```
import glfw
from OpenGL.GL import *

X_BASE = 500
Y_BASE = 500

def drawTraingle(old_points, new_points, label, pointsize=5):
    if not glfw.init():
        print("Failed to initialize GLFW")
        return -1

    window = glfw.create_window(1000, 1000, label, None,
None)if not window:
        glfw.terminate()
        return -1
    glfw.make_context_current(window)

    while not glfw.window_should_close(window):
        glClearColor(1.0, 1.0, 1.0, 1.0)
        glClear(GL_COLOR_BUFFER_BIT)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(0, 1200 ,1200, 0, 0, 1)

        # Draw old triangle in blue
        glColor4fv([0.0, 0.0, 1.0, 1.0])
        glPointSize(pointsize)
        glBegin(GL_TRIANGLES)
        for point in old_points:

            glVertex2f(point[0] + X_BASE, point[1] + Y_BASE)
        glEnd()

        # Draw new triangle in red
        glColor4fv([1.0, 0.0, 0.0, 1.0])
        glPointSize(pointsize)
        glBegin(GL_TRIANGLES)
        for point in new_points:

            glVertex2f(point[0] + X_BASE, point[1] + Y_BASE)
        glEnd()

        glfw.swap_buffers(window)
        glfw.poll_events()

    glfw.terminate()
```

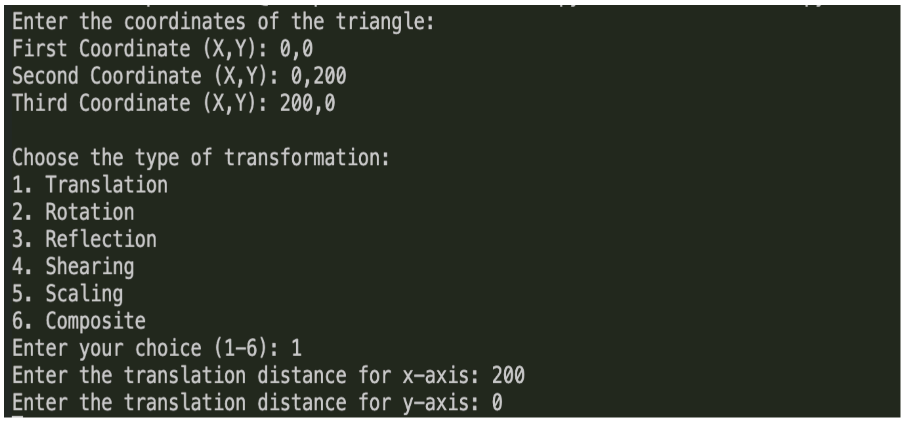
Output:

Here, the red triangle represents the transformed triangle, whereas the blue triangle represents the original triangle.

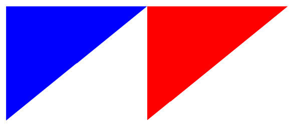
So now below is some of the output of single as well a composite transformation

- Translation

Translation of (0,0), (0,200),(200,0) by T(200,0)

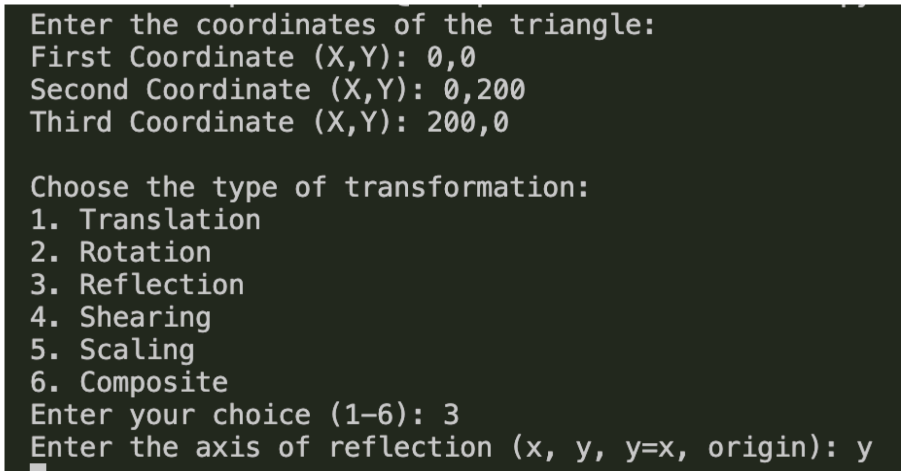


```
Enter the coordinates of the triangle:
First Coordinate (X,Y): 0,0
Second Coordinate (X,Y): 0,200
Third Coordinate (X,Y): 200,0

Choose the type of transformation:
1. Translation
2. Rotation
3. Reflection
4. Shearing
5. Scaling
6. Composite
Enter your choice (1-6): 1
Enter the translation distance for x-axis: 200
Enter the translation distance for y-axis: 0
```

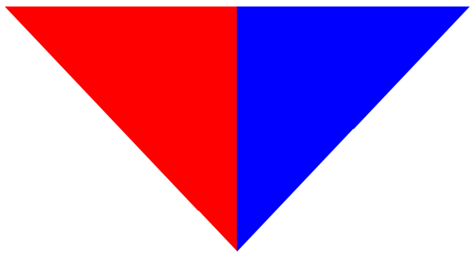


- Reflection

Reflection of triangle along x=0 line

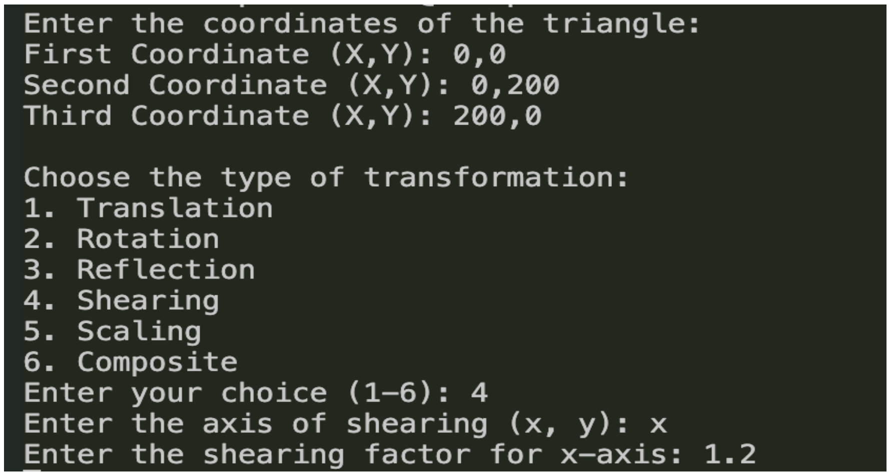


```
Enter the coordinates of the triangle:
First Coordinate (X,Y): 0,0
Second Coordinate (X,Y): 0,200
Third Coordinate (X,Y): 200,0

Choose the type of transformation:
1. Translation
2. Rotation
3. Reflection
4. Shearing
5. Scaling
6. Composite
Enter your choice (1-6): 3
Enter the axis of reflection (x, y, y=x, origin): y
```

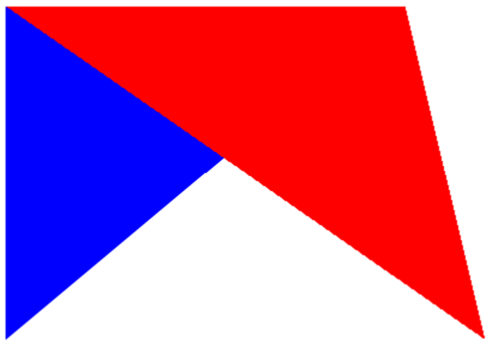


- Shearing

Shearing along X-axis with shx=1.2



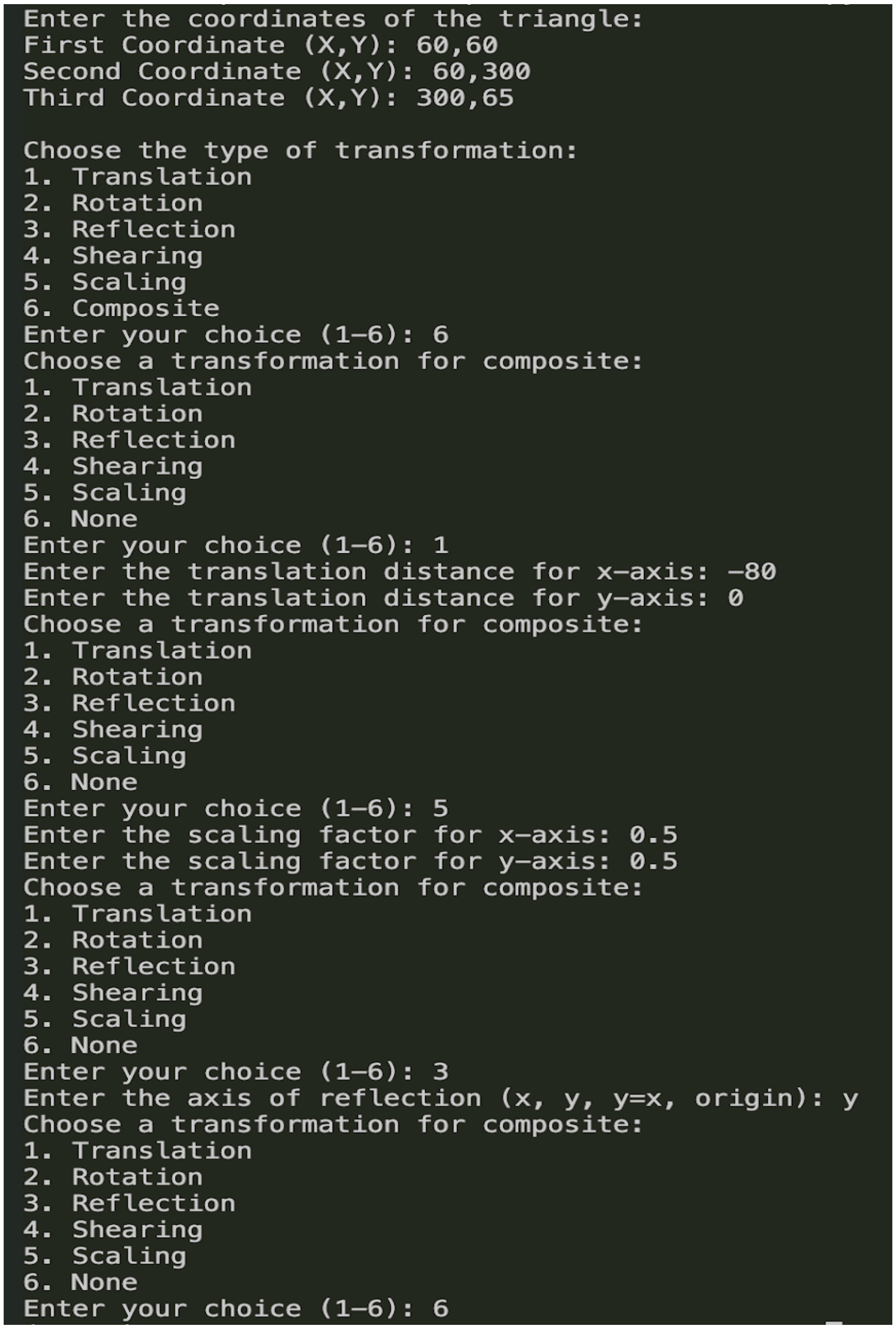
```
Enter the coordinates of the triangle:
First Coordinate (X,Y): 0,0
Second Coordinate (X,Y): 0,200
Third Coordinate (X,Y): 200,0

Choose the type of transformation:
1. Translation
2. Rotation
3. Reflection
4. Shearing
5. Scaling
6. Composite
Enter your choice (1-6): 4
Enter the axis of shearing (x, y): x
Enter the shearing factor for x-axis: 1.2
```



- Composite

Now here, we will perform composite transformations to visualize scaling.
Here are the series of transformation we plan to perform

1. Translation by T(-80,0)
2. Scaling S(0.5,0.5)
3. Reflection Along X=0 Line



```
Enter the coordinates of the triangle:
First Coordinate (X,Y): 60,60
Second Coordinate (X,Y): 60,300
Third Coordinate (X,Y): 300,65

Choose the type of transformation:
1. Translation
2. Rotation
3. Reflection
4. Shearing
5. Scaling
6. Composite
Enter your choice (1-6): 6
Choose a transformation for composite:
1. Translation
2. Rotation
3. Reflection
4. Shearing
5. Scaling
6. None
Enter your choice (1-6): 1
Enter the translation distance for x-axis: -80
Enter the translation distance for y-axis: 0
Choose a transformation for composite:
1. Translation
2. Rotation
3. Reflection
4. Shearing
5. Scaling
6. None
Enter your choice (1-6): 5
Enter the scaling factor for x-axis: 0.5
Enter the scaling factor for y-axis: 0.5
Choose a transformation for composite:
1. Translation
2. Rotation
3. Reflection
4. Shearing
5. Scaling
6. None
Enter your choice (1-6): 3
Enter the axis of reflection (x, y, y=x, origin): y
Choose a transformation for composite:
1. Translation
2. Rotation
3. Reflection
4. Shearing
5. Scaling
6. None
Enter your choice (1-6): 6
```