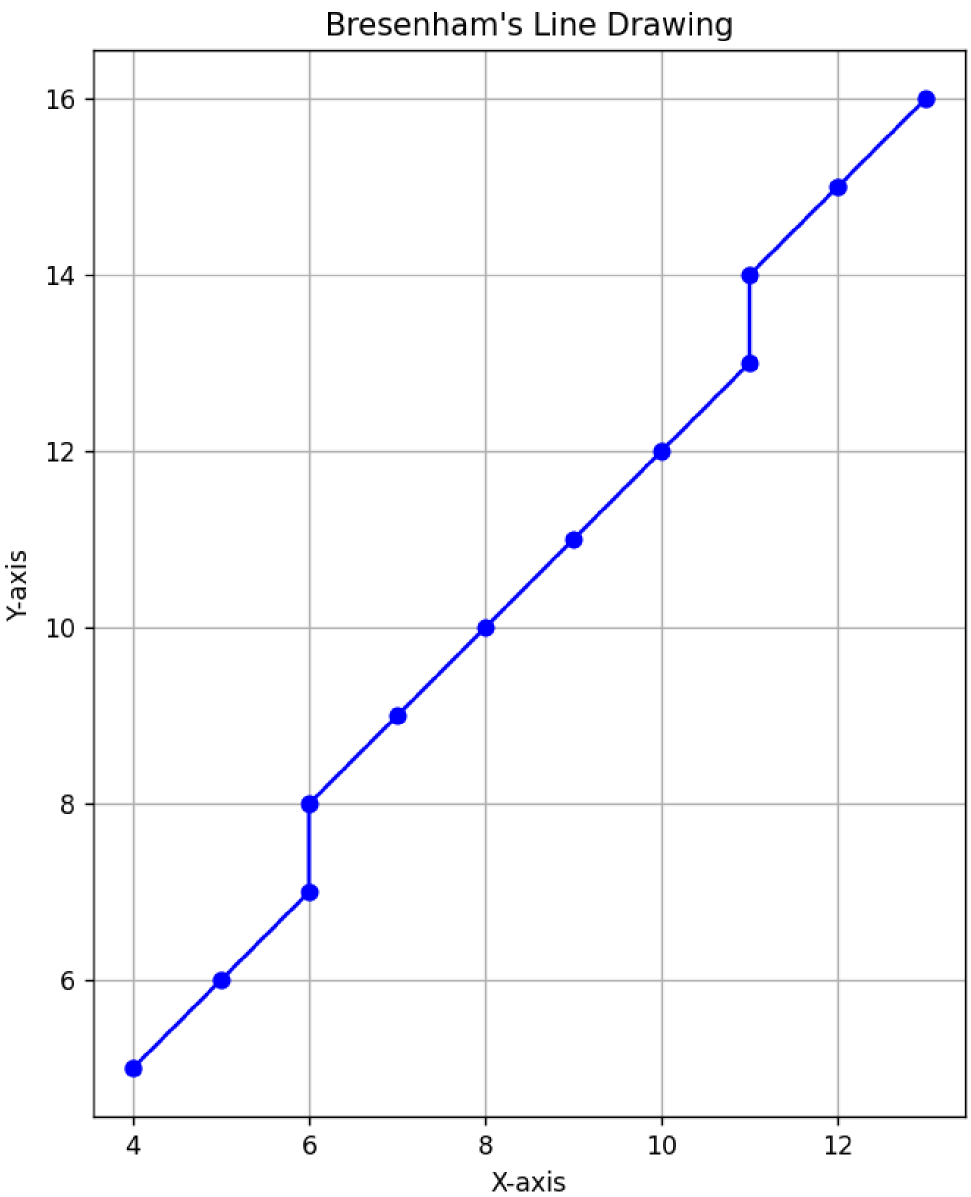
1. **Implement Bresenham's Line Drawing Algorithm**

```
import matplotlib.pyplot as plt

def bresenham_line(x1, y1, x2, y2):
    points = []
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x2 > x1 else -1
    sy = 1 if y2 > y1 else -1
    err = dx - dy

    while True:
        points.append((x1, y1))

        if x1 == x2 and y1 == y2:
            break

        e2 = 2 * err

        if e2 > -dy:
            err -= dy
            x1 += sx

        if e2 < dx:
            err += dx
            y1 += sy

    return points

def main():
    print("Bresenham's Line Drawing Algorithm")
    try:
        x1 = int(input("Enter x1: "))
        y1 = int(input("Enter y1: "))
        x2 = int(input("Enter x2: "))
        y2 = int(input("Enter y2: "))

        points = bresenham_line(x1, y1, x2, y2)

        # Extract x and y coordinates
        x_coords, y_coords = zip(*points)

        # Plotting the line
        plt.plot(x_coords, y_coords, marker='o', color='blue')
        plt.title("Bresenham's Line Drawing")
        plt.xlabel("X-axis")
        plt.ylabel("Y-axis")
```

```
        plt.grid(True)
        plt.gca().set_aspect('equal', adjustable='box')
        plt.show()

    except ValueError:
        print("Please enter valid integer values.")

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

OUTPUT

Bresenham's Line Drawing Algorithm
Enter x1: 4
Enter y1: 5
Enter x2: 13
Enter y2: 16

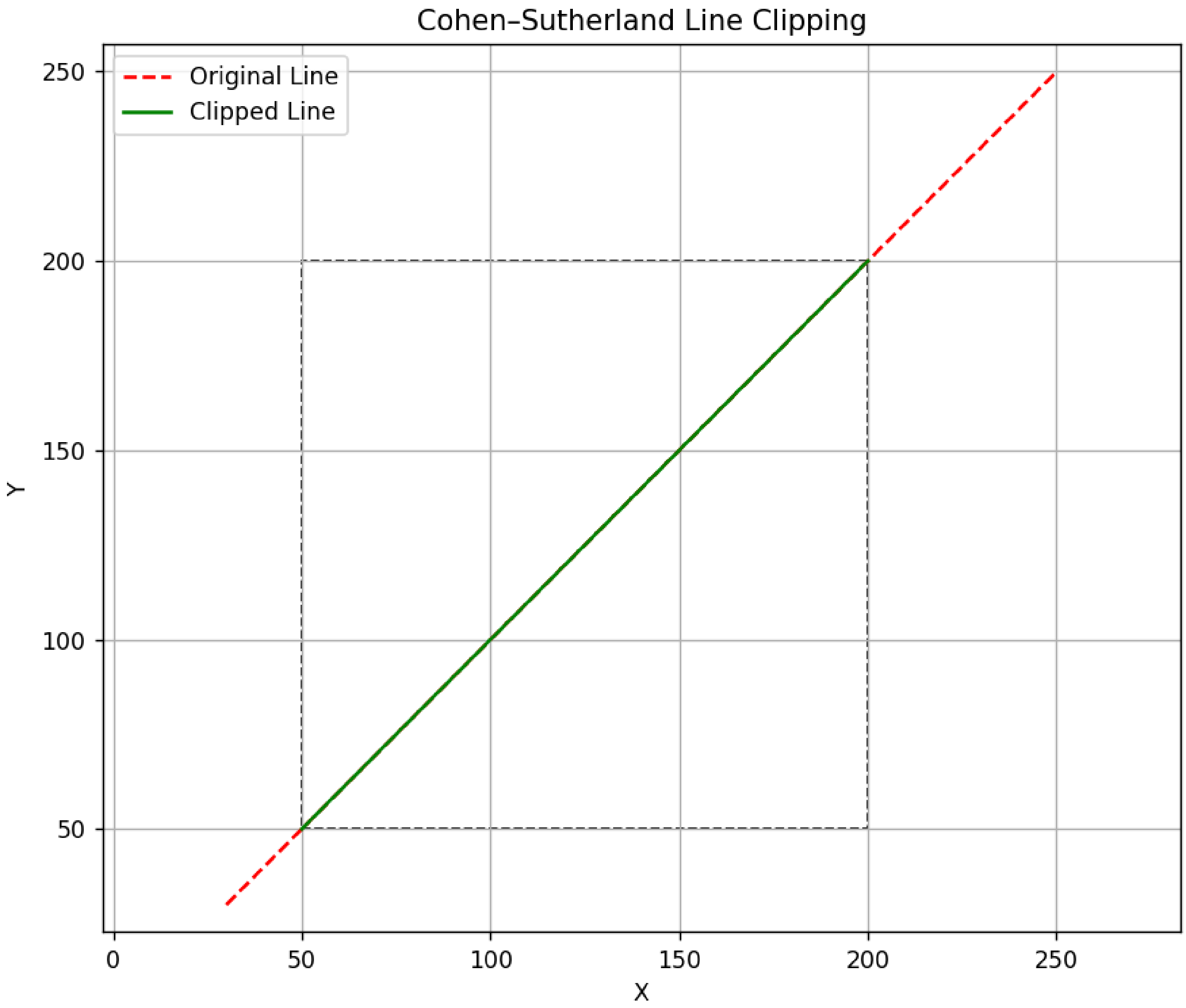


2. **Implement Cohen-Sutherland Line Clipping Algorithm**

```
import matplotlib.pyplot as plt
INSIDE, LEFT, RIGHT, BOTTOM, TOP = 0, 1, 2, 4, 8

def compute_code(x, y, rect):
  xmin, xmax, ymin, ymax = rect
  return (LEFT if x < xmin else RIGHT if x > xmax else 0) | \
      (BOTTOM if y < ymin else TOP if y > ymax else 0)

def clip_line(x1, y1, x2, y2, rect):
  code1, code2 = compute_code(x1, y1, rect), compute_code(x2, y2, rect)
  while True:
    if not (code1 | code2):
      return x1, y1, x2, y2
    elif code1 & code2:
      return None
    out = code1 or code2
    xmin, xmax, ymin, ymax = rect
    if out & TOP:    x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1); y = ymax
    elif out & BOTTOM:x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1); y = ymin
    elif out & RIGHT: y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1); x = xmax
    elif out & LEFT:  y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1); x = xmin
    if out == code1: x1, y1, code1 = x, y, compute_code(x, y, rect)
    else:         x2, y2, code2 = x, y, compute_code(x, y, rect)

def main():
  print("Cohen–Sutherland Line Clipping")
  x1, y1 = map(float, input("Enter x1 y1: ").split())
  x2, y2 = map(float, input("Enter x2 y2: ").split())
  rect = tuple(map(float, input("Enter xmin xmax ymin ymax: ").split()))
  clipped = clip_line(x1, y1, x2, y2, rect)

  fig, ax = plt.subplots()
  ax.add_patch(plt.Rectangle((rect[0], rect[2]), rect[1]-rect[0], rect[3]-rect[2], edgecolor='black', fill=False,
linestyle='--'))
  ax.plot([x1, x2], [y1, y2], 'r--', label="Original Line")
  if clipped:
    ax.plot([clipped[0], clipped[2]], [clipped[1], clipped[3]], 'g-', label="Clipped Line")
  else:
    print("Line is completely outside the clipping window.")

  plt.title("Cohen–Sutherland Line Clipping")
  plt.xlabel("X"); plt.ylabel("Y"); plt.grid(True); plt.axis('equal'); plt.legend()
  plt.show()

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

OUTPUT

Cohen–Sutherland Line Clipping
Enter x1 y1: 30 30
Enter x2 y2: 250 250
Enter xmin xmax ymin ymax: 50 200 50 200



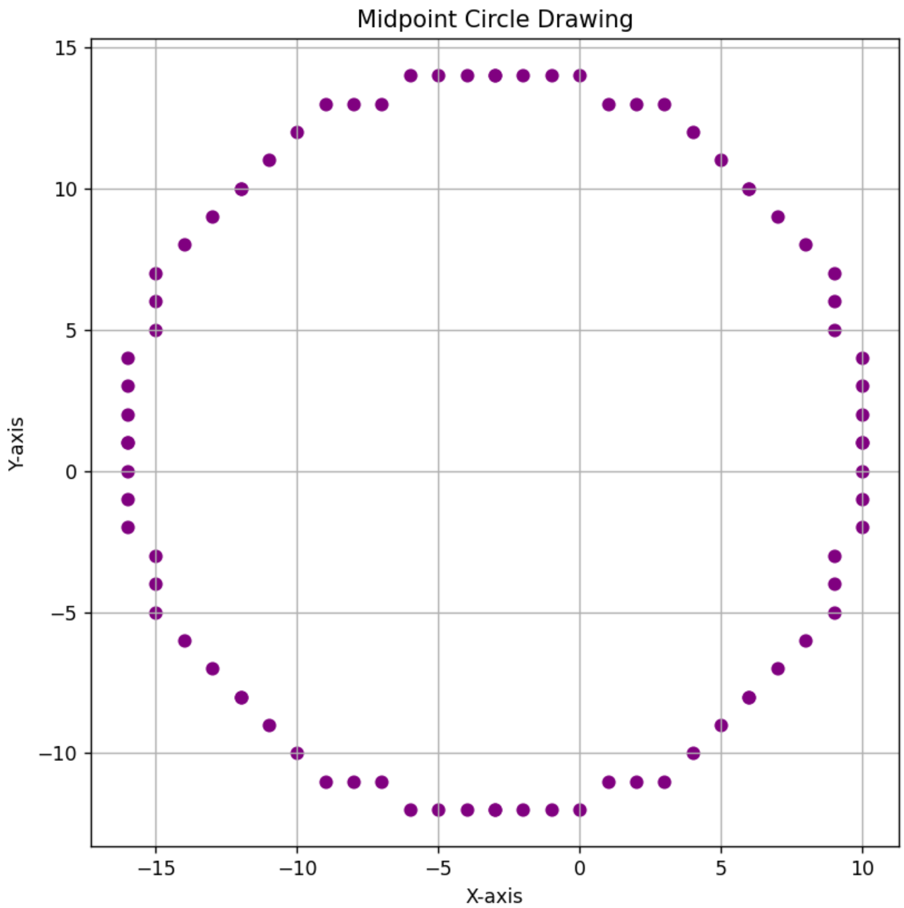
3. **Implement Midpoint Circle Drawing Algorithm**

```
import matplotlib.pyplot as plt

def plot_circle_points(xc, yc, x, y, points):
    # 8-way symmetry
    points.extend([
        (xc + x, yc + y),
        (xc - x, yc + y),
        (xc + x, yc - y),
        (xc - x, yc - y),
        (xc + y, yc + x),
        (xc - y, yc + x),
        (xc + y, yc - x),
        (xc - y, yc - x)
    ])

def midpoint_circle(xc, yc, r):
    x = 0
    y = r
    d = 1 - r
    points = []

    while x <= y:
        plot_circle_points(xc, yc, x, y, points)

        if d < 0:
            d = d + 2 * x + 3
        else:
            d = d + 2 * (x - y) + 5
            y -= 1

        x += 1

    return points

def main():
    print("Midpoint Circle Drawing Algorithm")
    try:
        xc = int(input("Enter center x-coordinate: "))
        yc = int(input("Enter center y-coordinate: "))
        r = int(input("Enter radius: "))

        if r < 0:
            raise ValueError("Radius must be non-negative.")

        points = midpoint_circle(xc, yc, r)
```

```
        # Extract x and y coordinates
        x_coords, y_coords = zip(*points)

        # Plotting the circle
        plt.scatter(x_coords, y_coords, color='purple')
        plt.title("Midpoint Circle Drawing")
        plt.xlabel("X-axis")
        plt.ylabel("Y-axis")
        plt.grid(True)
        plt.gca().set_aspect('equal', adjustable='box')
        plt.show()

    except ValueError as e:
        print("Error:", e)

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

OUTPUT

Midpoint Circle Drawing Algorithm
Enter center x-coordinate: -3
Enter center y-coordinate: 1
Enter radius: 13



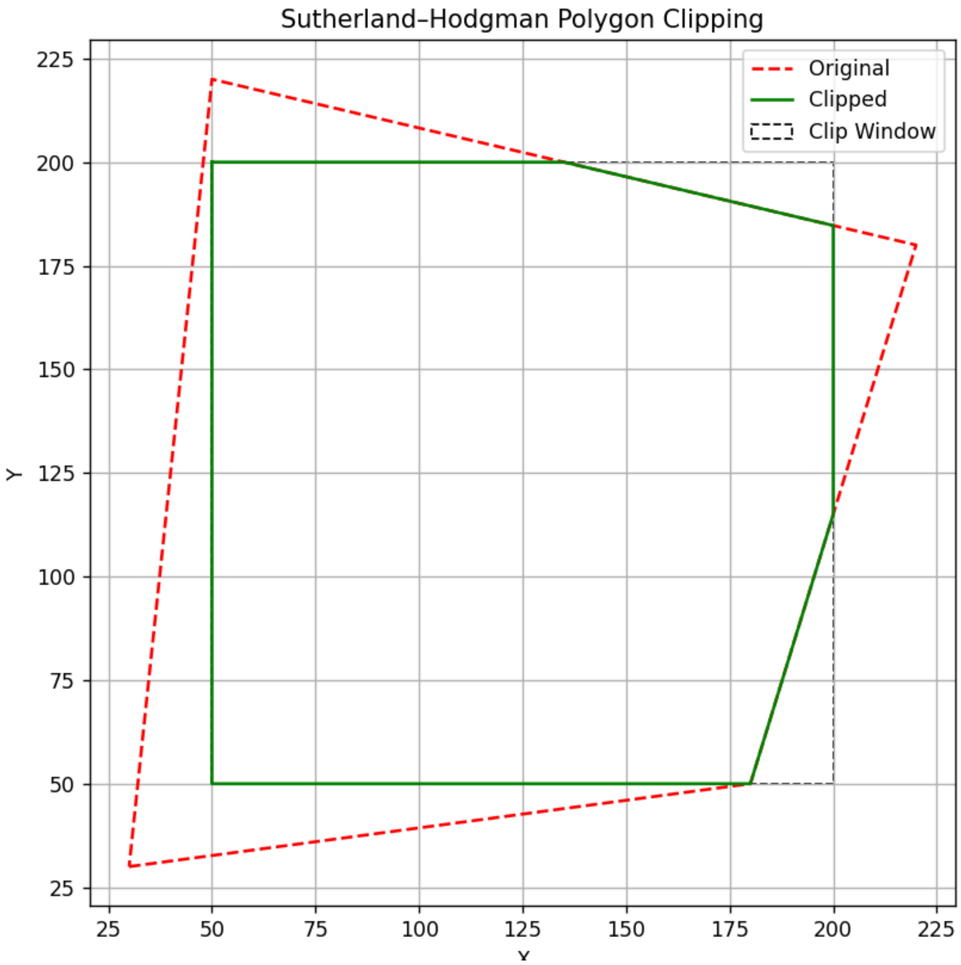
4. **Polygon Clipping using Sutherland-Hodgman Algorithm**

```
import matplotlib.pyplot as plt

def inside(p, edge, rect):
    x, y = p; xmin, xmax, ymin, ymax = rect
    return (x >= xmin if edge == 'LEFT' else
            x <= xmax if edge == 'RIGHT' else
            y >= ymin if edge == 'BOTTOM' else
            y <= ymax)

def intersect(p1, p2, edge, rect):
    x1, y1 = p1; x2, y2 = p2; xmin, xmax, ymin, ymax = rect
    dx, dy = x2 - x1, y2 - y1
    if dx == 0: m = float('inf')
    else: m = dy / dx

    if edge == 'LEFT': return (xmin, y1 + m * (xmin - x1))
    if edge == 'RIGHT': return (xmax, y1 + m * (xmax - x1))
    if edge == 'BOTTOM': return (x1 if m == float('inf') else x1 + (ymin - y1) / m, ymin)
    if edge == 'TOP': return (x1 if m == float('inf') else x1 + (ymax - y1) / m, ymax)

def sutherland_hodgman(polygon, rect):
    for edge in ['LEFT', 'RIGHT', 'BOTTOM', 'TOP']:
        output = []
        s = polygon[-1]
        for e in polygon:
            if inside(e, edge, rect):
                if not inside(s, edge, rect): output.append(intersect(s, e, edge, rect))
                output.append(e)
            elif inside(s, edge, rect):
                output.append(intersect(s, e, edge, rect))
            s = e
        polygon = output
    return polygon

def main():
    print("Sutherland–Hodgman Polygon Clipping")
    n = int(input("Enter number of vertices: "))
    polygon = [tuple(map(float, input(f"Enter x[{i+1}], y[{i+1}]: ").split())) for i in range(n)]
    rect = tuple(map(float, input("Enter xmin, xmax, ymin, ymax: ").split()))
    clipped = sutherland_hodgman(polygon, rect)

    fig, ax = plt.subplots()
    ax.plot(*zip(*(polygon + [polygon[0]])), 'r--', label='Original')
    if clipped:
        ax.plot(*zip(*(clipped + [clipped[0]])), 'g-', label='Clipped')
```

```
    ax.add_patch(plt.Rectangle((rect[0], rect[2]), rect[1]-rect[0], rect[3]-rect[2], edgecolor='black', fill=False, linestyle='--', label='Clip Window'))
    plt.title("Sutherland–Hodgman Polygon Clipping")
    plt.xlabel("X"); plt.ylabel("Y"); plt.legend(); plt.grid(True)
    ax.set_aspect('equal', adjustable='box'); plt.show()

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

OUTPUT

```
Sutherland–Hodgman Polygon Clipping
Enter number of vertices: 4
Enter x[1], y[1]: 30 30
Enter x[2], y[2]: 180 50
Enter x[3], y[3]: 220 180
Enter x[4], y[4]: 50 220
Enter xmin, xmax, ymin, ymax: 50 200 50 200
```



5. **WAP to apply various 2D Transformations on a 2D object.**

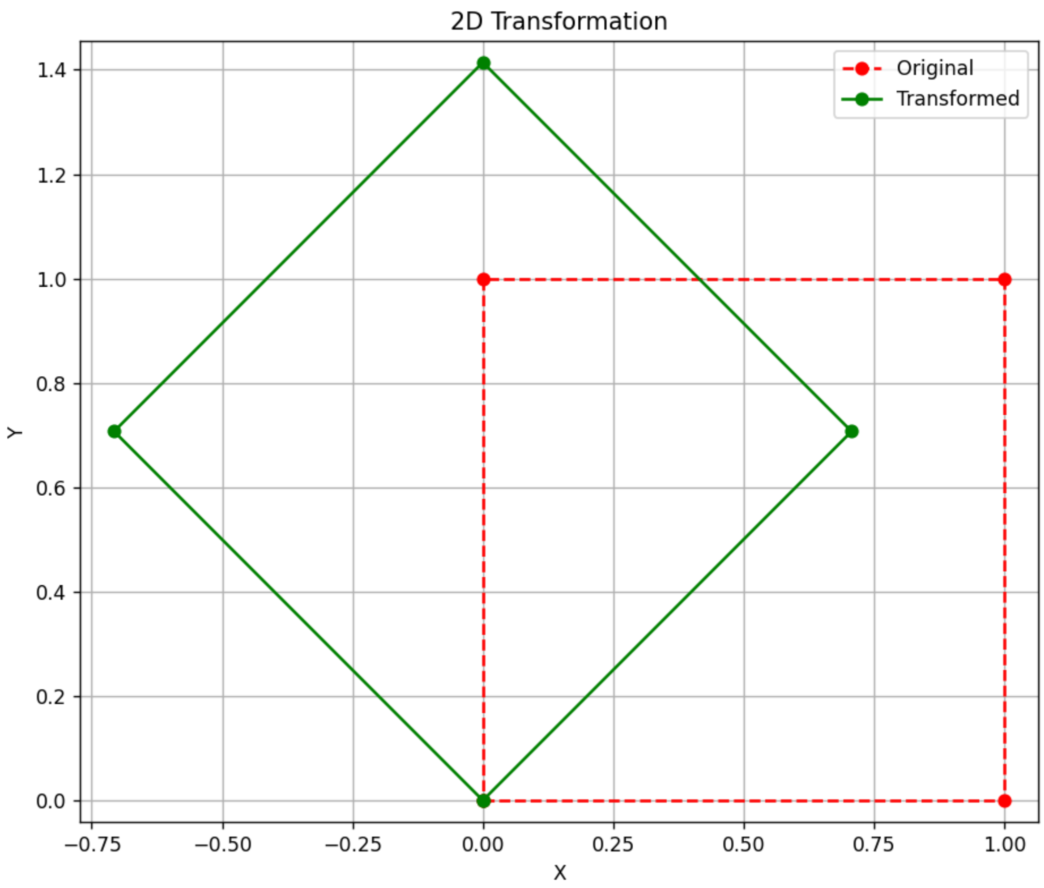
```
import numpy as np
import matplotlib.pyplot as plt

def get_polygon():
    n = int(input("Enter number of vertices: "))
    return np.array([[*map(float, input(f"x[{i+1}] y[{i+1}]: ").split()), 1] for i in range(n)])

def transform(p, mat): return p @ mat.T

def matrices():
    return {
        '1': lambda: np.array([[1, 0, float(input("tx: "))], [0, 1, float(input("ty: "))], [0, 0, 1]]),
        '2': lambda: np.array([[float(input("sx: ")), 0, 0], [0, float(input("sy: ")), 0], [0, 0, 1]]),
        '3': lambda: (lambda a: np.array([[np.cos(a), -np.sin(a), 0], [np.sin(a), np.cos(a), 0], [0, 0,
1]]))(np.radians(float(input("Angle (deg): ")))),
        '4': lambda: np.array([[1, float(input("shx: ")), 0], [float(input("shy: ")), 1, 0], [0, 0, 1]]),
        '5': lambda: {'x': [[1, 0, 0], [0, -1, 0], [0, 0, 1]], 'y': [[-1, 0, 0], [0, 1, 0], [0, 0, 1]], 'origin': [[-1, 0, 0], [0, -1, 0],
[0, 0, 1]]}[input("Axis (x/y/origin): ").lower()]
    }

def plot(polygon, transformed):
    def close(p): return np.vstack([p[:, :2], p[0, :2]])
    plt.clf()
    plt.plot(*close(polygon).T, 'ro--', label="Original")
    plt.plot(*close(transformed).T, 'go-', label="Transformed")
    plt.title("2D Transformation"); plt.xlabel("X"); plt.ylabel("Y")
    plt.legend(); plt.axis('equal'); plt.grid(True)
    plt.pause(0.1)

def main():
    print("2D Transformations using Homogeneous Coordinates")
    polygon = get_polygon()
    transformed = polygon.copy()
    plt.ion(); plt.figure()

    while True:
        print("\n1: Translate  2: Scale  3: Rotate  4: Shear  5: Reflect  q: Quit")
        choice = input("Choose transformation: ")
        if choice == 'q': break

        if choice in matrices():
            try:
                mat = np.array(matrices()[choice](), dtype=float)
                transformed = transform(transformed, mat)
                plot(polygon, transformed)
            except: print("Invalid input.")
```

```
        else:
            print("Invalid choice.")
    plt.ioff(); plt.show()

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

OUTPUT

2D Transformations using Homogeneous Coordinates
Enter number of vertices: 4
x[1] y[1]: 0 0
x[2] y[2]: 1 0
x[3] y[3]: 1 1
x[4] y[4]: 0 1

1: Translate
2: Scale
3: Rotate
4: Shear
5: Reflect
q: Quit
Choose transformation: 3
Angle (deg): 45



6. **WAP to draw Bezier and Hermite Curve.**

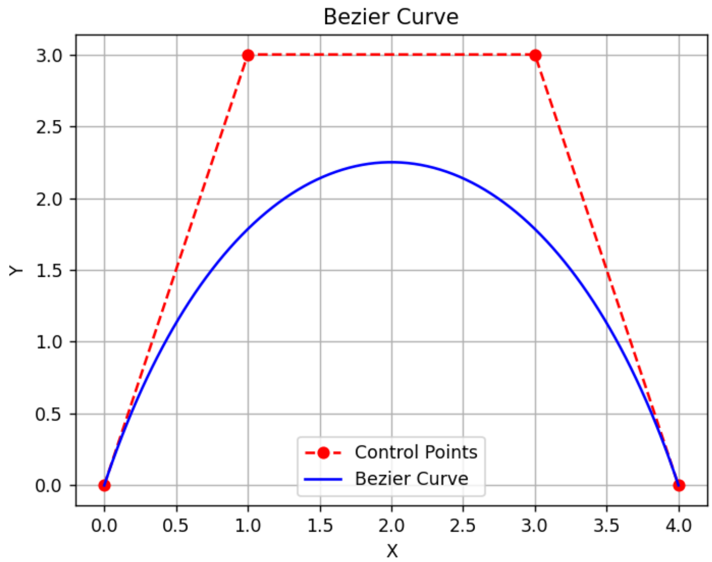
```
import numpy as np
import matplotlib.pyplot as plt

def bezier(P0, P1, P2, P3, t):
    return ((1 - t)**3)[:, None] * P0 + (3 * (1 - t)**2 * t)[:, None] * P1 + (3 * (1 - t) * t**2)[:, None] * P2 +
(t**3)[:, None] * P3

def hermite(P0, P1, T0, T1, t):
    h1, h2 = (2*t**3 - 3*t**2 + 1)[:, None], (t**3 - 2*t**2 + t)[:, None]
    h3, h4 = (-2*t**3 + 3*t**2)[:, None], (t**3 - t**2)[:, None]
    return h1 * P0 + h2 * T0 + h3 * P1 + h4 * T1

def get_points(n, prompt): return [np.array(list(map(float, input(f"{prompt} {i+1}: ").split()))) for i in
range(n)]

def plot_curve(ctrl, curve, name):
    plt.plot(*ctrl.T, 'ro--', label='Control Points')
    plt.plot(*curve.T, 'b-', label=f'{name} Curve')
    plt.title(f'{name} Curve'); plt.xlabel('X'); plt.ylabel('Y')
    plt.legend(); plt.grid(True); plt.axis('equal'); plt.show()

def main():
    print("1. Bezier Curve\n2. Hermite Curve")
    choice = input("Choose (1 or 2): "); t = np.linspace(0, 1, 100)

    if choice == '1':
        P = get_points(4, "Bezier Point")
        curve = bezier(*P, t); plot_curve(np.vstack(P), curve, "Bezier")

    elif choice == '2':
        P0, P1 = get_points(2, "Endpoint")
        T0, T1 = get_points(2, "Tangent")
        curve = hermite(P0, P1, T0, T1, t)
        plot_curve(np.vstack([P0, P1]), curve, "Hermite")
    else:
        print("Invalid choice.")

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

OUTPUT

Bezier Curve

1. Bezier Curve
2. Hermite Curve

Choose (1 or 2): 1
Bezier Point 1: 0 0
Bezier Point 2: 1 3
Bezier Point 3: 3 3
Bezier Point 4: 4 0



Hermite Curve

1. Bezier Curve
2. Hermite Curve

Choose (1 or 2): 2
Endpoint 1: 0 0
Endpoint 2: 4 0
Tangent 1: 1 3
Tangent 2: 1 -3