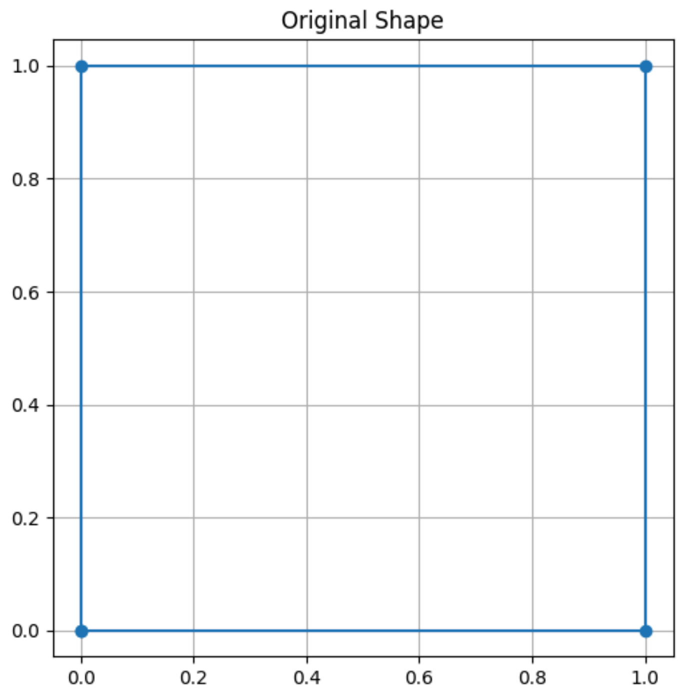
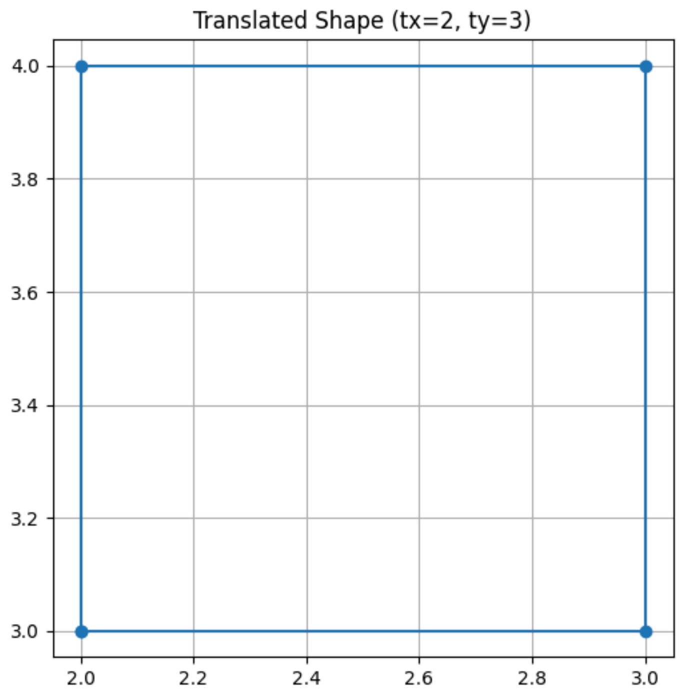
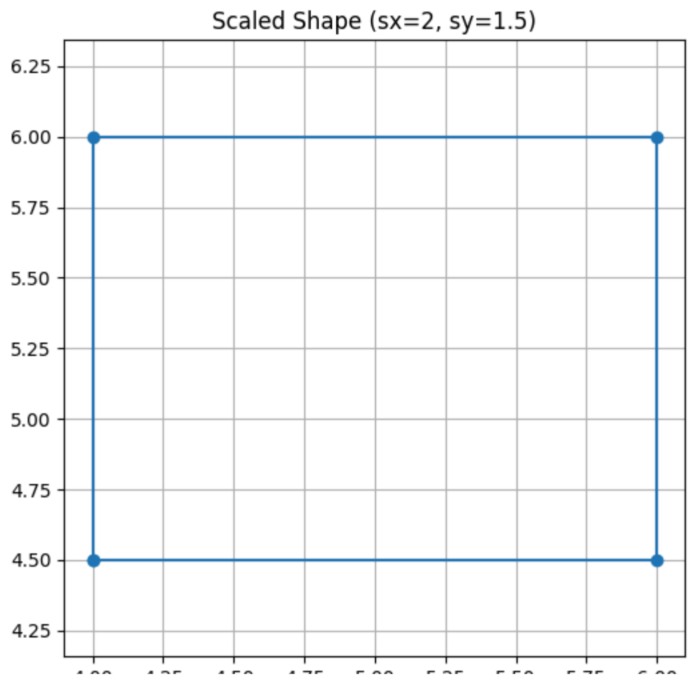
```
import numpy as np
import matplotlib.pyplot as plt

# Function to plot a 2D shape
def plot_shape(shape, title):
    plt.figure(figsize=(6, 6))
    plt.plot(shape[0, :], shape[1, :], marker='o')  # Connect points and mark vertices
    plt.title(title)
    plt.axis('equal')
    plt.grid()
    plt.show()

# Define a 2D shape (square)
square = np.array([
    [0, 0, 1],  # Bottom-left
    [1, 0, 1],  # Bottom-right
    [1, 1, 1],  # Top-right
    [0, 1, 1],  # Top-left
    [0, 0, 1]   # Closing the square
]).T  # Transpose for matrix multiplication

# Display the original shape
plot_shape(square, "Original Shape")

# 1. Translation
def translate(shape, tx, ty):
    translation_matrix = np.array([[1, 0, tx],
                                   [0, 1, ty],
                                   [0, 0, 1]])
    return np.dot(translation_matrix, shape)

translated_shape = translate(square, 2, 3)
plot_shape(translated_shape, "Translated Shape (tx=2, ty=3)")

# 2. Scaling
def scale(shape, sx, sy):
    scale_matrix = np.array([[sx, 0, 0],
                             [0, sy, 0],
                             [0, 0, 1]])
    return np.dot(scale_matrix, shape)

scaled_shape = scale(translated_shape, 2, 1.5)
plot_shape(scaled_shape, "Scaled Shape (sx=2, sy=1.5)")
plot_shape(square, "Original Shape")

# 3. Rotation
def rotate(shape, angle):
    rad = np.radians(angle)  # Convert degrees to radians
    rotation_matrix = np.array([[np.cos(rad), -np.sin(rad), 0],
                                [np.sin(rad), np.cos(rad), 0],
                                [0, 0, 1]])
    return np.dot(rotation_matrix, shape)

rotated_shape = rotate(scaled_shape, 45)
plot_shape(rotated_shape, "Rotated Shape (45 degrees)")
plot_shape(square, "Original Shape")

# 4. Reflection
def reflect(shape, axis):
    if axis == 'x':
        reflection_matrix = np.array([[1, 0, 0],
                                      [0, -1, 0],
                                      [0, 0, 1]])
    elif axis == 'y':
        reflection_matrix = np.array([[-1, 0, 0],
                                      [0, 1, 0],
                                      [0, 0, 1]])
    else:
        raise ValueError("Axis must be 'x' or 'y'")
    return np.dot(reflection_matrix, shape)

reflected_shape = reflect(rotated_shape, 'x')
plot_shape(reflected_shape, "Reflected Shape (X-axis)")
plot_shape(square, "Original Shape")

# 5. Shearing
def shear(shape, shx, shy):
    shear_matrix = np.array([[1, shx, 0],
                             [shy, 1, 0],
                             [0, 0, 1]])
    return np.dot(shear_matrix, shape)
```

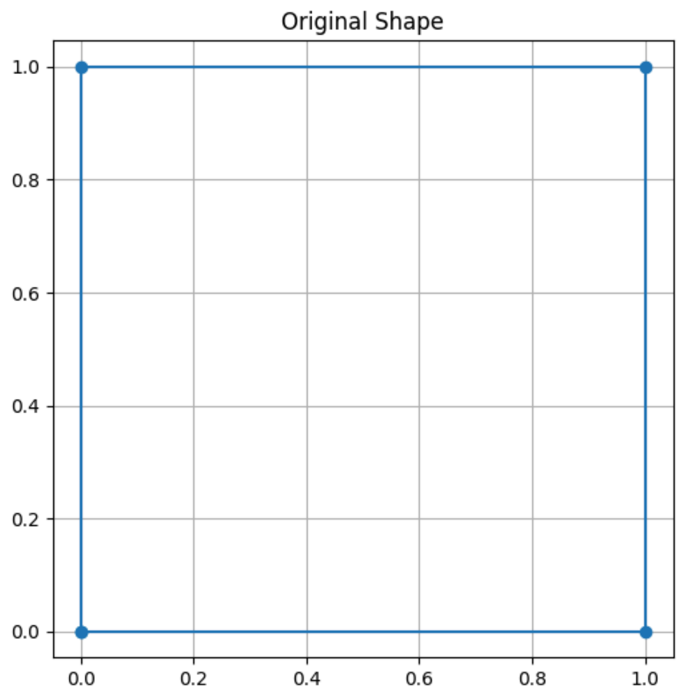
```
sheared_shape = shear(reflected_shape, 0.5, 0.2)
plot_shape(sheared_shape, "Sheared Shape (shx=0.5, shy=0.2)")
plot_shape(square, "Original Shape")

# Composite Transformation (e.g., Translate -> Scale -> Rotate)
composite_transformation = rotate(scale(translate(square, 3, 3), 1.2, 0.8), 30)
plot_shape(composite_transformation, "Composite Transformation (Translation + Scaling + Rotation)")
plot_shape(square, "Original Shape")
```

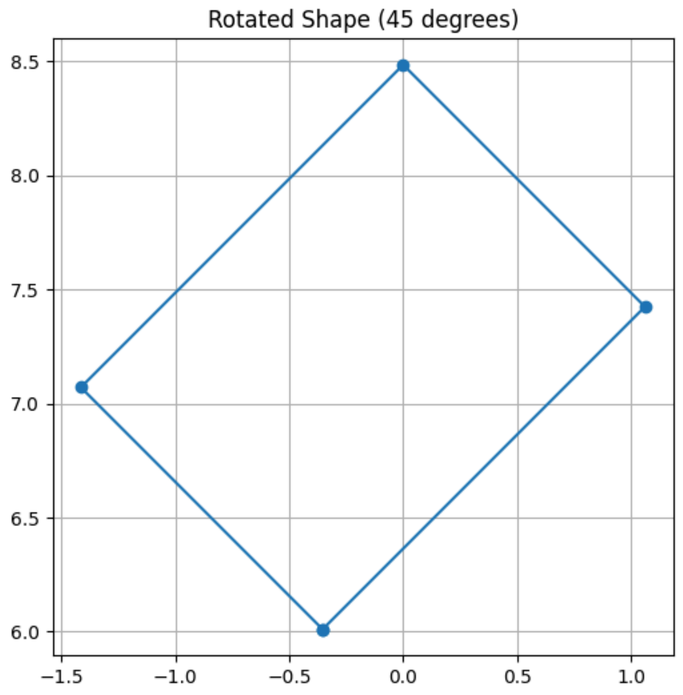

Original Shape
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

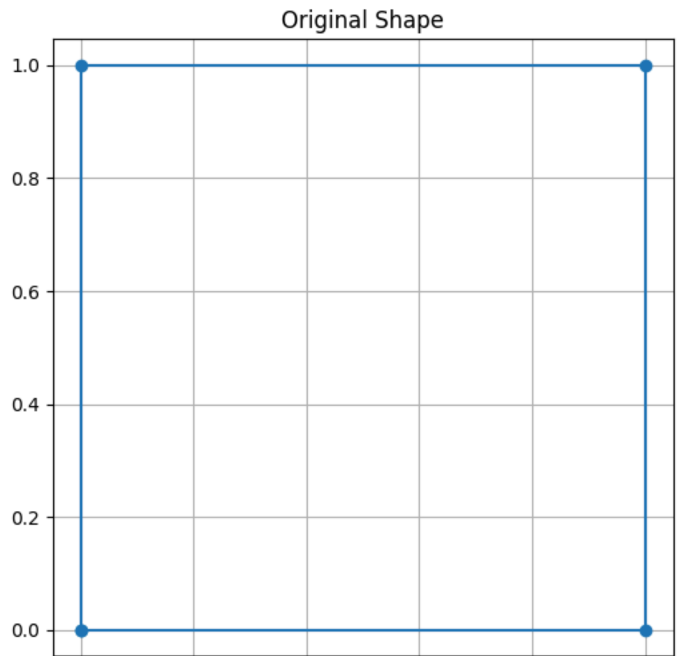

Translated Shape (tx=2, ty=3)
4.0
3.8
3.6
3.4
3.2
3.0
2.0
2.2
2.4
2.6
2.8
3.0


Scaled Shape (sx=2, sy=1.5)
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25

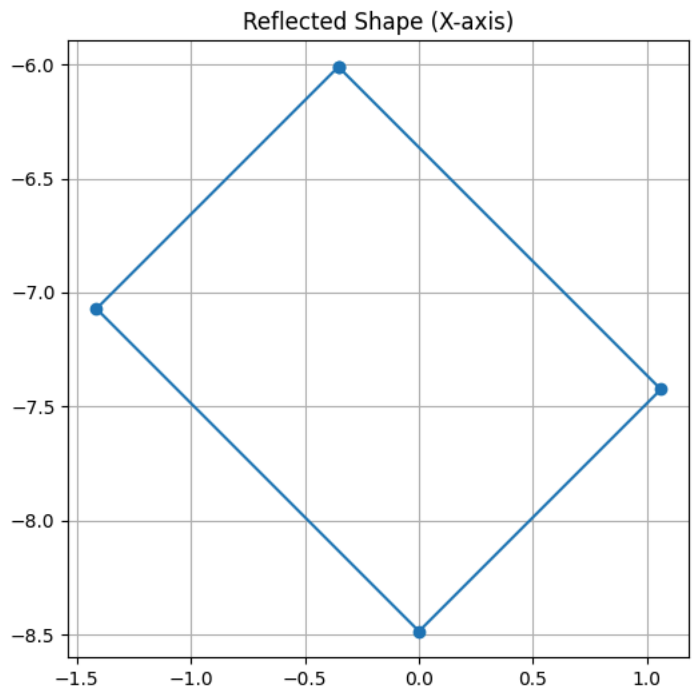
4.00
4.25
4.50
4.75
5.00
5.25
5.50
5.75
6.00

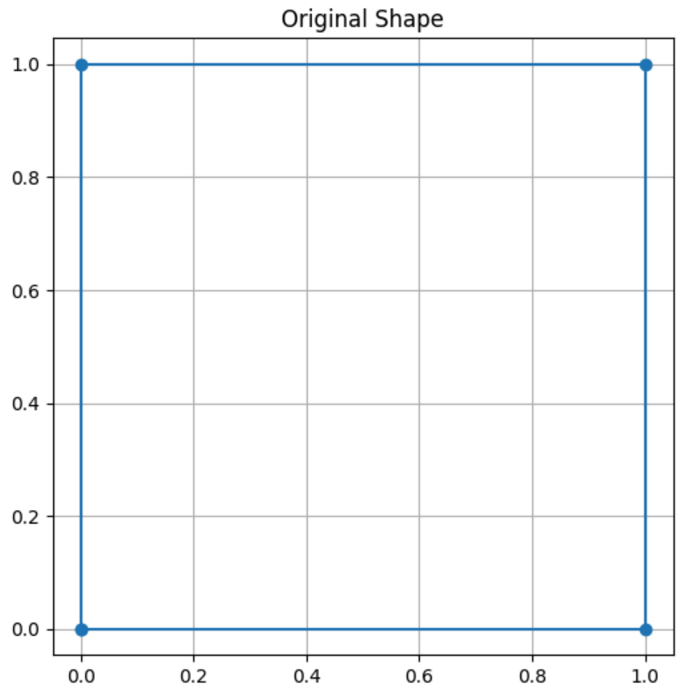

Original Shape
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

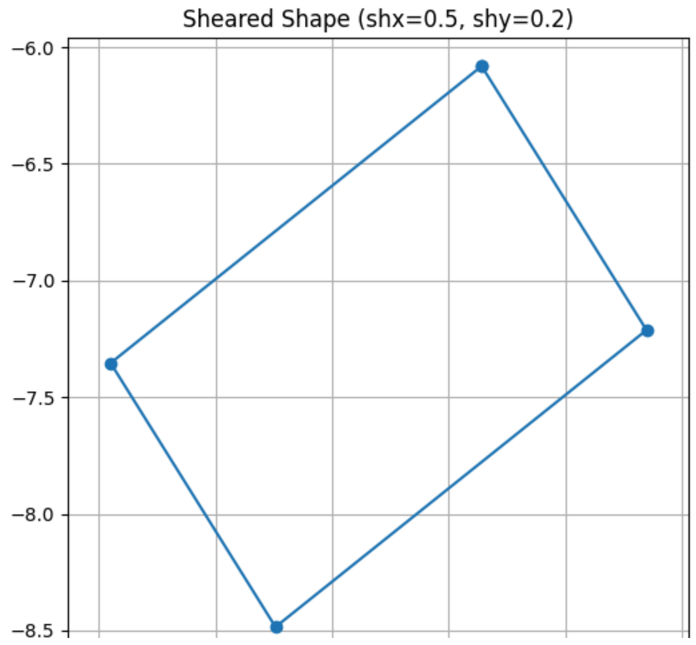

Rotated Shape (45 degrees)
8.5
8.0
7.5
7.0
6.5
6.0
−1.5
−1.0
−0.5
0.0
0.5
1.0

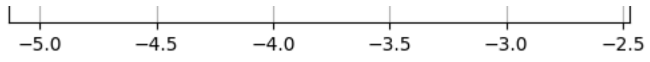

Original Shape
1.0
0.8
0.6
0.4
0.2
0.0

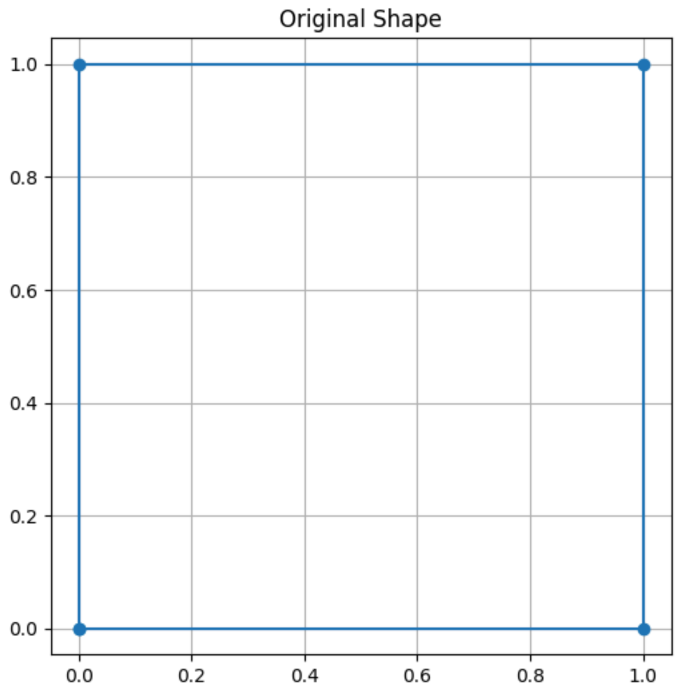
0.0
0.2
0.4
0.6
0.8
1.0

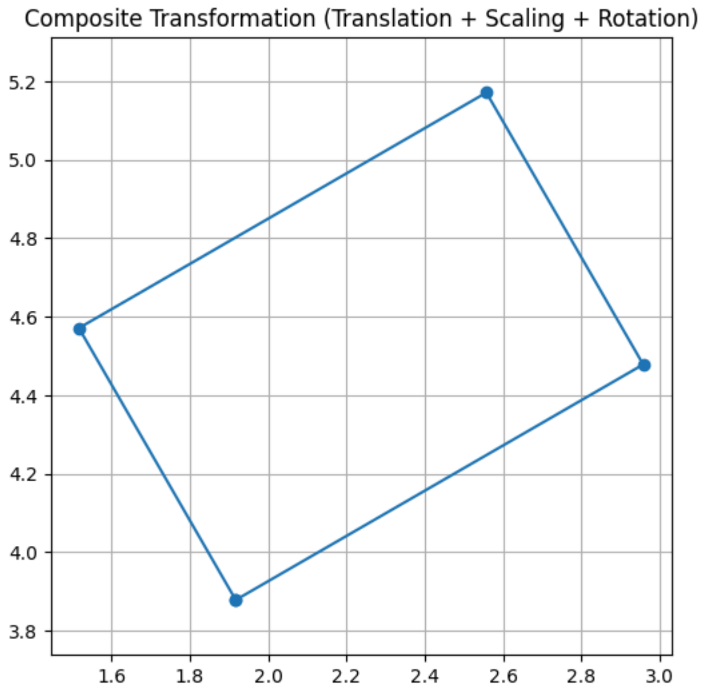

Reflected Shape (X-axis)
−6.0
−6.5
−7.0
−7.5
−8.0
−8.5
−1.5
−1.0
−0.5
0.0
0.5
1.0

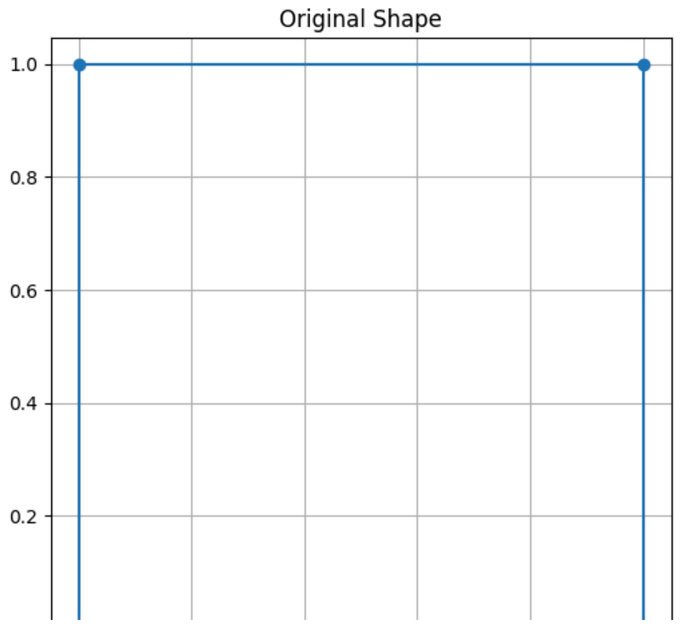

Original Shape
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

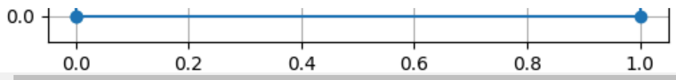

Sheared Shape (shx=0.5, shy=0.2)
−6.0
−6.5
−7.0
−7.5
−8.0
−8.5


−5.0
−4.5
−4.0
−3.5
−3.0
−2.5


Original Shape
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Composite Transformation (Translation + Scaling + Rotation)
5.2
5.0
4.8
4.6
4.4
4.2
4.0
3.8
1.6
1.8
2.0
2.2
2.4
2.6
2.8
3.0


Original Shape
1.0
0.8
0.6
0.4
0.2


0.0
0.0
0.2
0.4
0.6
0.8
1.0

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function to display an image in Colab
def show_image(title, image):
    plt.figure(figsize=(8, 6))
    plt.title(title)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for correct color display
    plt.axis('off')
    plt.show()

from google.colab import files
uploaded = files.upload()  # Upload your image
image_path = list(uploaded.keys())[0]
image = cv2.imread(image_path)
show_image("Original Image", image)

# Scaling Function
def scale_image(image, fx, fy):
    if fx <= 0 or fy <= 0:
        raise ValueError("Scaling factors (fx and fy) must be positive.")
    return cv2.resize(image, None, fx=fx, fy=fy, interpolation=cv2.INTER_LINEAR)

# Example Scaling
scaled_up = scale_image(image, 1.5, 1.5)
show_image("Scaled Image (1.5x)", scaled_up)

scaled_down = scale_image(image, 0.5, 0.5)
show_image("Scaled Image (0.5x)", scaled_down)

# Translation Function
def translate_image(image, tx, ty):
    rows, cols = image.shape[:2]
    M = np.float32([[1, 0, tx], [0, 1, ty]])
    return cv2.warpAffine(image, M, (cols, rows))

translated_image = translate_image(image, 100, 50)
show_image("Translated Image (tx=100, ty=50)", translated_image)

# Rotation Function
def rotate_image(image, angle):
    rows, cols = image.shape[:2]
    M = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
    return cv2.warpAffine(image, M, (cols, rows))

rotated_image = rotate_image(image, 45)
show_image("Rotated Image (45 degrees)", rotated_image)

# Reflection Function
def reflect_image(image):
    return cv2.flip(image, 1)  # Flip horizontally

reflected_image = reflect_image(image)
show_image("Reflected Image (Horizontal Flip)", reflected_image)

# Cropping Function
def crop_image(image, x, y, width, height):
    return image[y:y+height, x:x+width]

cropped_image = crop_image(image, 50, 50, 200, 200)
show_image("Cropped Image", cropped_image)

# Shearing Function (X-axis)
def shear_image_x(image, shear_factor):
    rows, cols = image.shape[:2]
    M = np.float32([[1, shear_factor, 0], [0, 1, 0]])
    return cv2.warpAffine(image, M, (cols + int(shear_factor * rows), rows))

sheared_image_x = shear_image_x(image, 0.3)
show_image("Sheared Image (X-axis, factor=0.3)", sheared_image_x)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Screenshot 2025-01-30 112851.png to Screenshot 2025-01-30 112851.png
```

Original Image

Scaled Image (1.5x)

Scaled Image (0.5x)