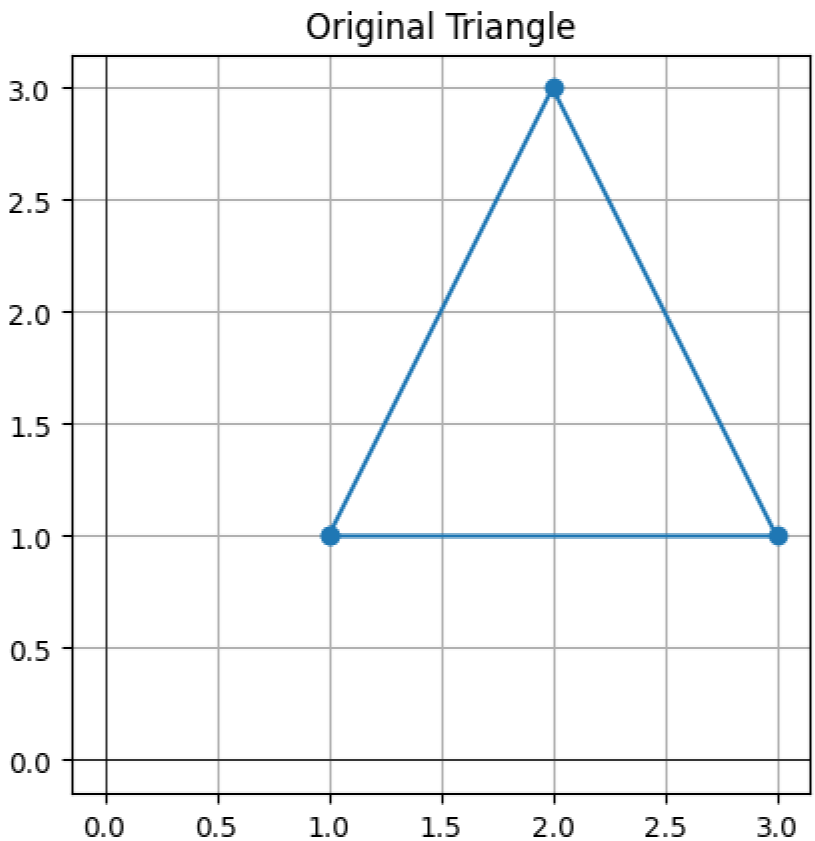
2D GEOMETRIC TRANSFORMATIONS USING HO-MOGENEOUS COORDINATES

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

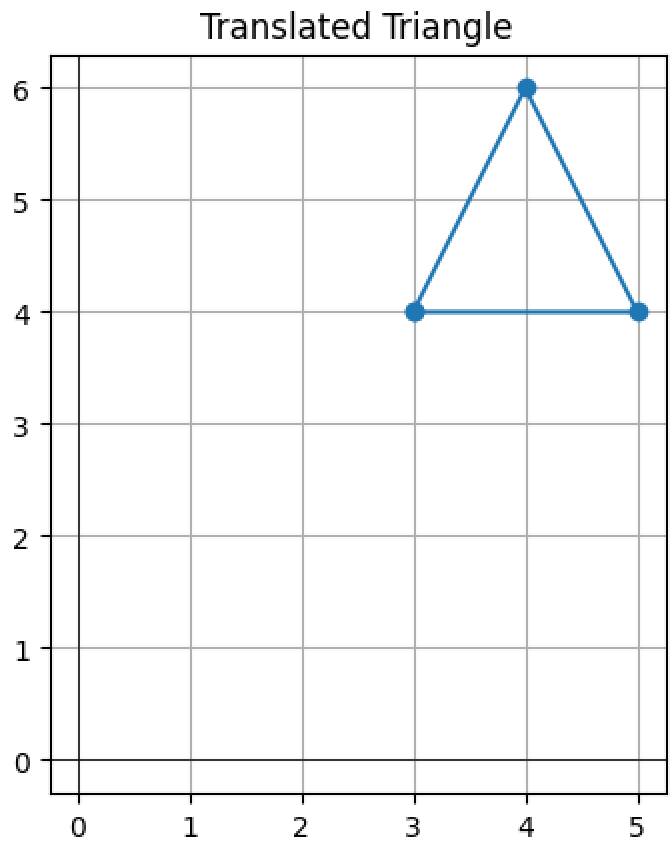
In [4]:
```
def plot_shape(points, title="Shape"):
    points = np.vstack([points, points[0]])  # Close the shape
    plt.figure()
    plt.plot(points[:, 0], points[:, 1], marker='o')
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)
    plt.gca().set_aspect('equal', 'box')
    plt.grid(True)
    plt.title(title)
    plt.show()


def apply_transformation(points, matrix):
    return (matrix @ points.T).T
```

In [5]:
```
triangle = np.array([
    [1, 1],
    [3, 1],
    [2, 3]
])

plot_shape(triangle, "Original Triangle")
```

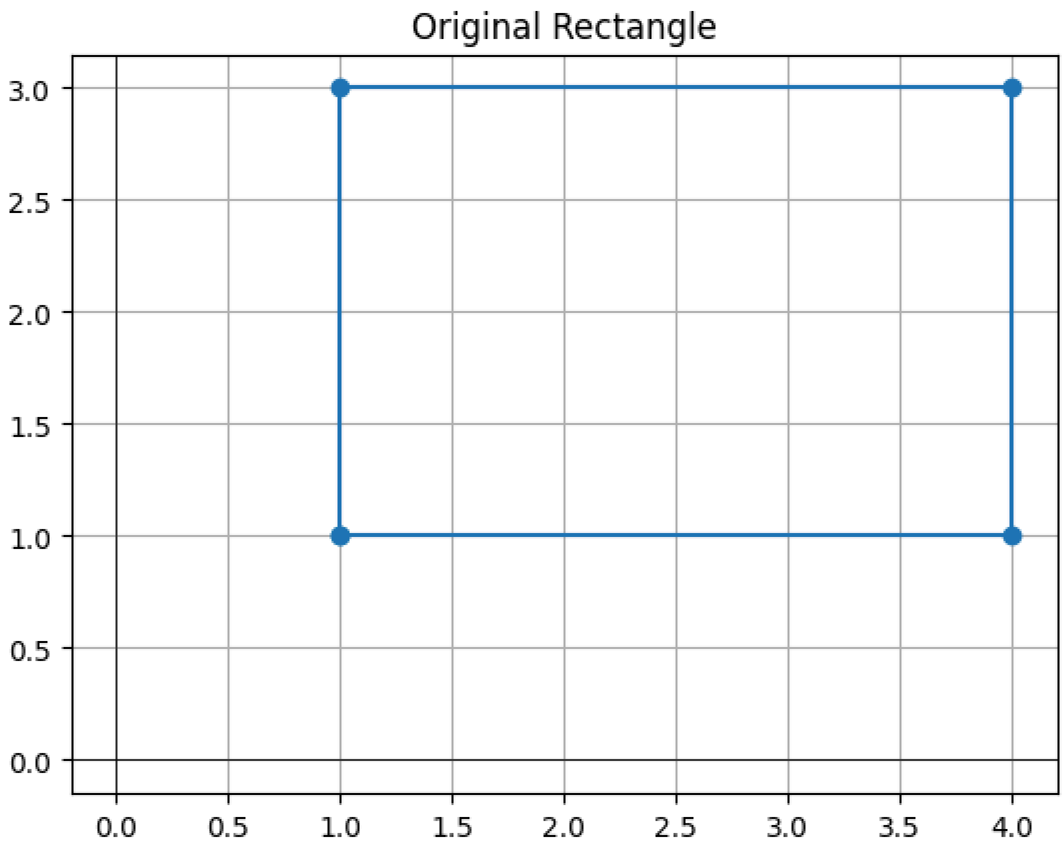


In [8]:

```
tx, ty = 2, 3

translation_matrix = np.array([
    [1, 0, tx],
    [0, 1, ty],
    [0, 0, 1]
])

triangle_h = np.hstack([triangle, np.ones((3,1))])
translated_triangle = (translation_matrix @ triangle_h.T).T[:, :2]

plot_shape(translated_triangle, "Translated Triangle")
```

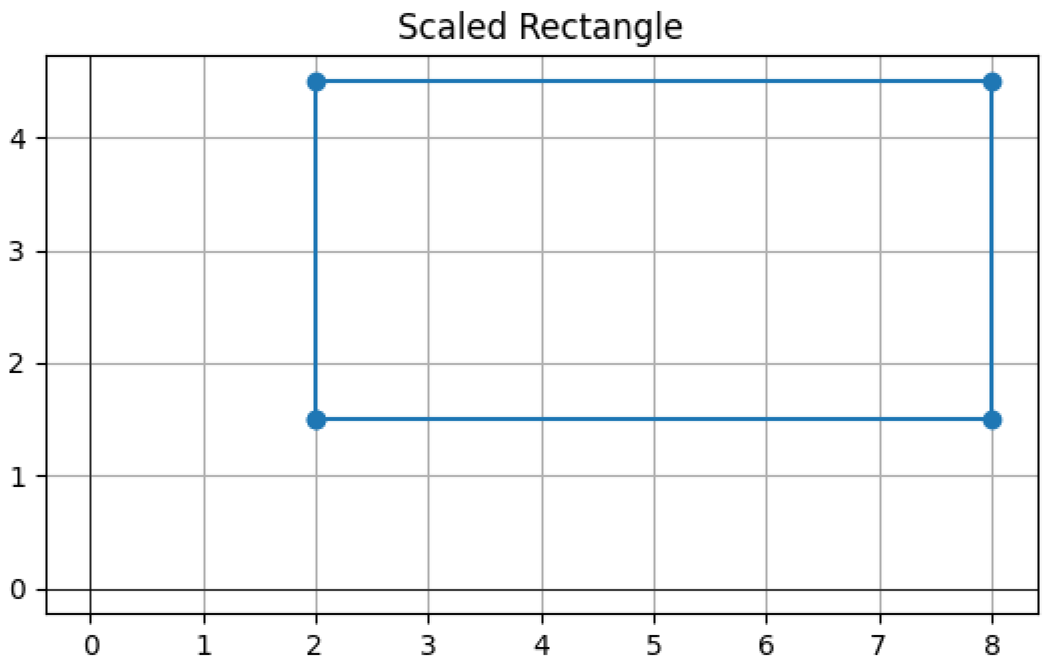


In [6]:
```
rectangle = np.array([
    [1, 1],
    [4, 1],
    [4, 3],
    [1, 3]
])

plot_shape(rectangle, "Original Rectangle")
```

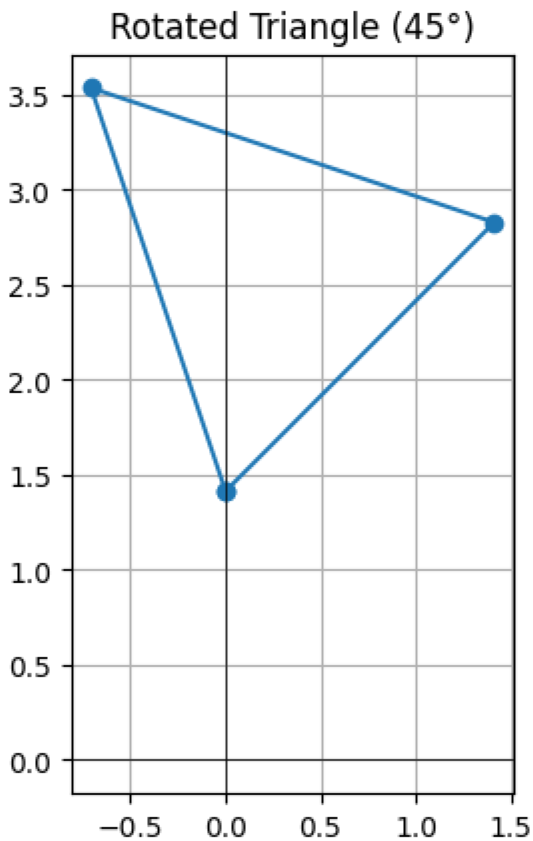


In [7]:

```
sx, sy = 2, 1.5

scaling_matrix = np.array([
    [sx, 0],
    [0, sy]
])

scaled_rectangle = apply_transformation(rectangle, scaling_matrix)

plot_shape(scaled_rectangle, "Scaled Rectangle")
```

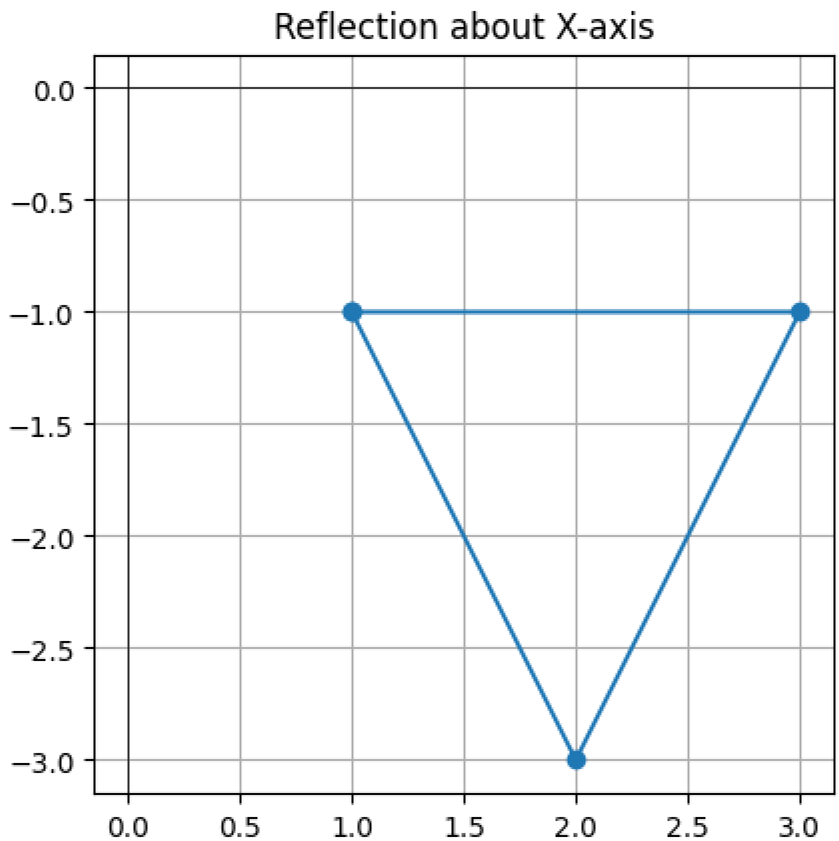


In [9]:
```
theta = np.radians(45)

rotation_matrix = np.array([
    [np.cos(theta), -np.sin(theta)],
    [np.sin(theta),  np.cos(theta)]
])

rotated_triangle = apply_transformation(triangle, rotation_matrix)

plot_shape(rotated_triangle, "Rotated Triangle (45°)")
```

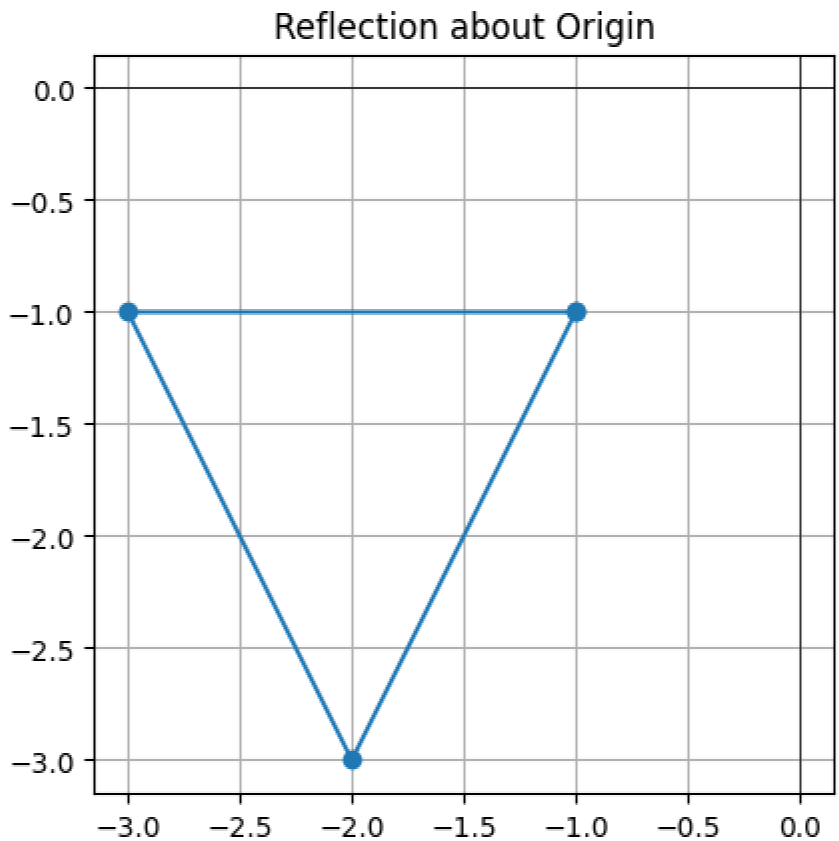


In [10]:

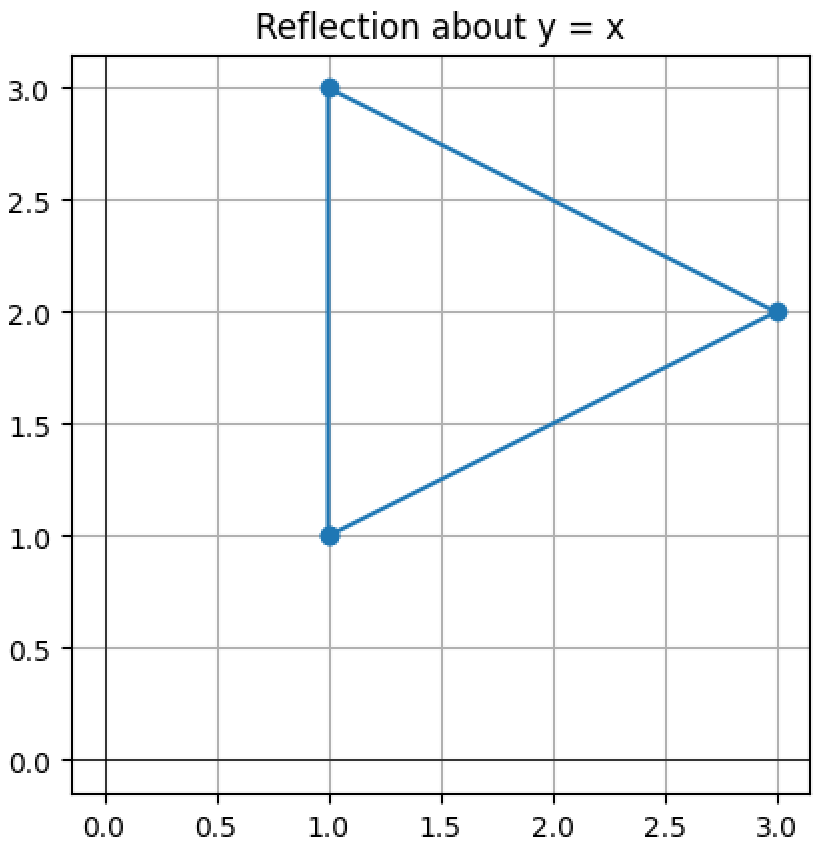
```
reflect_x = np.array([
    [1, 0],
    [0, -1]
])

ref_x_triangle = apply_transformation(triangle, reflect_x)

plot_shape(ref_x_triangle, "Reflection about X-axis")
```



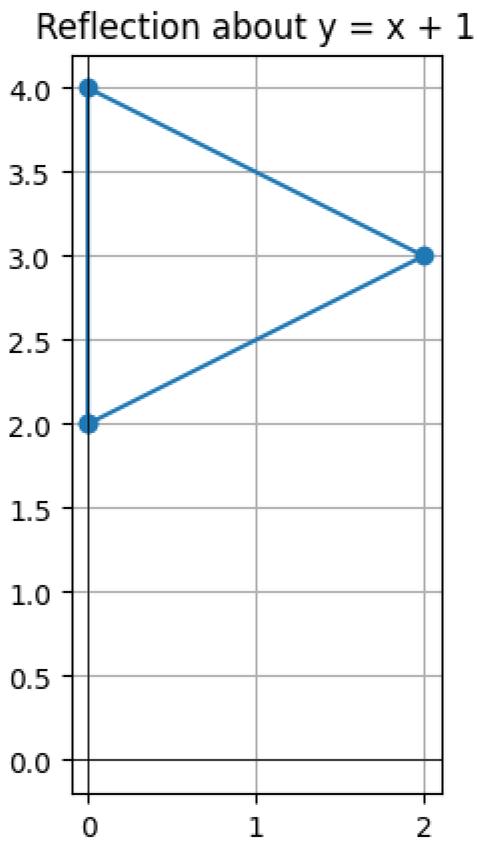
In [11]:
```
reflect_origin = np.array([
    [-1, 0],
    [0, -1]
])

ref_origin_triangle = apply_transformation(triangle, reflect_origin)

plot_shape(ref_origin_triangle, "Reflection about Origin")
```



In [12]:
```
reflect_y_eq_x = np.array([
    [0, 1],
    [1, 0]
])

ref_yx_triangle = apply_transformation(triangle, reflect_y_eq_x)

plot_shape(ref_yx_triangle, "Reflection about y = x")
```



In [13]:
```
def reflect_about_line(points, m, c):
    reflected = []
    for x, y in points:
        d = (x + (y - c)*m) / (1 + m**2)
        x_ref = 2*d - x
        y_ref = 2*d*m - y + 2*c
        reflected.append([x_ref, y_ref])
    return np.array(reflected)

m = 1
c = 1

ref_line_triangle = reflect_about_line(triangle, m, c)

plot_shape(ref_line_triangle, "Reflection about y = x + 1")
```


Reflection about y = x + 1
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2