Pabna University of Science and Technology

Department of
Computer Science and Engineering

Faculty of

Engineering and Technology

Lab Report On

Course Code: CSE 4204.

Course Title: Computer Graphics Sessional.

Submitted by:

Name: Md. Habibul Islam

Roll: 200124

Session: 2019-20.

4th year 2nd semester

Dept. Of CSE, PUST.

Submitted to:

Md. Shafiul Azam

Associate Professor,

Dept. Of CSE, PUST.

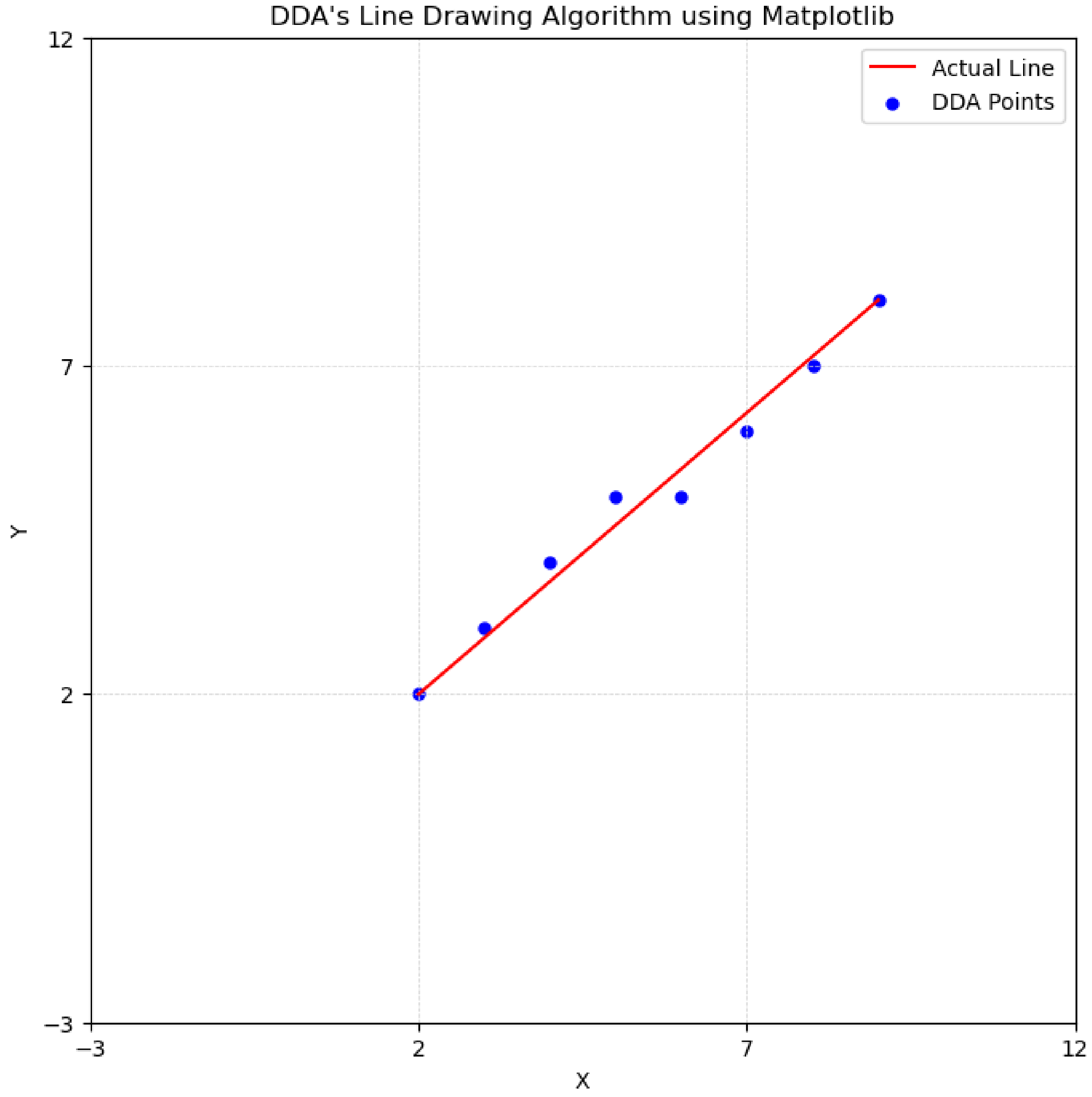
Date of Submission: May 26, 2025.

Lab Index

SL No.	Problem Statement	Page’s
1	DDA Line drawing algorithm	
2	Bresenham Line drawing algorithm	
3	Mid-point Circle drawing algorithm	
4	Mid-point Ellipse drawing algorithm	
5	Translation	
6	Rotation	
7	Scaling	
8	Reflection	
9	Shearing	
10	Point clipping algorithm	
11	Line clipping algorithm	
12	Polygon clipping algorithm	
13	Boundary filling algorithm	
14	Flood filling algorithm	

Input points: $(2,2)$ $(9,8)$

Output:



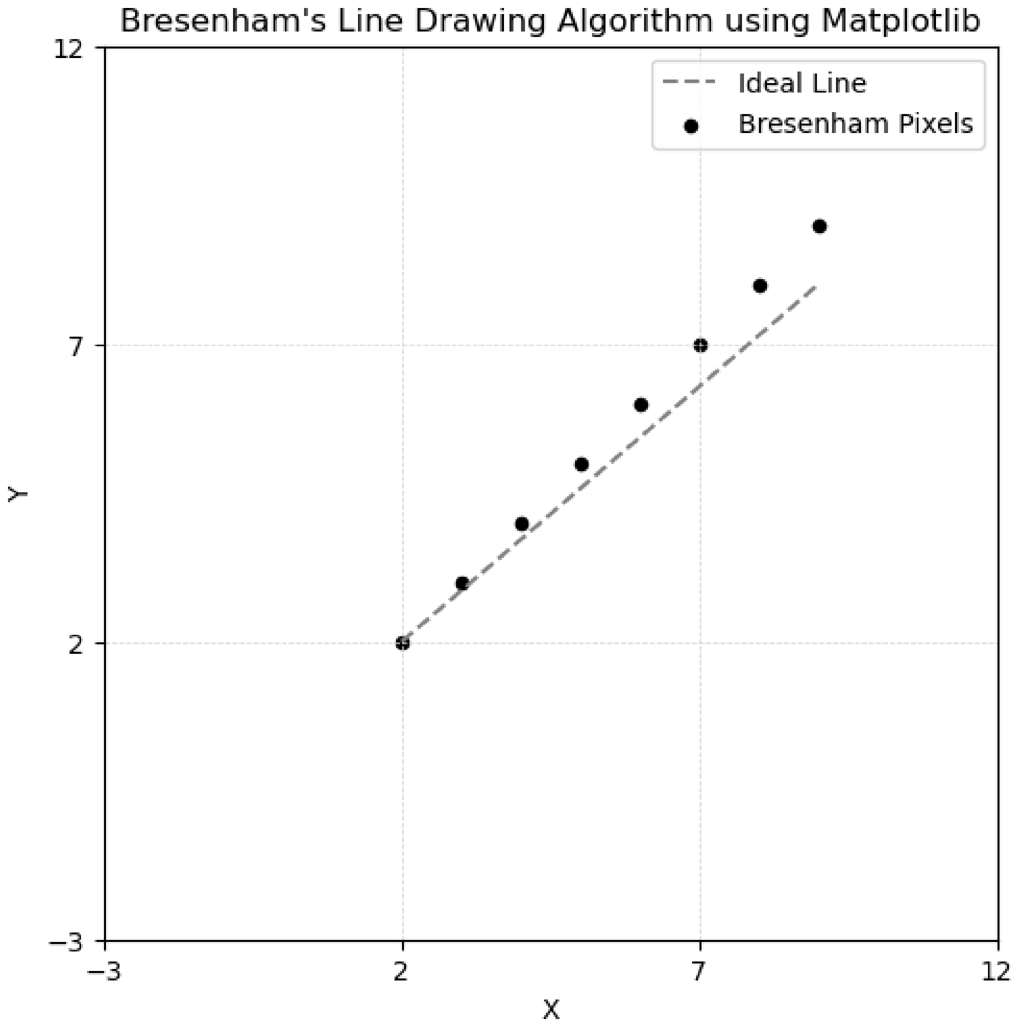
Code:

```
import matplotlib.pyplot as plt
def bresenham_line(x0, y0, x1, y1):
    if x0 > x1:  # Ensure drawing left to right
        x0, y0, x1, y1 = x1, y1, x0, y0
    points = [(x0, y0)]
    dx = x1 - x0
    dy = y1 - y0
    two_dy = 2 * abs(dy)
    two_dy_minus_dx = 2 * abs(dy) - dx
    p = two_dy - dx
    x, y = x0, y0
    y_step = 1 if dy > 0 else -1  # handle upward/downward slope
    for _ in range(dx):
        x += 1
        if p < 0:
            p += two_dy
        else:
            y += y_step
            p += two_dy_minus_dx
        points.append((x, y))
    return points

x1, y1 = int(input("Enter start x: ")), int(input("Enter start y: "))
x2, y2 = int(input("Enter end x: ")), int(input("Enter end y: "))
bresenham_points = bresenham_line(x1, y1, x2, y2)
x_coords, y_coords = zip(*bresenham_points)

plt.figure(figsize=(6, 7))
plt.plot([x1, x2], [y1, y2], label="Ideal Line", color="gray", linestyle="--")
plt.scatter(x_coords, y_coords, c="black", label="Bresenham Pixels", s=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.grid(True, which='both', color='lightgray', linestyle='--', linewidth=0.5)
plt.xticks(range(min(x_coords)-5, max(x_coords)+5, 5))
plt.yticks(range(min(y_coords)-5, max(y_coords)+5, 5))
plt.legend()
plt.title("Bresenham's Line Drawing Algorithm using Matplotlib")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

Input points:

```
Enter start x: 2
Enter start y: 2
Enter end x: 9
Enter end y: 8
```

Output:



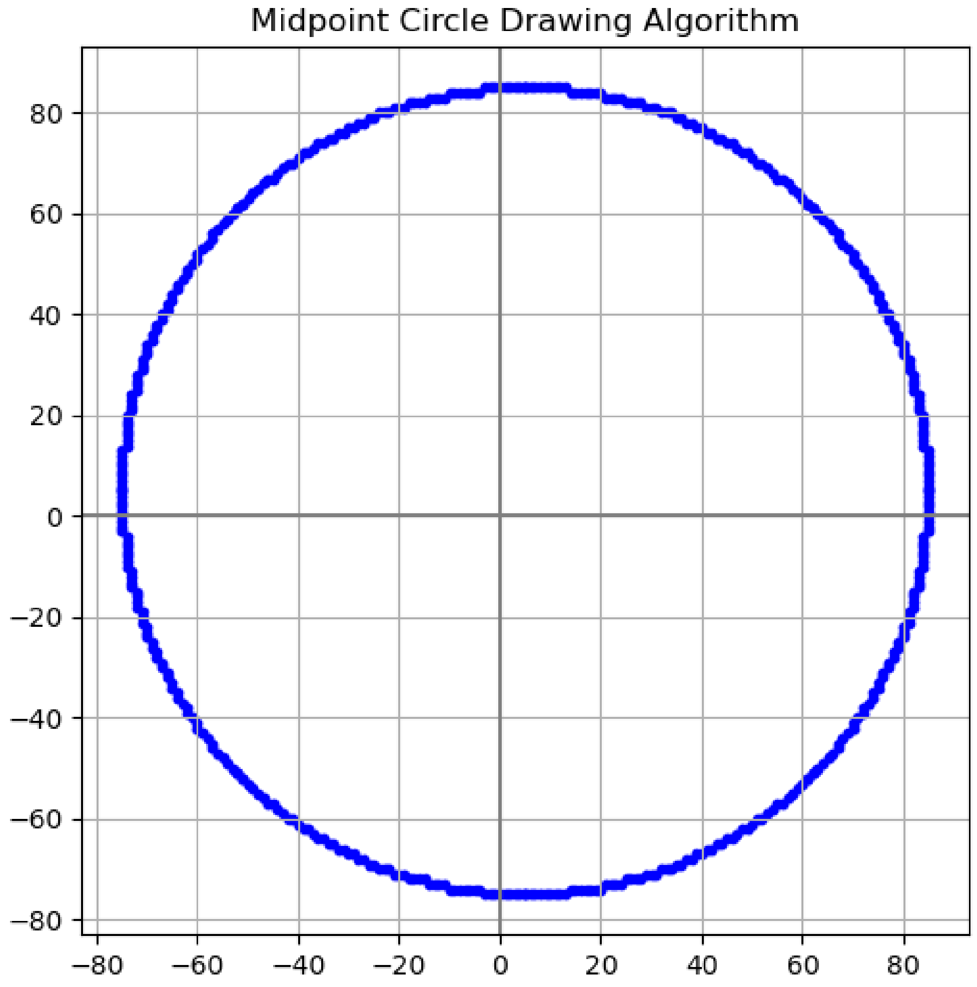
Code:

```
import matplotlib.pyplot as plt

def draw_circle_midpoint(xc, yc, radius):
    x = 0
    y = radius
    p = 1 - radius
    x_points = []
    y_points = []
    def plot_circle_points(xc, yc, x, y):
        points = [
            (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) ]
        for px, py in points:
            x_points.append(px)
            y_points.append(py)
    plot_circle_points(xc, yc, x, y)
    while x < y:
        x += 1
        if p < 0:
            p += 2 * x + 1
        else:
            y -= 1
            p += 2 * (x - y) + 1
        plot_circle_points(xc, yc, x, y)
    plt.figure(figsize=(6, 6))
    plt.scatter(x_points, y_points, color="blue", s=10)
    plt.axhline(0, color='gray')  # X-axis
    plt.axvline(0, color='gray')  # Y-axis
    plt.title("Midpoint Circle Drawing Algorithm")
    plt.gca().set_aspect('equal', adjustable='box')
    plt.grid(True)
    plt.show()

center_x = int(input("Enter center x: "))
center_y = int(input("Enter center y: "))
radius = int(input("Enter radius: "))

draw_circle_midpoint(center_x, center_y, radius)
```

Input points:

```
Enter center x: 5
Enter center y: 5
Enter radius: 80
```

Output:



Code:

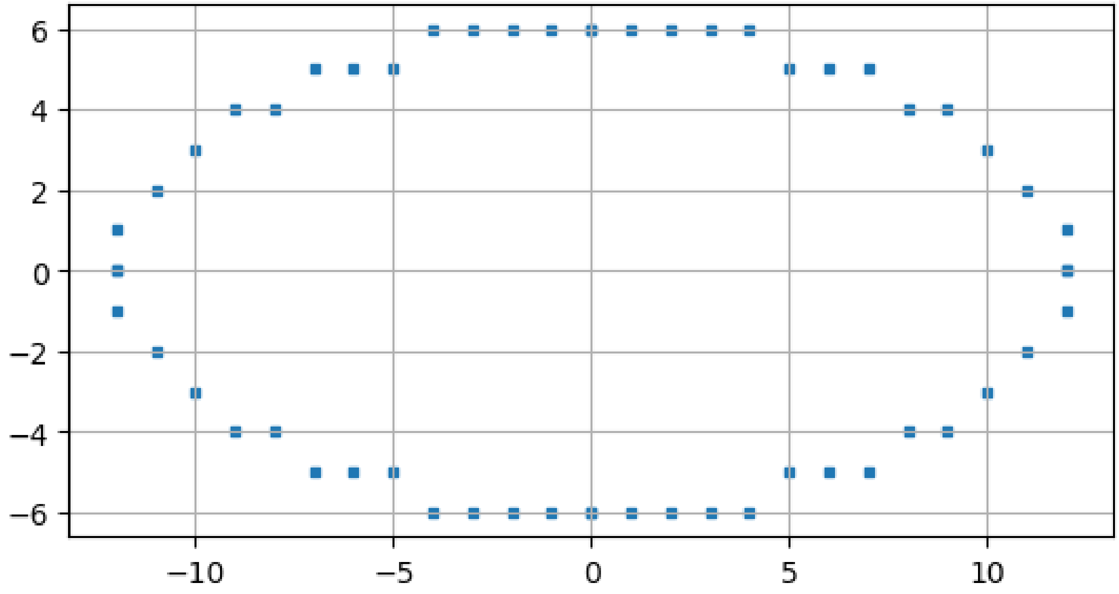
```
import matplotlib.pyplot as plt
def midpoint_ellipse(center, radius_x, radius_y):
    activated_pixels = []
    xc, yc = center
    x = 0
    y = radius_y
    rx_sq = radius_x * radius_x
    ry_sq = radius_y * radius_y
    two_rx_sq = 2 * rx_sq
    two_ry_sq = 2 * ry_sq
    decision = ry_sq - (rx_sq * radius_y) + (0.25 * rx_sq)
    del_x = two_ry_sq * x
    del_y = two_rx_sq * y
    while del_x < del_y:
        activated_pixels.extend([(xc+x, yc+y), (xc+x, yc-y), (xc-x, yc+y), (xc-x, yc-y)])
        if decision < 0:
            x += 1
            del_x += two_ry_sq
            decision += del_x + ry_sq
        else:
            x += 1
            y -= 1
            del_x += two_ry_sq
            del_y -= two_rx_sq
            decision += del_x - del_y + ry_sq
    decision = ry_sq * (x + 0.5)**2 + rx_sq * (y - 1)**2 - rx_sq * ry_sq
    while y >= 0:
        activated_pixels.extend([(xc+x, yc+y), (xc+x, yc-y), (xc-x, yc+y), (xc-x, yc-y)])
        if decision > 0:
            y -= 1
            del_y -= two_rx_sq
            decision += rx_sq - del_y
        else:
            y -= 1
            x += 1
            del_y -= two_rx_sq
            del_x += two_ry_sq
            decision += rx_sq - del_y + del_x
    return activated_pixels
```

```
def plot_points(activated_pixels):
    if not activated_pixels:
        print("No pixels to plot")
        return
    x_points, y_points = zip(*activated_pixels)
    plt.scatter(x_points, y_points, marker='s', s=10)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.grid(True)
    plt.show()
xc = int(input("Enter xc: "))
yc = int(input("Enter yc: "))
radius_x = int(input("Enter major axis: "))
radius_y = int(input("Enter minor axis: "))
activated_pixels = midpoint_ellipse((xc, yc), radius_x, radius_y)
plot_points(activated_pixels)
```

Input points:

```
Enter center x: 5
Enter center y: 5
Enter major-axis: 12
Enter minor-axis: 6
```

Output:



Code:

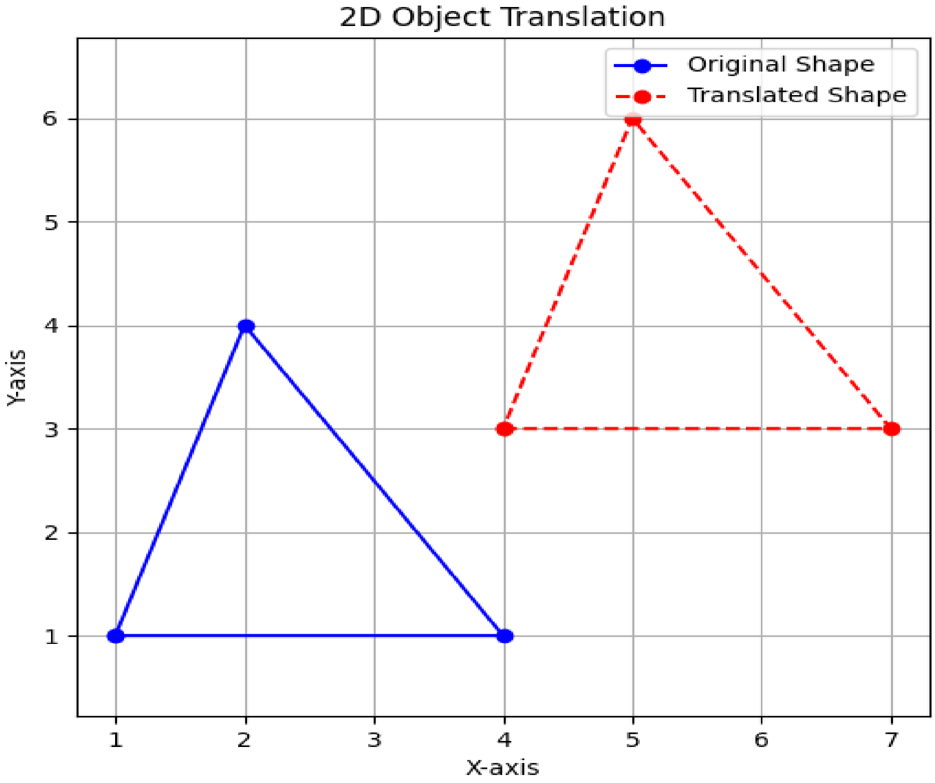
```
import matplotlib.pyplot as plt
def translate_object(points, tx, ty):
    translated_points = []
    for x, y in points:
        new_x = x + tx
        new_y = y + ty
        translated_points.append((new_x, new_y))
    return translated_points
print('Input:')
n = int(input("Enter number of points: "))
original_points = []
for i in range(n):
    x, y = map(int, input(f"Enter coordinates for point {i+1} (x y): ").split())
    original_points.append((x, y))

tx = int(input("Enter translation in x (tx): "))
ty = int(input("Enter translation in y (ty): "))
translated_points = translate_object(original_points, tx, ty)

print('Output:\n')
original_points.append(original_points[0])
translated_points.append(translated_points[0])

print("Original Points:", original_points)
print("Translated Points:", translated_points)

ox, oy = zip(*original_points)
tx, ty = zip(*translated_points)

plt.figure(figsize=(6, 6))
plt.plot(ox, oy, 'bo-', label='Original Shape')
plt.plot(tx, ty, 'ro--', label='Translated Shape')
plt.title("2D Object Translation")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```

Input:

```
Enter number of points: 3
Enter coordinates for point 1 (x y): 1 1
Enter coordinates for point 2 (x y): 4 1
Enter coordinates for point 3 (x y): 2 4
Enter translation in x (tx): 3
Enter translation in y (ty): 2
```

Output:

```
Original Points: [(1, 1), (4, 1), (2, 4), (1, 1)]
Translated Points: [(4, 3), (7, 3), (5, 6), (4, 3)]
```



Code:

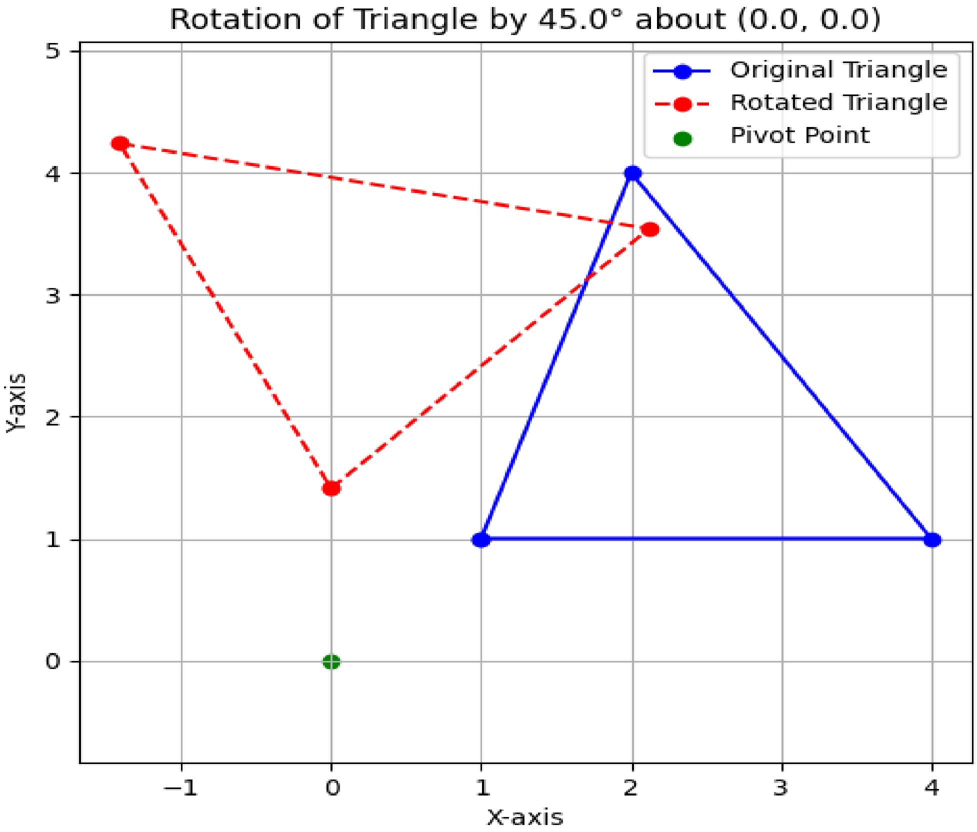
```
import math
import matplotlib.pyplot as plt
def rotate_object(points, angle_degrees, pivot=(0, 0)):
    angle_rad = math.radians(angle_degrees)
    cos_theta = math.cos(angle_rad)
    sin_theta = math.sin(angle_rad)
    h, k = pivot
    rotated_points = []
    for x, y in points:
        x -= h , y -= k
        x_new = x * cos_theta - y * sin_theta
        y_new = x * sin_theta + y * cos_theta
        x_rotated = x_new + h ,  y_rotated = y_new + k
        rotated_points.append((round(x_rotated, 2), round(y_rotated, 2)))
    return rotated_points
n = int(input("Enter number of vertices (should be 3 for a triangle):"))
original_points = []
for i in range(0,n):
    x, y = map(float, input(f"Enter coordinates for point {i+1} (x y): ").split())
    original_points.append((x, y))
angle = float(input("Enter rotation angle (in degrees): "))
pivot_x, pivot_y = map(float, input("Enter pivot point (x y): ").split())
pivot_point = (pivot_x, pivot_y)
rotated_points = rotate_object(original_points, angle, pivot_point)
print("\nOriginal Points:", original_points)
print("Rotated Points:", rotated_points)
original_points.append(original_points[0])
rotated_points.append(rotated_points[0])
ox, oy = zip(*original_points)
rx, ry = zip(*rotated_points)
plt.figure(figsize=(6, 6))
plt.plot(ox, oy, 'bo-', label='Original Triangle')
plt.plot(rx, ry, 'ro--', label='Rotated Triangle')
plt.scatter(*pivot_point, color='green', label='Pivot Point')
plt.title(f"Rotation of Triangle by {angle}° about {pivot_point}")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

Input:

```
Enter number of vertices (should be 3 for a triangle):3
Enter coordinates for point 1 (x y): 1 1
Enter coordinates for point 2 (x y): 4 1
Enter coordinates for point 3 (x y): 2 4
Enter rotation angle (in degrees): 45
Enter pivot point (x y): 0 0
```

Output:

```
Original Points: [(1.0, 1.0), (4.0, 1.0), (2.0, 4.0)]
Rotated Points: [(0.0, 1.41), (2.12, 3.54), (-1.41, 4.24)]
```



Code:

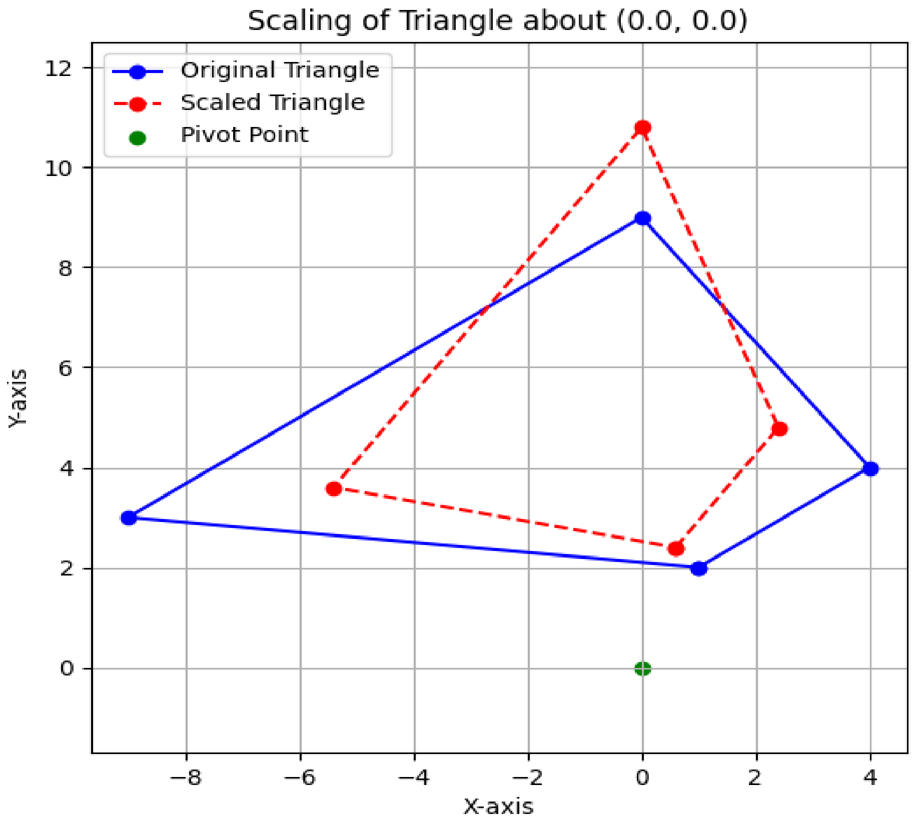
```
import matplotlib.pyplot as plt
def scale_object(points, sx, sy, pivot=(0, 0)):
    h, k = pivot
    scaled_points = []
    for x, y in points:
        x_scaled = h + (x - h) * sx
        y_scaled = k + (y - k) * sy
        scaled_points.append((round(x_scaled, 2), round(y_scaled, 2)))
    return scaled_points

n = int(input("Enter number of vertices (should be 3 for a triangle): "))
original_points = []
for i in range(0,n):
    x, y = map(float, input(f"Enter coordinates for point {i+1} (x y): ").split())
    original_points.append((x, y))
sx = float(input("Enter scaling factor in x-direction (sx): "))
sy = float(input("Enter scaling factor in y-direction (sy): "))

pivot_x, pivot_y = map(float, input("Enter pivot point (x y): ").split())
pivot_point = (pivot_x, pivot_y)
scaled_points = scale_object(original_points, sx, sy, pivot_point)
print("\nOriginal Points:", original_points)
print("Scaled Points:", scaled_points)
original_points.append(original_points[0])
scaled_points.append(scaled_points[0])
ox, oy = zip(*original_points)
sx_, sy_ = zip(*scaled_points)

plt.figure(figsize=(6, 6))
plt.plot(ox, oy, 'bo-', label='Original Triangle')
plt.plot(sx_, sy_, 'ro--', label='Scaled Triangle')
plt.scatter(*pivot_point, color='green', label='Pivot Point')
plt.title(f"Scaling of Triangle about {pivot_point}")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```

Input:

```
Enter number of vertices (should be 3 for a triangle): 4
Enter coordinates for point 1 (x y): 1 2
Enter coordinates for point 2 (x y): 4 4
Enter coordinates for point 3 (x y): 0 9
Enter coordinates for point 4 (x y): -9 3
Enter scaling factor in x-direction (sx): .6
Enter scaling factor in y-direction (sy): 1.2
Enter pivot point (x y): 0 0
```

Output:

```
Original Points: [(1.0, 2.0), (4.0, 4.0), (0.0, 9.0), (-9.0, 3.0)]
Scaled Points: [(0.6, 2.4), (2.4, 4.8), (0.0, 10.8), (-5.4, 3.6)]
```



Code:

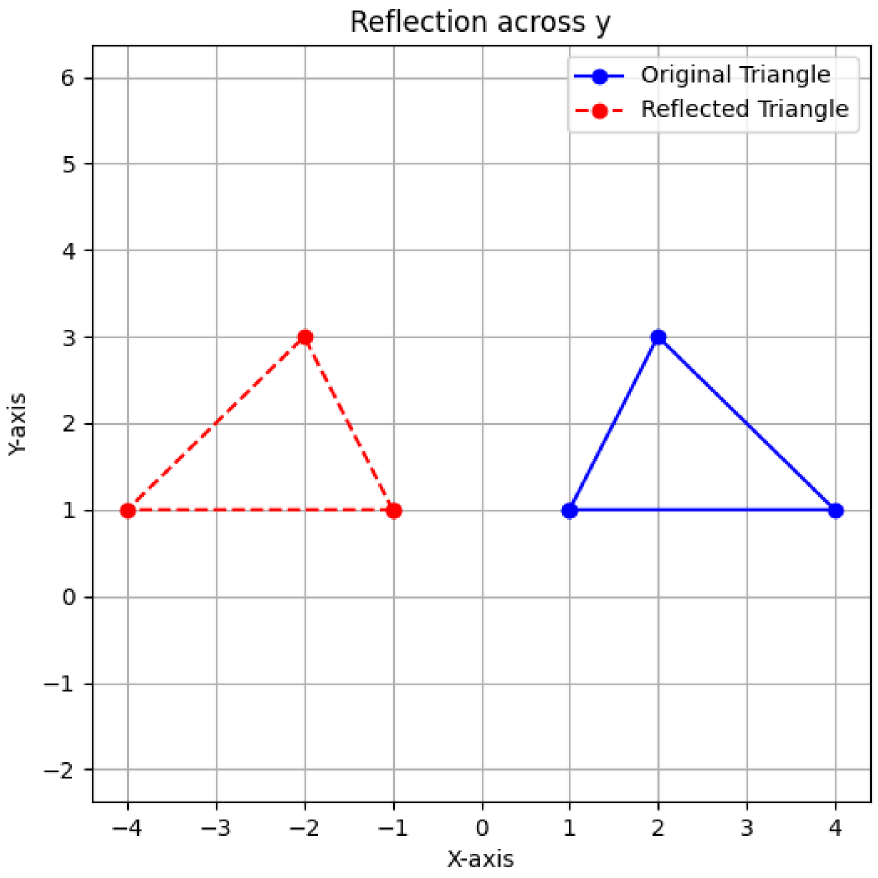
```
import matplotlib.pyplot as plt
def reflect_object(points, axis='x'):
    reflected_points = []
    for x, y in points:
        if axis == 'x':
            reflected_points.append((x, -y))
        elif axis == 'y':
            reflected_points.append((-x, y))
        elif axis == 'origin':
            reflected_points.append((-x, -y))
        elif axis == 'y=x':
            reflected_points.append((y, x))
        else:
            raise ValueError("Invalid reflection axis. Choose from 'x', 'y', 'origin', 'y=x'.")
    return reflected_points
n = int(input("Enter number of vertices (should be 3 for a triangle): "))
original_points = []
for i in range(0,n):
    x, y = map(float, input(f"Enter coordinates for point {i+1} (x y): ").split())
    original_points.append((x, y))
print("\nChoose axis of reflection (x, y, origin, y=x):")
axis = input("Enter axis: ").strip().lower()
reflected_points = reflect_object(original_points, axis)
print("\nOriginal Points:", original_points)
print("Reflected Points:", reflected_points)
original_points.append(original_points[0])
reflected_points.append(reflected_points[0])
ox, oy = zip(*original_points)
rx, ry = zip(*reflected_points)
plt.figure(figsize=(6, 6))
plt.plot(ox, oy, 'bo-', label='Original Triangle')
plt.plot(rx, ry, 'ro--', label='Reflected Triangle')
plt.title(f"Reflection across {axis}")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```

Input:

```
Enter number of vertices (should be 3 for a triangle): 3
Enter coordinates for point 1 (x y): 1 1
Enter coordinates for point 2 (x y): 4 1
Enter coordinates for point 3 (x y): 2 3

Choose axis of reflection (x, y, origin, y=x):
Enter axis: y
```

Output:

```
Original Points: [(1.0, 1.0), (4.0, 1.0), (2.0, 3.0)]
Reflected Points: [(-1.0, 1.0), (-4.0, 1.0), (-2.0, 3.0)]]
```



Code:

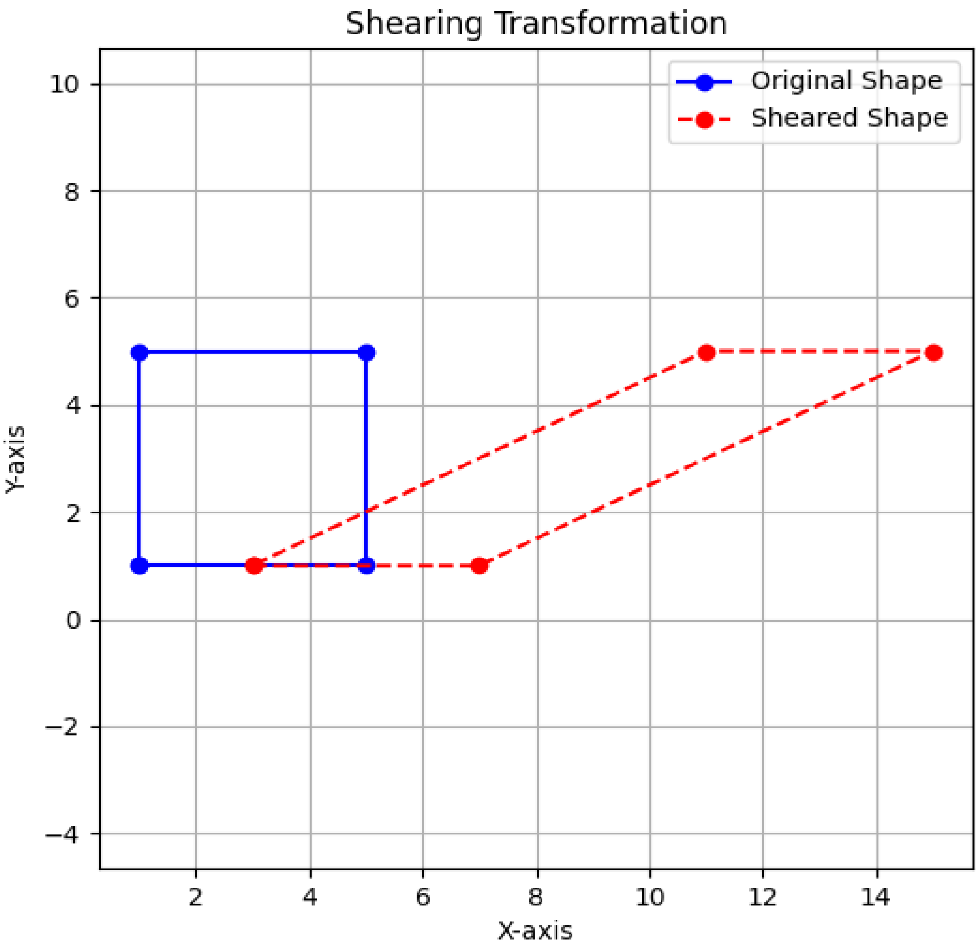
```
import matplotlib.pyplot as plt

def shear_object(points, shx=0, shy=0):
    sheared_points = []
    for x, y in points:
        x_sheared = x + shx * y
        y_sheared = y + shy * x
        sheared_points.append((round(x_sheared, 2), round(y_sheared, 2)))
    return sheared_points
n = int(input("Enter number of vertices: "))
original_points = []
print("Enter coordinates:")
for i in range(n):
    x, y = map(float, input(f"Point {i+1} (x y): ").split())
    original_points.append((x, y))
shx = float(input("Enter shear factor in X-direction (shx): "))
shy = float(input("Enter shear factor in Y-direction (shy): "))
sheared_points = shear_object(original_points, shx, shy)
print("\nOriginal Points:", original_points)
print("Sheared Points:", sheared_points)
original_points.append(original_points[0])
sheared_points.append(sheared_points[0])

ox, oy = zip(*original_points)
sx, sy = zip(*sheared_points)

plt.figure(figsize=(6, 6))
plt.plot(ox, oy, 'bo-', label='Original Shape')
plt.plot(sx, sy, 'ro--', label='Sheared Shape')
plt.title("Shearing Transformation")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.axis('equal')
plt.legend()
plt.show()
```

Input:

```
Enter number of vertices: 4
Enter coordinates:
Point 1 (x y): 1 1
Point 2 (x y): 5 1
Point 3 (x y): 5 5
Point 4 (x y): 1 5
Enter shear factor in X-direction (shx): 2
Enter shear factor in Y-direction (shy): 0
```

Output:

```
Original Points: [(1.0, 1.0), (5.0, 1.0), (5.0, 5.0), (1.0, 5.0)]
Sheared Points: [(3.0, 1.0), (7.0, 1.0), (15.0, 5.0), (11.0, 5.0)]
```



Code:

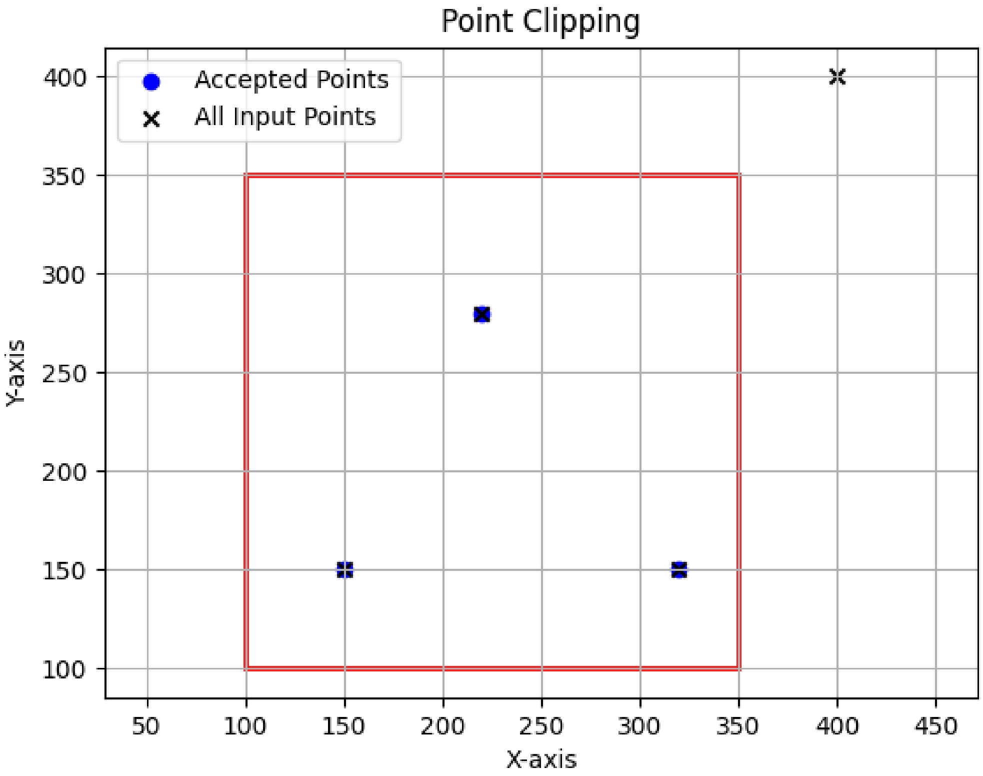
```
import matplotlib.pyplot as plt

def point_clipping(points, xmin, ymin, xmax, ymax):
    inside_points = []
    for x, y in points:
        if xmin <= x <= xmax and ymin <= y <= ymax:
            inside_points.append((x, y))
    return inside_points

n = int(input("Enter number of points: "))
points = []
for i in range(n):
    x, y = map(float, input(f"Point {i+1} (x y): ").split())
    points.append((x, y))

xmin, ymin, xmax, ymax = map(float, input(f"Enter window (xmin, ymin, xmax, ymax): ").split())
accepted_points = point_clipping(points, xmin, ymin, xmax, ymax)
fig, ax = plt.subplots()
rect = plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin, edgecolor='red', facecolor='none',
linewidth=2)
ax.add_patch(rect)
if accepted_points:
    x_vals, y_vals = zip(*accepted_points)
    plt.scatter(x_vals, y_vals, color='blue', label='Accepted Points')
x_all, y_all = zip(*points)
plt.scatter(x_all, y_all, color='black', marker='x', label='All Input Points')
plt.title('Point Clipping')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```

Input:

```
Enter number of points: 4
Point 1 (x y): 150 150
Point 2 (x y): 220 280
Point 3 (x y): 320 150
Point 4 (x y): 400 400
Enter window (xmin, ymin, xmax, ymax): 100 100 350 350
```

Output:

```
Accepted Points: [(150.0, 150.0), (220.0, 280.0), (320.0, 150.0)]
Rejected Points: [(400.0, 400.0)]
```



Code:

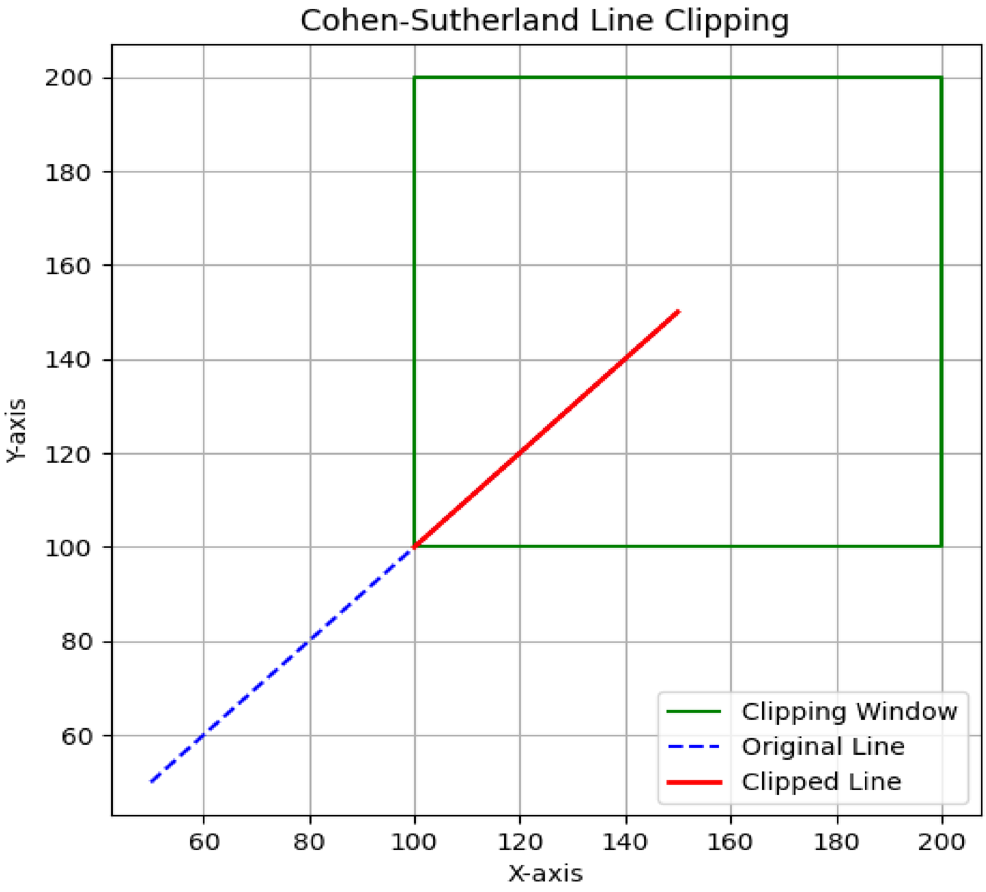
```
import matplotlib.pyplot as plt

def point_clipping(points, xmin, ymin, xmax, ymax):
    inside_points = []
    for x, y in points:
        if xmin <= x <= xmax and ymin <= y <= ymax:
            inside_points.append((x, y))
    return inside_points

n = int(input("Enter number of points: "))
points = []
for i in range(n):
    x, y = map(float, input(f"Point {i+1} (x y): ").split())
    points.append((x, y))

xmin, ymin, xmax, ymax = map(float, input(f"Enter window (xmin, ymin, xmax, ymax): ").split())
accepted_points = point_clipping(points, xmin, ymin, xmax, ymax)
fig, ax = plt.subplots()
rect = plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin, edgecolor='red', facecolor='none', linewidth=2)
ax.add_patch(rect)
if accepted_points:
    x_vals, y_vals = zip(*accepted_points)
    plt.scatter(x_vals, y_vals, color='blue', label='Accepted Points')
x_all, y_all = zip(*points)
plt.scatter(x_all, y_all, color='black', marker='x', label='All Input Points')
plt.title('Point Clipping')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```

Input:

```
Enter the coordinates of the line:
Enter (x1 y1): 50 50
Enter (x2 y2): 150 150
Enter the clipping window (xmin ymin xmax ymax): 100 100 200 200
```

Output:

```
Original Line Segment: (50.00, 50.00) to (150.00, 150.00)
Clipped Line Segment: (100.00, 100.00) to (150.00, 150.00)
```



Code:

```
import matplotlib.pyplot as plt

def inside(p, edge, boundary):
    x, y = p
    if edge == 'LEFT': return x >= boundary
    if edge == 'RIGHT': return x <= boundary
    if edge == 'BOTTOM': return y >= boundary
    if edge == 'TOP': return y <= boundary

def intersect(p1, p2, edge, boundary):
    x1, y1 = p1
    x2, y2 = p2
    if edge in ['LEFT', 'RIGHT']:
        x = boundary
        y = y1 + (y2 - y1) * (boundary - x1) / (x2 - x1)
    else:
        y = boundary
        x = x1 + (x2 - x1) * (boundary - y1) / (y2 - y1)
    return (round(x, 2), round(y, 2))

def clip_polygon(polygon, edge, boundary):
    clipped = []
    n = len(polygon)
    for i in range(n):
        curr = polygon[i]
        prev = polygon[i - 1]
        if inside(curr, edge, boundary):
            if inside(prev, edge, boundary):
                clipped.append(curr)
            else:
                clipped.append(intersect(prev, curr, edge, boundary))
                clipped.append(curr)
        elif inside(prev, edge, boundary):
            clipped.append(intersect(prev, curr, edge, boundary))
    return clipped
```

```
def sutherland_hodgman(polygon, xmin, ymin, xmax, ymax):
    for edge, boundary in [('LEFT', xmin), ('RIGHT', xmax),
                           ('BOTTOM', ymin), ('TOP', ymax)]:
        polygon = clip_polygon(polygon, edge, boundary)
    return polygon
n = int(input("Enter number of vertices in the polygon: "))
polygon = []
for i in range(n):
    x, y = map(float, input(f"Enter coordinates for vertex {i+1} (x y): ").split())
    polygon.append((x, y))

print("Enter clipping window (xmin ymin xmax ymax):")
xmin, ymin, xmax, ymax = map(float, input().split())
clipped_polygon = sutherland_hodgman(polygon, xmin, ymin, xmax, ymax)
print("\nOriginal Polygon:", polygon)
print("Clipped Polygon:", clipped_polygon)
def draw_polygon(points, color, label):
    x_coords, y_coords = zip(*(points + [points[0]]))  # Close the polygon
    plt.plot(x_coords, y_coords, color, label=label)
plt.figure(figsize=(6, 6))
plt.plot([xmin, xmax, xmax, xmin, xmin],
         [ymin, ymin, ymax, ymax, ymin], 'g-', label="Clipping Window")
draw_polygon(polygon, 'b--', 'Original Polygon')

# Clipped polygon
if clipped_polygon:
    draw_polygon(clipped_polygon, 'r-', 'Clipped Polygon')

plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.legend()
plt.title("Sutherland-Hodgman Polygon Clipping")
plt.axis('equal')
plt.show()
```

Input:

```
Enter number of vertices in the polygon: 6
Enter coordinates for vertex 1 (x y): 50 150
Enter coordinates for vertex 2 (x y): 230 150
Enter coordinates for vertex 3 (x y): 80 350
Enter coordinates for vertex 4 (x y): 280 250
Enter coordinates for vertex 5 (x y): 350 250
Enter coordinates for vertex 6 (x y): 280 80
Enter clipping window (xmin ymin xmax ymax): 100 100 300 300
```

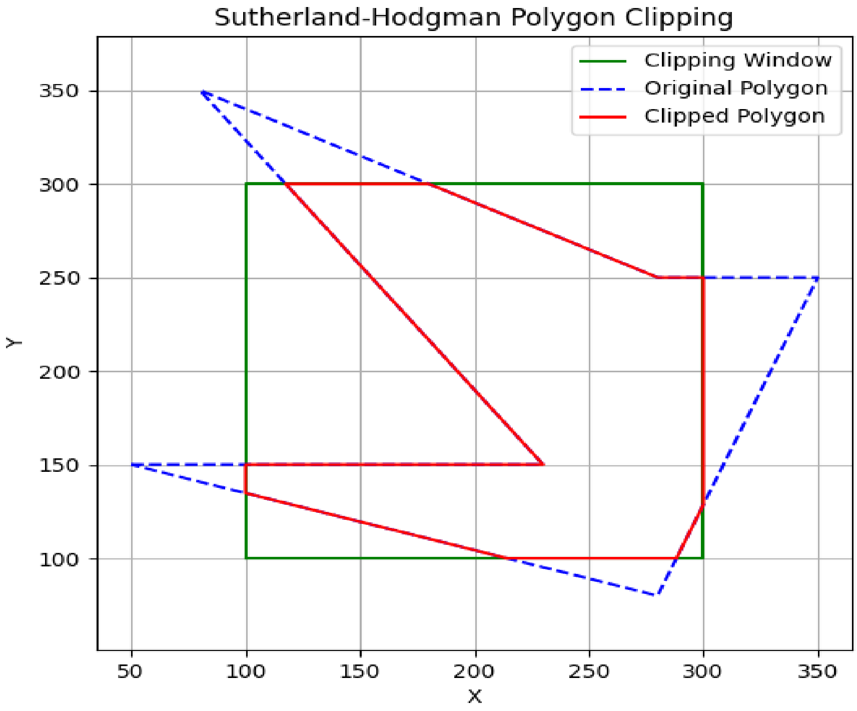
Output:

Original Polygon:

```
[(50.0, 150.0), (230.0, 150.0), (80.0, 350.0), (280.0,     250.0), (350.0,
250.0), (280.0, 80.0)]
```

Clipped Polygon:

```
[(214.28, 100.0), (100.0, 134.78), (100.0, 150.0), (230.0, 150.0),
(117.5, 300.0), (180.0, 300.0), (280.0, 250.0), (300.0, 250.0), (300.0,
128.57), (288.24, 100.0)]
```



Code:

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

xmin = int(input("Enter window x min: "))
ymin = int(input("Enter window y min: "))
xmax = int(input("Enter window x max: "))
ymax = int(input("Enter window y max: "))
grid = np.zeros((xmax+5,ymax+5))

def dda_line(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    steps = max(abs(dx), abs(dy))
    if steps == 0:
        return [(round(x1), round(y1))]
    x_inc = dx / steps
    y_inc = dy / steps
    x, y = x1, y1
    points = []
    for _ in range(steps + 1):  # +1 to include both endpoints
        points.append((round(x), round(y)))
        x += x_inc
        y += y_inc
    return points
def draw_window(xl,yl,xh,yh):
    points1 = dda_line(xl,yl,xh,yl)
    points2 = dda_line(xl,yl,xl,yh)
    points3 = dda_line(xl,yh,xh,yh)
    points4 = dda_line(xh,yl,xh,yh)
    for point in points1:
        grid[point[0]][point[1]]=1
    for point in points2:
        grid[point[0]][point[1]]=1
    for point in points3:
        grid[point[0]][point[1]]=1
    for point in points4:
        grid[point[0]][point[1]]=1
draw_window(xmin,ymin,xmax,ymax)
plt.imshow(grid)
```

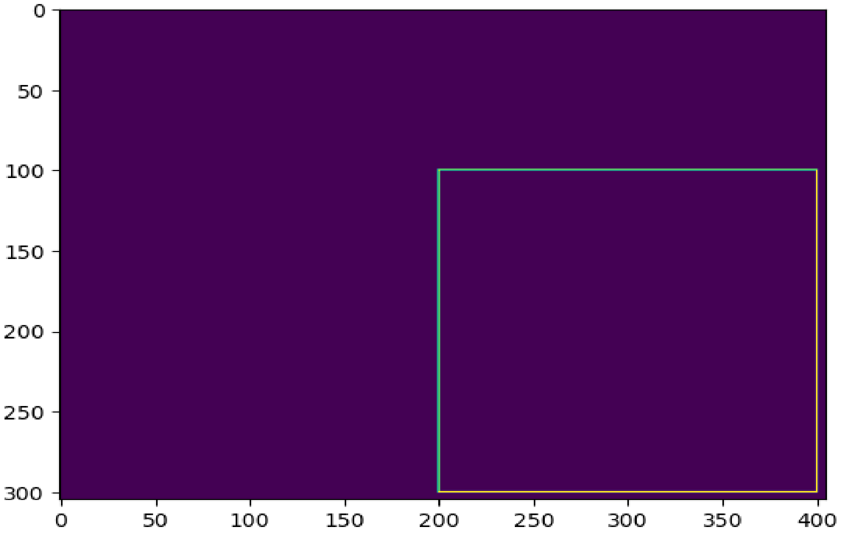
```
def boundary_fill(x, y):
    stack = [(x, y)]
    while stack:
        cx, cy = stack.pop()
        if grid[cx][cy] != 1 and grid[cx][cy] != 2:
            grid[cx][cy] = 2
            # Add neighboring pixels to the stack
            stack.extend([
                (cx + 1, cy), (cx - 1, cy), (cx, cy + 1), (cx, cy - 1),
                (cx + 1, cy + 1), (cx - 1, cy - 1), (cx + 1, cy - 1), (cx - 1, cy + 1)
            ])


boundary_fill(int((xmax + xmin) / 2), int((ymax + ymin) / 2))


plt.imshow(grid)
```

Input:

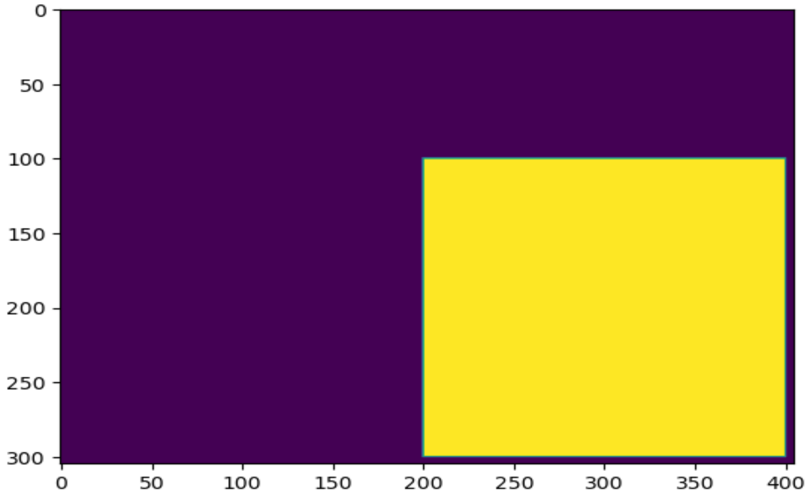
```
Enter window xmin: 100
Enter window ymin: 200
Enter window xmax: 300
Enter window ymax: 400
Enter start point coordinates (x1, y1): 200 300
```

Output:

Before Filling:



After Filling:



```
import numpy as np
import matplotlib.pyplot as plt
xmin = int(input("Enter window x min: "))
ymin = int(input("Enter window y min: "))
xmax = int(input("Enter window x max: "))
ymax = int(input("Enter window y max: "))

grid = np.zeros((xmax+5,ymax+5))
def dda_line(x1, y1, x2, y2):
   dx = x2 - x1
   dy = y2 - y1
   steps = max(abs(dx), abs(dy))
   # Avoid division by zero (if steps=0, it's just a single point)
   if steps == 0:
      return [(round(x1), round(y1))]
   x_inc = dx / steps
   y_inc = dy / steps
   x, y = x1, y1
   points = []
   for _ in range(steps + 1):  # +1 to include both endpoints
      points.append((round(x), round(y)))
      x += x_inc
      y += y_inc
   return points
def draw_window(xl,yl,xh,yh):
   points1 = dda_line(xl,yl,xh,yl)
   points2 = dda_line(xl,yl,xl,yh)
   points3 = dda_line(xl,yh,xh,yh)
   points4 = dda_line(xh,yl,xh,yh)
   for point in points1:
      grid[point[0]][point[1]]=1
   for point in points2:
      grid[point[0]][point[1]]=1
   for point in points3:
      grid[point[0]][point[1]]=1
   for point in points4:
      grid[point[0]][point[1]]=1
draw_window(xmin,ymin,xmax,ymax)
```

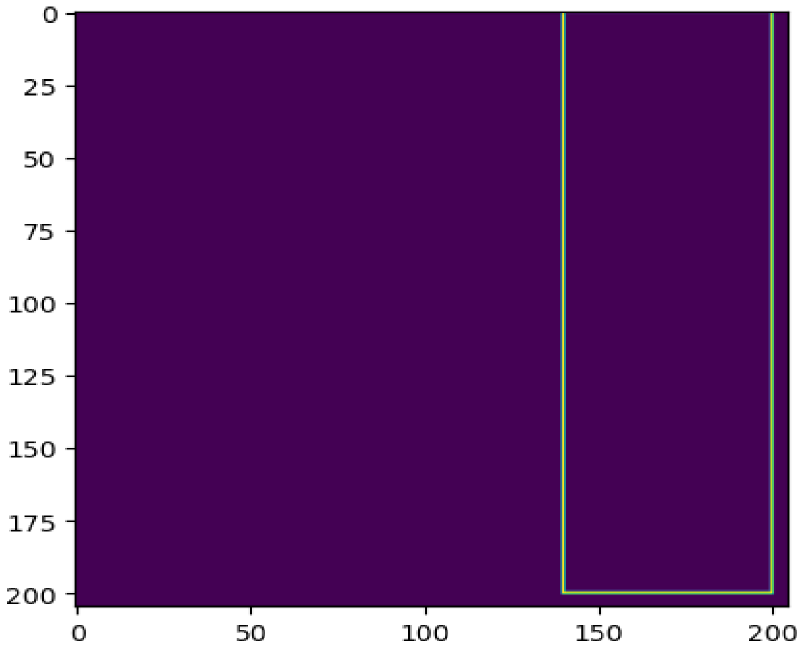
```
def flood_fill(x,y):
    if grid[x][y] == 0:
        grid[x][y] = 2
    else:
        return
    flood_fill(x+1,y)
    flood_fill(x-1,y)
    flood_fill(x,y+1)
    flood_fill(x,y-1)
    flood_fill(x+1,y+1)
    flood_fill(x-1,y-1)
    flood_fill(x+1,y-1)
    flood_fill(x-1,y+1)
flood_fill(int((xmax+xmin)/2), int((xmax+xmin)/2))
plt.imshow(grid)
```

Input:

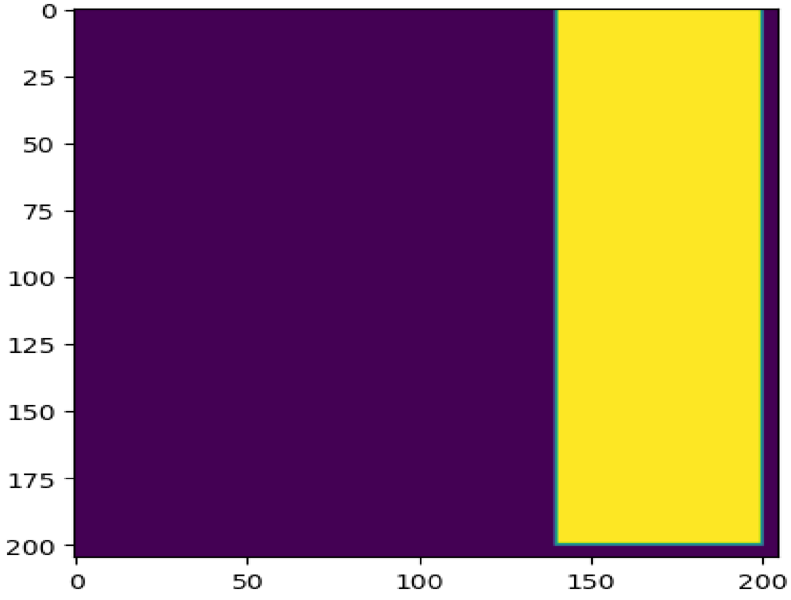
```
Enter window xmin: 100
Enter window ymin: 200
Enter window xmax: 300
Enter window ymax: 400
Enter start point coordinates (x1, y1): 200 300
```

Output:

Before Filling:



After Filling:


0
25
50
75
100
125
150
175
200
0
50
100
150
200