Department of Computer Applications

National Institute of Technology, Kurukshetra

MCA (2024 – 2027)

Computer Graphics & Multimedia Lab (MCA – 233)

Lab Records

Submitted By: -

Name: Ankur Raj
Roll No.: 524410026
Program: MCA
Semester: 3rd
Group - 03

Submitted To: -

Dr. Suresh S.

DECELERATION

I, Ankur Raj, hereby declare that the CG Lab Manual submitted by me is the outcome of my individual effort and understanding. In the process of preparing this manual, I have referred to a variety of learning materials including books, online tutorials, and official documentation but solely for gaining clarity and insight into concepts and programming techniques.

I confirm that no part of this manual has been copied or reproduced from any other student or existing report. All the programs, explanations, and implementations included are written and tested by me independently. If any external sources (such as websites, articles, or textbooks) were consulted, they were strictly used for reference and conceptual learning, not for replication.

Throughout the manual, I have aimed to demonstrate:

- Structured and modular programming style,
- Effective handling of dynamic memory where applicable,
- Readable, well-commented code to ensure clarity and maintainability.

I fully recognize the significance of academic honesty and personal accountability, and I accept full responsibility for the originality and integrity of the content submitted.

Name: Ankur Raj

Roll No: 524410026

INDEX

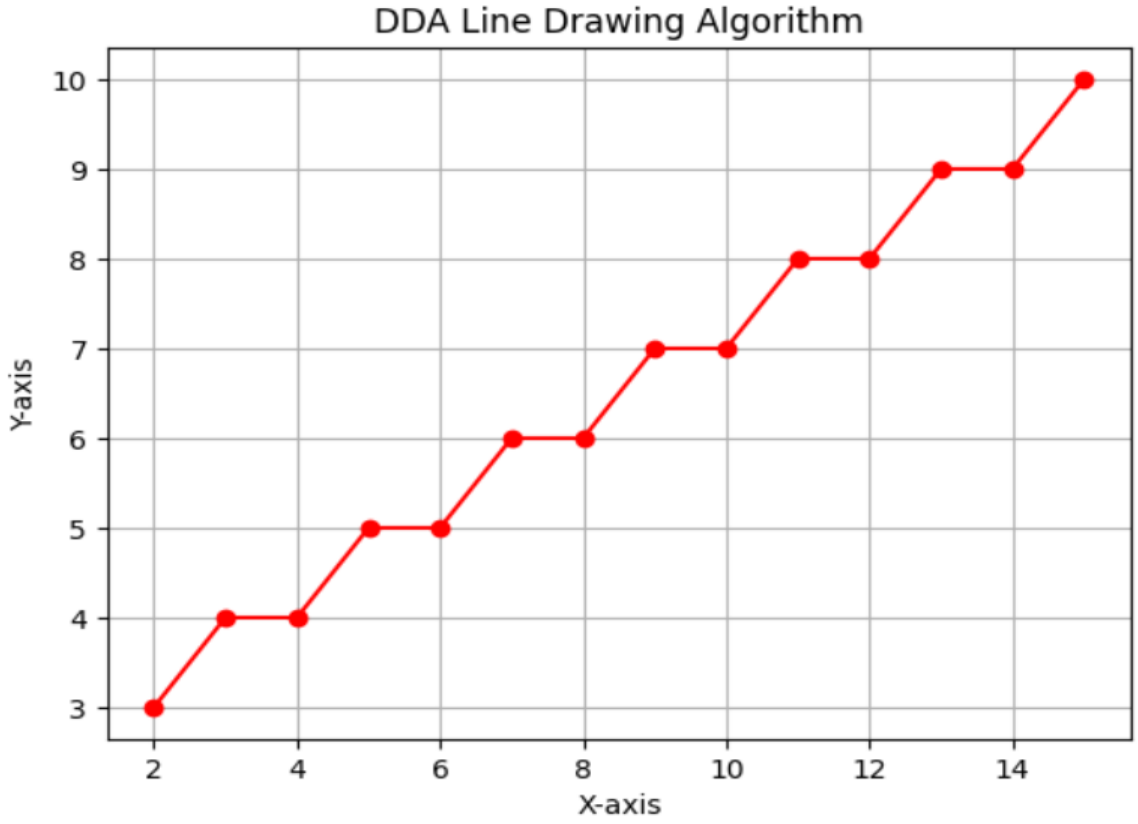
Experiment – 1

To implement DDA algorithms for line and circle.

Program:

```
import matplotlib.pyplot as plt
def dda_line(x1, y1, x2, y2):
    x = x1
    y = y1

    dx = x2 - x1
    dy = y2 - y1

    steps = int(max(abs(dx), abs(dy)))

    x_inc = dx / steps
    y_inc = dy / steps

    x_points = []
    y_points = []

    for i in range(steps + 1):
        x_points.append(round(x))
        y_points.append(round(y))
        x += x_inc
        y += y_inc

    plt.plot(x_points, y_points, 'ro-')
    plt.title('DDA Line Drawing Algorithm')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.grid(True)
    plt.show()
x1, y1 = 2, 3
x2, y2 = 15, 10
dda_line(x1, y1, x2, y2)
```

Output:

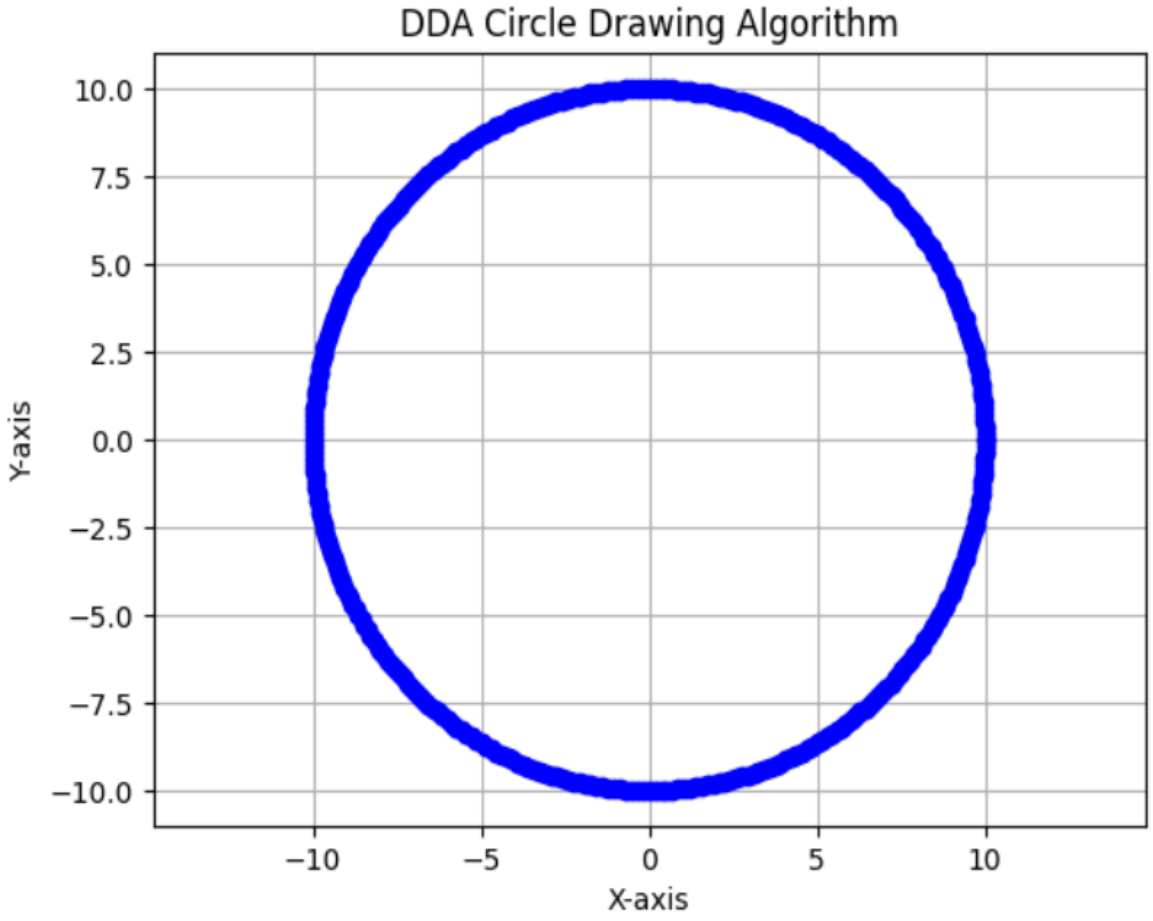


Program:

```
import matplotlib.pyplot as plt
import math

def dda_circle(x_center, y_center, radius):
    x_points = []
    y_points = []

    angle = 0
    while angle <= 360:
        x = x_center + radius * math.cos(math.radians(angle))
        y = y_center + radius * math.sin(math.radians(angle))
        x_points.append(x)
        y_points.append(y)
        angle += 1  # step of 1 degree

    plt.plot(x_points, y_points, 'bo-')
    plt.title('DDA Circle Drawing Algorithm')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.axis('equal')
```

```
    plt.grid(True)
    plt.show()
x_center, y_center, radius = 0, 0, 10
dda_circle(x_center, y_center, radius)
```

Output:



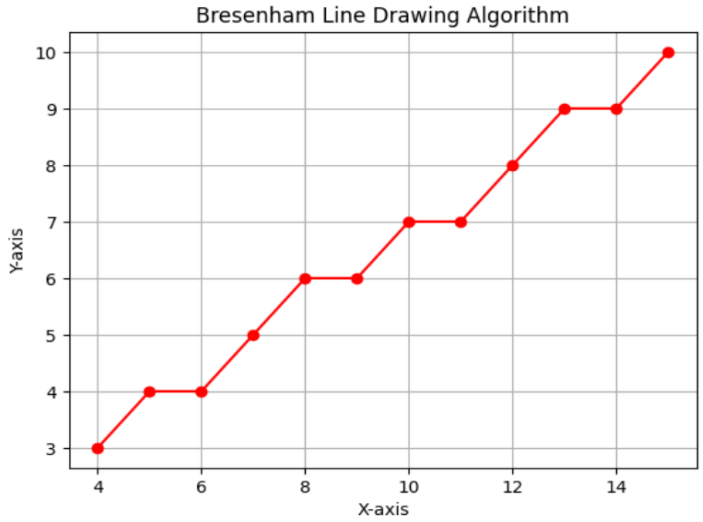
Experiment-2

To implement Bresenham's algorithms for line, circle, and ellipse drawing.

Program:

```
import matplotlib.pyplot as plt

def bresenham_line(x1, y1, x2, y2):
    points = []

    dx = x2 - x1
    dy = y2 - y1

    m = dy / dx

    if m < 1:
        p = 2 * dy - dx

        while x1 <= x2:
            points.append((x1, y1))
            x1 += 1

            if p < 0:
                p = p + 2 * dy
            else:
                p = p + 2 * dy - 2 * dx
                y1 += 1

    else:
        p = 2 * dx - dy

        while y1 <= y2:
            points.append((x1, y1))
            y1 += 1

            if p < 0:
                p = p + 2 * dx
            else:
                p = p + 2 * dx - 2 * dy
                x1 += 1

    return points

# -------------------- DRAW WITH MATPLOTLIB --------------------
pts = bresenham_line(50, 50, 300, 300)
```

```
# Extract X and Y lists
xs = [p[0] for p in pts]
ys = [p[1] for p in pts]

plt.figure(figsize=(6, 6))
plt.scatter(xs, ys, s=10, color='green')   # draw points as green dots

plt.title("Bresenham Line (Matplotlib)")
plt.xlim(0, 350)
plt.ylim(0, 350)
plt.gca().invert_yaxis()   # optional: match screen coordinates
plt.grid(True)
plt.show()
```

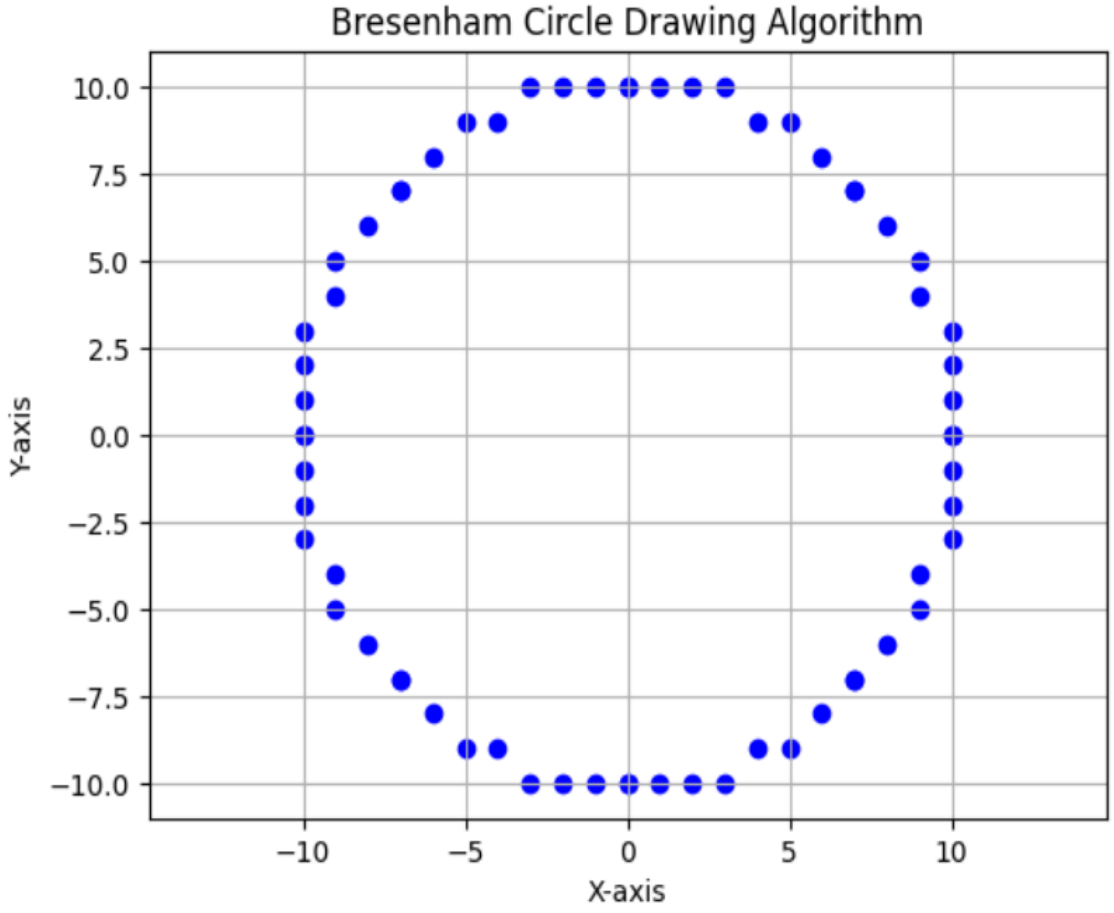


Program:

```
import matplotlib.pyplot as plt

def plot_circle_points(x_center, y_center, x, y, x_points, y_points):

    x_points += [x_center + x, x_center - x, x_center + x, x_center - x,
                 x_center + y, x_center - y, x_center + y, x_center - y]
    y_points += [y_center + y, y_center + y, y_center - y, y_center - y,
                 y_center + x, y_center + x, y_center - x, y_center - x]

def bresenham_circle(x_center, y_center, r):
    x = 0
    y = r
    d = 3 - 2 * r
    x_points = []
    y_points = []

    while x <= y:
        plot_circle_points(x_center, y_center, x, y, x_points, y_points)
```

```
        if d < 0:
            d = d + 4 * x + 6
        else:
            d = d + 4 * (x - y) + 10
            y -= 1
        x += 1

    plt.scatter(x_points, y_points, color='b')
    plt.title('Bresenham Circle Drawing Algorithm')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.axis('equal')
    plt.grid(True)
    plt.show()

bresenham_circle(0, 0, 10)
```

Output:



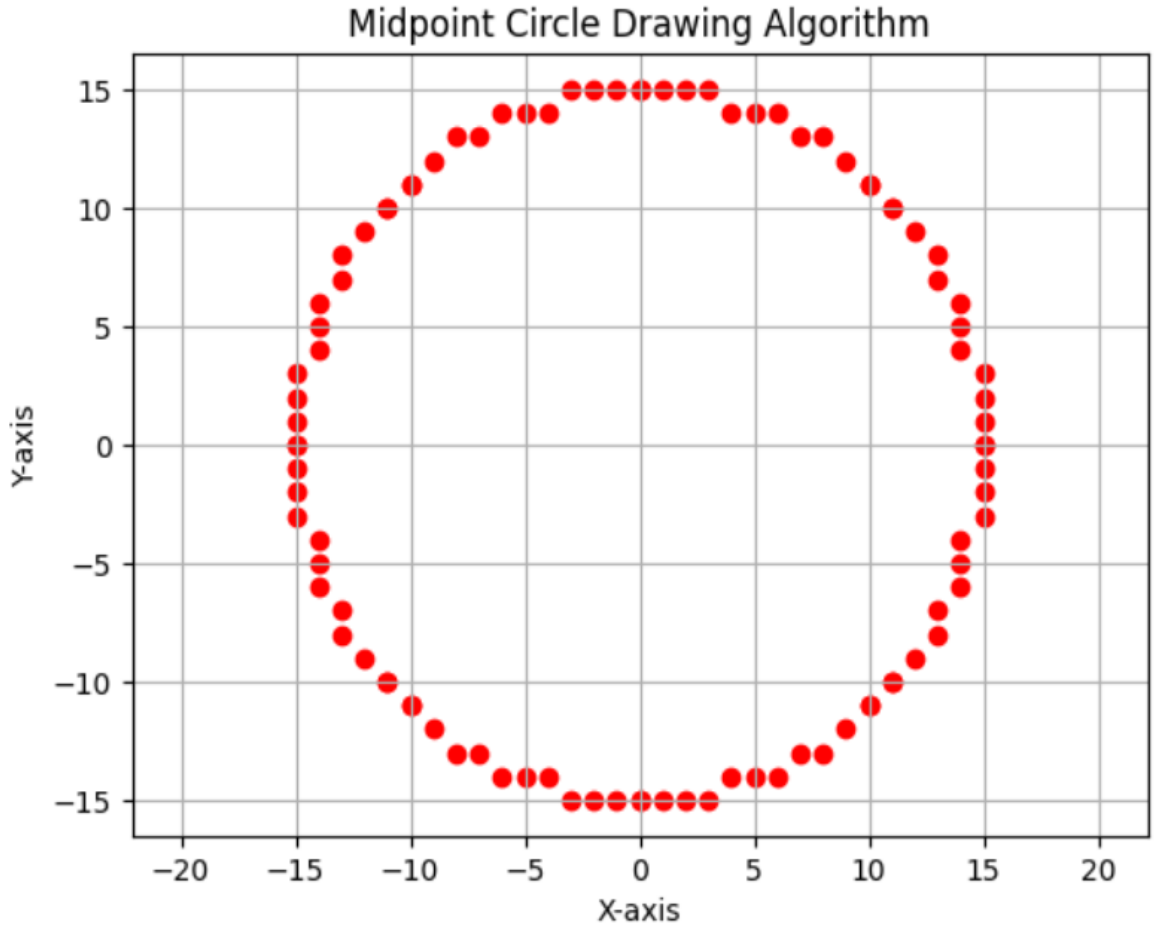
Experiment – 3

To implement Mid-Point circle algorithm.

Program:

```
import matplotlib.pyplot as plt
def plot_circle_points(x_center, y_center, x, y, x_points, y_points):
    x_points += [x_center + x, x_center - x, x_center + x, x_center - x,
                 x_center + y, x_center - y, x_center + y, x_center - y]
    y_points += [y_center + y, y_center + y, y_center - y, y_center - y,
                 y_center + x, y_center + x, y_center - x, y_center - x]

def midpoint_circle(x_center, y_center, r):
    x = 0
    y = r
    p = 1 - r

    x_points = []
    y_points = []

    plot_circle_points(x_center, y_center, x, y, x_points, y_points)

    while x < y:
        x += 1
        if p < 0:
            p = p + 2 * x + 1
        else:
            y -= 1
            p = p + 2 * (x - y) + 1

        plot_circle_points(x_center, y_center, x, y, x_points, y_points)

    plt.scatter(x_points, y_points, color='r')
    plt.title('Midpoint Circle Drawing Algorithm')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.axis('equal')
    plt.grid(True)
    plt.show()

midpoint_circle(0, 0, 15)
```

Output:


Midpoint Circle Drawing Algorithm
15
10
5
0
−5
−10
−15
Y-axis
−20
−15
−10
−5
0
5
10
15
20
X-axis

Experiment - 4

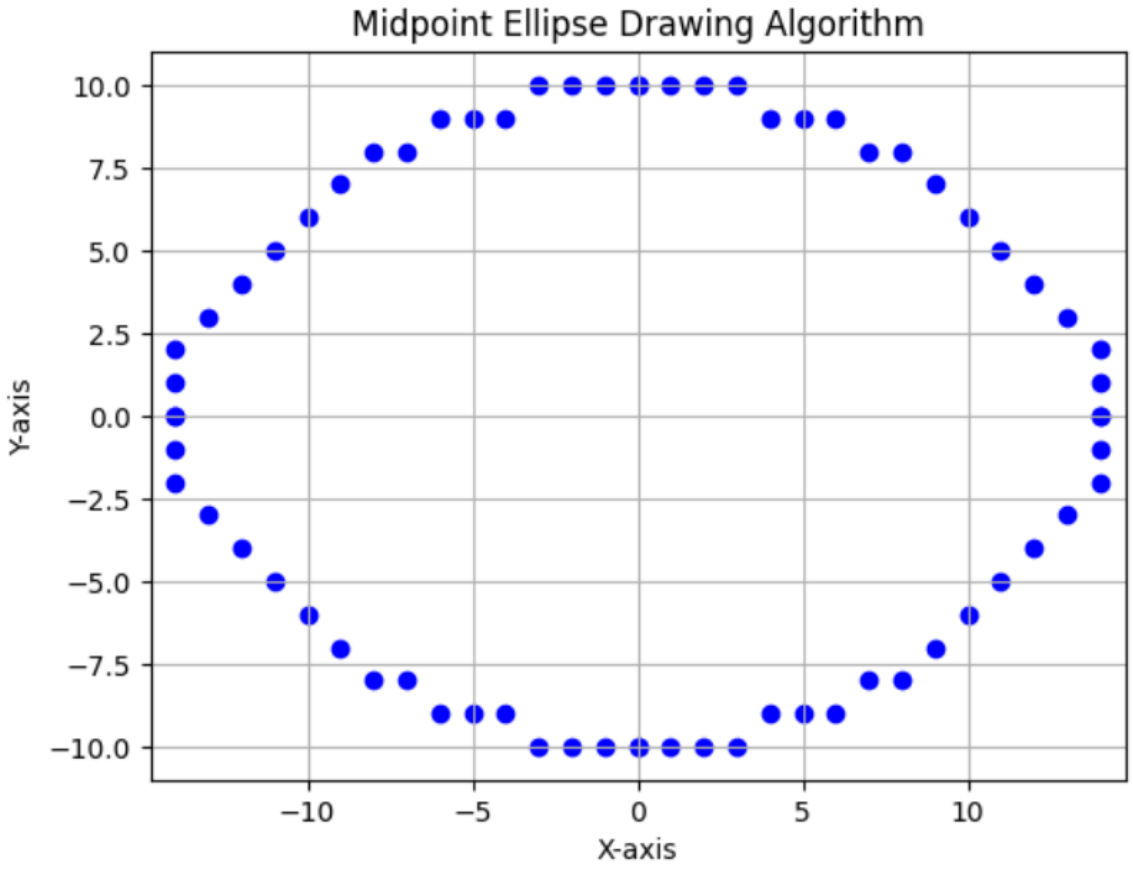
To implement Mid-Point ellipse algorithm.

Program:

```
import matplotlib.pyplot as plt

def plot_ellipse_points(x_center, y_center, x, y, x_points, y_points):
    x_points += [x_center + x, x_center - x, x_center + x, x_center - x]
    y_points += [y_center + y, y_center + y, y_center - y, y_center - y]

def midpoint_ellipse(x_center, y_center, rx, ry):
    x_points = []
    y_points = []

    x = 0
    y = ry

    p1 = (ry**2) - (rx**2 * ry) + (0.25 * rx **2)

    dx = 2 * ry **2 * x
    dy = 2 * rx **2 * y

    # --- Region 1 ---
    while dx < dy:
        plot_ellipse_points(x_center, y_center, x, y, x_points, y_points)
        x += 1
        dx = 2 * ry **2 * x
        if p1 < 0:
            p1 += ry **2 * (2 * x + 3)
        else:
            y -= 1
            dy = 2 * rx **2 * y
            p1 += ry **2 * (2 * x + 3) + rx **2 * (-2 * y + 2)

    # --- Region 2 initial decision parameter ---
    p2 = (ry **2) * (x + 0.5)**2 + (rx **2) * (y - 1)**2 - (rx **2 * ry **2)

    # --- Region 2 ---
    while y >= 0:
        plot_ellipse_points(x_center, y_center, x, y, x_points, y_points)
        y -= 1
```

```
        dy = 2 * rx **2 * y
        if p2 > 0:
            p2 += rx **2 * (-2 * y + 3)
        else:
            x += 1
            dx = 2 * ry **2 * x
            p2 += ry **2 * (2 * x + 2) + rx **2 * (-2 * y + 3)

    # Plot ellipse
    plt.scatter(x_points, y_points, color='b')
    plt.title('Midpoint Ellipse Drawing Algorithm')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.axis('equal')
    plt.grid(True)
    plt.show()

midpoint_ellipse(0, 0, 15, 10)
```

Output:

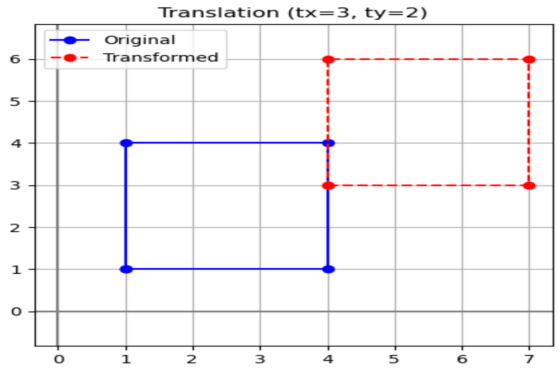


Experiment - 5

To perform 2D Transformations such as translation, rotation, scaling, reflection, and shearing.

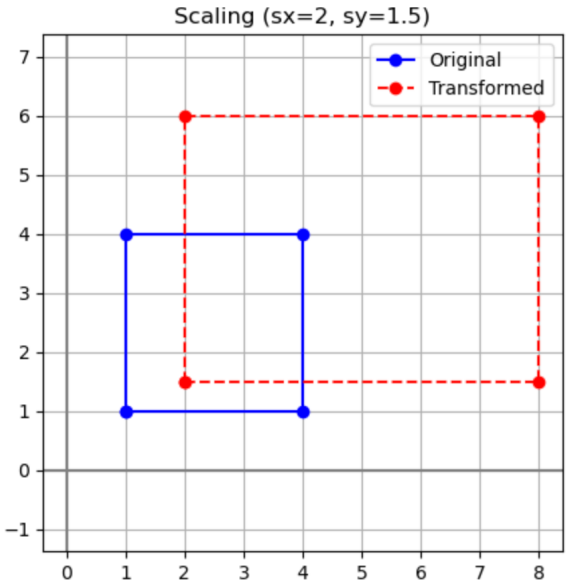
Program:

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

def translate(point, tx, ty):
    point_h = np.array([point[0], point[1], 1])
    T = np.array([[1, 0, tx],
                  [0, 1, ty],
                  [0, 0, 1]])
    result = np.dot(T, point_h)
    return result[0], result[1]

# Example
point = (2, 3)
tx, ty = 4, 5
new_point = translate(point, tx, ty)

print("Original:", point)
print("Translated:", new_point)

# Plot
plt.figure()
plt.scatter(*point, color='blue', label="Original")
plt.scatter(*new_point, color='red', label="Translated")
plt.axhline(0, color='gray');
plt.axvline(0, color='gray')
plt.legend(); plt.title("Translation")
plt.grid(True);
plt.show()q
```

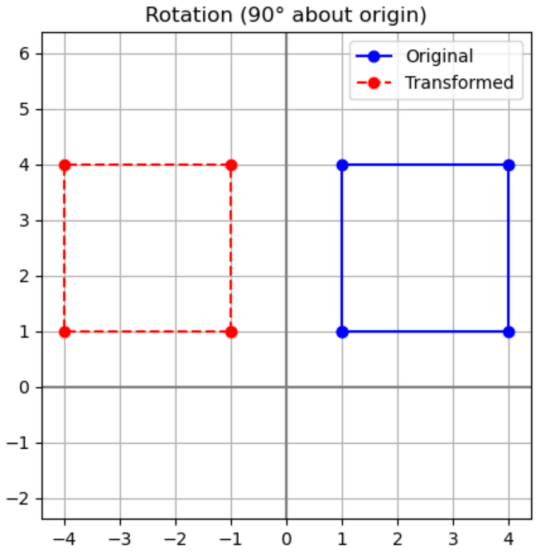


Scaling

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

def scale(point, sx, sy):
    point_h = np.array([point[0], point[1], 1])
    S = np.array([[sx, 0, 0],
                  [0, sy, 0],
                  [0, 0, 1]])
    result = np.dot(S, point_h)
    return result[0], result[1]

# Example
point = (2, 3)
sx, sy = 2, 3
new_point = scale(point, sx, sy)

print("Original:", point)
print("Scaled:", new_point)

# Plot
plt.figure()
plt.scatter(*point, color='blue', label="Original")
plt.scatter(*new_point, color='red', label="Scaled")
plt.axhline(0, color='gray'); plt.axvline(0, color='gray')
plt.legend();
plt.title("Scaling")
plt.grid(True);
plt.show()
```

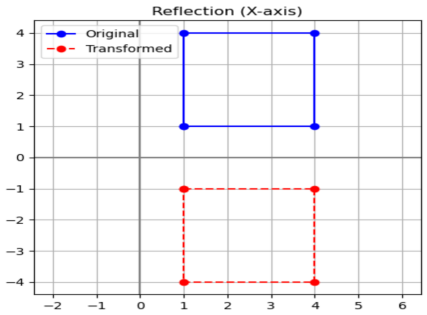



```
# Rotation
import numpy as np
import matplotlib.pyplot as plt

def rotate(point, theta_deg):
    theta = np.radians(theta_deg)
    point_h = np.array([point[0], point[1], 1])
    R = np.array([[np.cos(theta), -np.sin(theta), 0],
                  [np.sin(theta),  np.cos(theta), 0],
                  [0, 0, 1]])
    result = np.dot(R, point_h)
    return result[0], result[1]

# Example
point = (2, 3)
new_point = rotate(point, 90)

print("Original:", point)
print("Rotated (90°):", new_point)

# Plot
plt.figure()
plt.scatter(*point, color='blue', label="Original")
plt.scatter(*new_point, color='red', label="Rotated 90°")
plt.axhline(0, color='gray'); plt.axvline(0, color='gray')
plt.legend();
plt.title("Rotation (90°)")
plt.grid(True);
plt.show()
```




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

def reflect(point, axis='x'):
    point_h = np.array([point[0], point[1], 1])
    if axis == 'x':
        M = np.array([[1, 0, 0],
                      [0, -1, 0],
                      [0, 0, 1]])
    elif axis == 'y':
        M = np.array([[-1, 0, 0],
                      [0, 1, 0],
                      [0, 0, 1]])
    elif axis == 'origin':
        M = np.array([[-1, 0, 0],
                      [0, -1, 0],
                      [0, 0, 1]])
    result = np.dot(M, point_h)
    return result[0], result[1]

# Example
point = (2, 3)
new_point = reflect(point, 'x')  # reflect across x-axis

print("Original:", point)
print("Reflected (X-axis):", new_point)

# Plot
plt.figure()
```

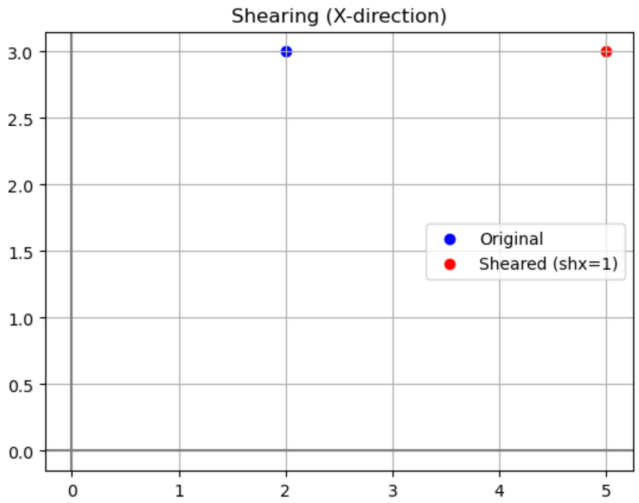
```
plt.scatter(*point, color='blue', label="Original")
plt.scatter(*new_point, color='red', label="Reflected (X-axis)")
plt.axhline(0, color='gray');
plt.axvline(0, color='gray')
plt.legend();
plt.title("Reflection (X-axis)")
plt.grid(True);
plt.show()
```




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

def shear(point, shx=0, shy=0):
    point_h = np.array([point[0], point[1], 1])
    Sh = np.array([[1, shx, 0],
                   [shy, 1, 0],
                   [0, 0, 1]])
    result = np.dot(Sh, point_h)
    return result[0], result[1]

# Example
point = (2, 3)
new_point = shear(point, shx=1)

print("Original:", point)
print("Sheared (shx=1):", new_point)

# Plot
plt.figure()
plt.scatter(*point, color='blue', label="Original")
plt.scatter(*new_point, color='red', label="Sheared (shx=1)")
plt.axhline(0, color='gray');
plt.axvline(0, color='gray')
plt.legend();
plt.title("Shearing (X-direction)")
plt.grid(True);
plt.show()
```

Output:

```
Original: (2, 3)
Sheared (shx=1): (5, 3)
```



Experiment - 6

To implement Cohen–Sutherland 2D clipping and window–viewport mapping.

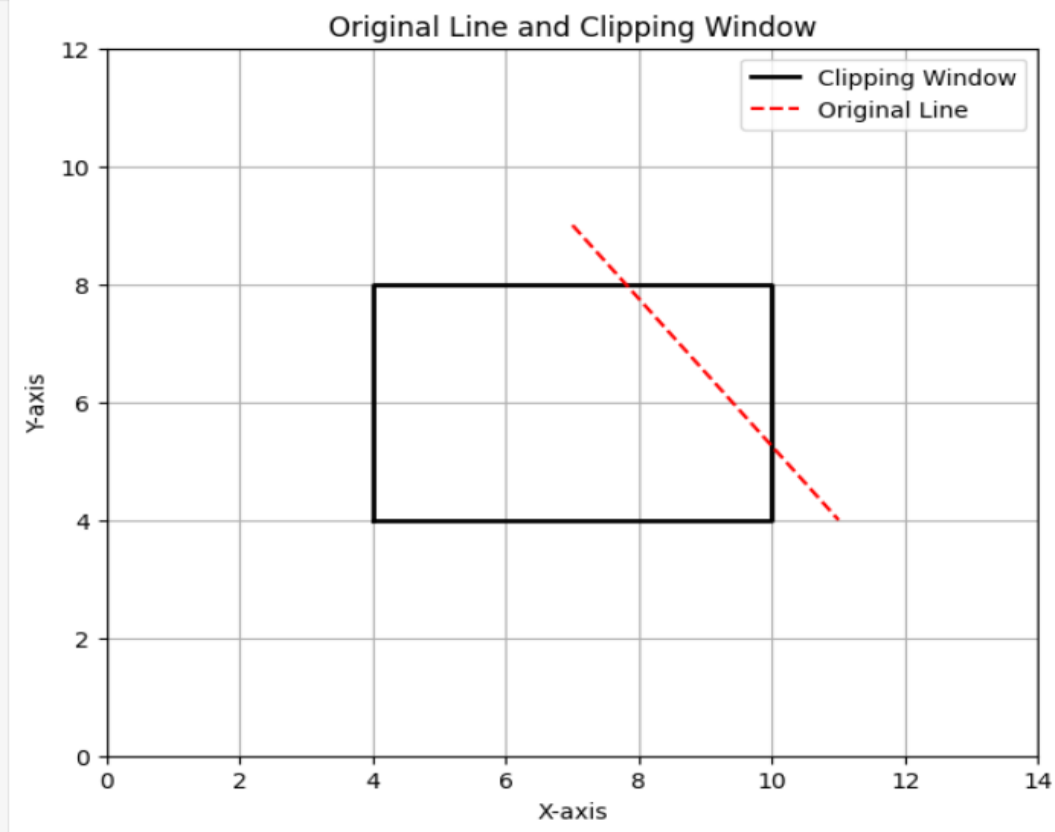
Program:

```
import matplotlib.pyplot as plt

INSIDE = 0
LEFT = 1
RIGHT = 2
BOTTOM = 4
TOP = 8

def compute_code(x, y, x_min, y_min, x_max, y_max):
    code = INSIDE
    if x < x_min:
        code |= LEFT
    elif x > x_max:
        code |= RIGHT
    if y < y_min:
        code |= BOTTOM
    elif y > y_max:
        code |= TOP
    return code

def cohen_sutherland_clip(x1, y1, x2, y2, x_min, y_min, x_max, y_max):
    code1 = compute_code(x1, y1, x_min, y_min, x_max, y_max)
    code2 = compute_code(x2, y2, x_min, y_min, x_max, y_max)
    accept = False
```

```
    while True:
        if code1 == 0 and code2 == 0:
            accept = True
            break
        elif (code1 & code2) != 0:
            break
        else:
            code_out = code1 if code1 != 0 else code2
            if code_out & TOP:
                x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1)
                y = y_max
            elif code_out & BOTTOM:
                x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1)
                y = y_min
            elif code_out & RIGHT:
                y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1)
                x = x_max
            elif code_out & LEFT:
                y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1)
                x = x_min

            if code_out == code1:
                x1, y1 = x, y
                code1 = compute_code(x1, y1, x_min, y_min, x_max, y_max)
            else:
                x2, y2 = x, y
                code2 = compute_code(x2, y2, x_min, y_min, x_max, y_max)

    if accept:
        return (x1, y1, x2, y2)
    else:
        return None

x_min, y_min, x_max, y_max = 4, 4, 10, 8   # Window
x1, y1, x2, y2 = 7, 9, 11, 4               # Line

clipped = cohen_sutherland_clip(x1, y1, x2, y2, x_min, y_min, x_max, y_max)

plt.figure(figsize=(7, 6))
plt.title("Original Line and Clipping Window")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.xlim(0, 14)
plt.ylim(0, 12)
```

```
# Window
plt.plot([x_min, x_max, x_max, x_min, x_min],
        [y_min, y_min, y_max, y_max, y_min],
        color='black', linewidth=2, label='Clipping Window')

# Original line
plt.plot([x1, x2], [y1, y2], 'r--', label='Original Line')

plt.legend()
plt.show()

# --- Figure 2: Clipped line only ---
plt.figure(figsize=(7, 6))
plt.title("Clipped Line After Cohen–Sutherland")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.xlim(0, 14)
plt.ylim(0, 12)

plt.plot([x_min, x_max, x_max, x_min, x_min],
        [y_min, y_min, y_max, y_max, y_min],
        color='black', linewidth=2, label='Clipping Window')

if clipped:
    x1c, y1c, x2c, y2c = clipped
    plt.plot([x1c, x2c], [y1c, y2c], 'g', linewidth=2, label='Clipped Line')
    plt.scatter([x1c, x2c], [y1c, y2c], color='blue')
    print("Clipped line coordinates:")
    print(f"({x1c:.2f}, {y1c:.2f}) to ({x2c:.2f}, {y2c:.2f})")
else:
    print("Line is completely outside the window.")

plt.legend()
plt.show()
```

Output:


Original Line and Clipping Window
Clipping Window
Original Line
Y-axis
X-axis
0
2
4
6
8
10
12
14

```
Clipped line coordinates:
(7.80, 8.00) to (10.00, 5.25)
```

Experiment – 7

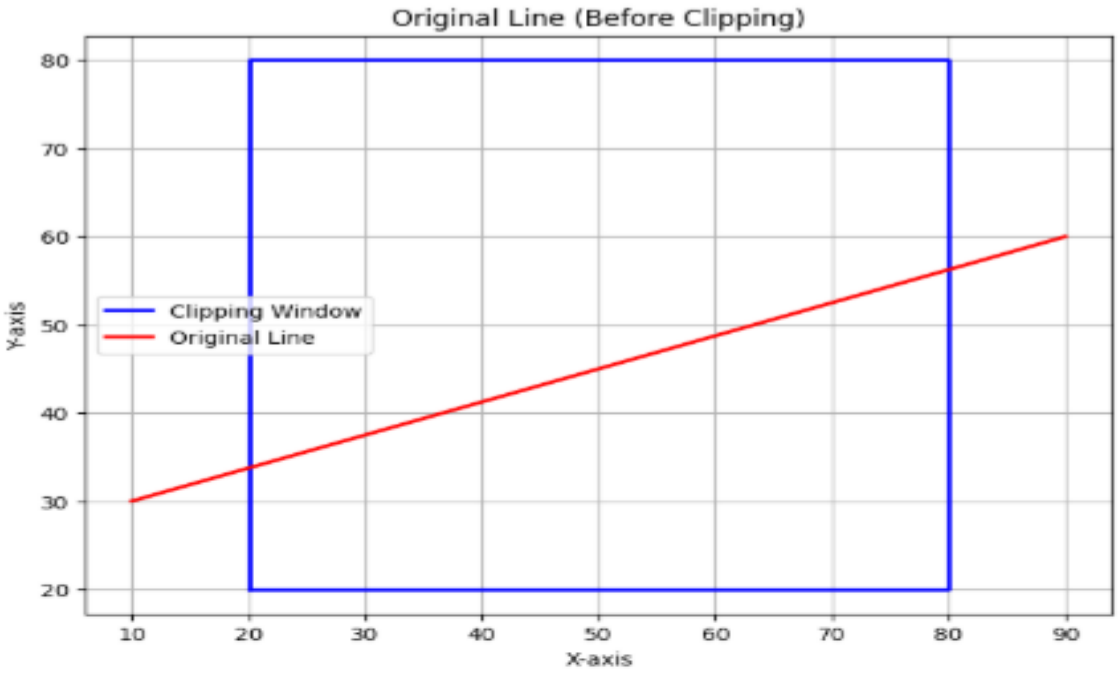
To implement Liang–Barsky line clipping algorithm.

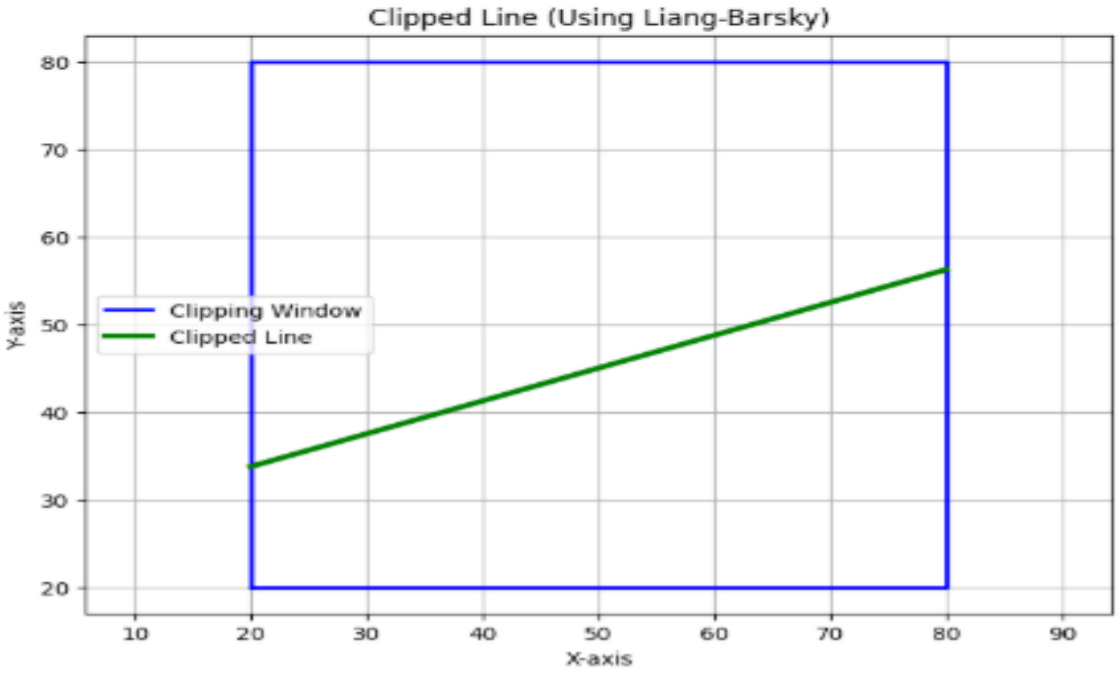
Program:

```
import matplotlib.pyplot as plt

def liang_barsky(x_min, y_min, x_max, y_max, x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    p = [-dx, dx, -dy, dy]
    q = [x1 - x_min, x_max - x1, y1 - y_min, y_max - y1]
    t_enter = 0.0
    t_exit = 1.0

    for i in range(4):
        if p[i] == 0:  # Line parallel to boundary
            if q[i] < 0:
                return None  # Outside and parallel → reject
        else:
            t = q[i] / p[i]
            if p[i] < 0:        # entering
                t_enter = max(t_enter, t)
            else:               # leaving
                t_exit = min(t_exit, t)

    if t_enter > t_exit:
        return None

    x1_clip = x1 + t_enter * dx
    y1_clip = y1 + t_enter * dy
    x2_clip = x1 + t_exit * dx
    y2_clip = y1 + t_exit * dy

    return x1_clip, y1_clip, x2_clip, y2_clip

# Clipping window
x_min, y_min = 20, 20
x_max, y_max = 80, 80

# Line endpoints
x1, y1 = 10, 30
x2, y2 = 90, 60

clipped_line = liang_barsky(x_min, y_min, x_max, y_max, x1, y1, x2, y2)
```

```
# ------------------------------------------------------------
#  IMAGE 1 → ORIGINAL LINE (NOT CLIPPED)
# ------------------------------------------------------------
plt.figure(figsize=(8, 6))
plt.title("Original Line (Before Clipping)")

# Draw window
plt.plot([x_min, x_max, x_max, x_min, x_min],
         [y_min, y_min, y_max, y_max, y_min],
         'b', linewidth=2, label="Clipping Window")

# Draw original line
plt.plot([x1, x2], [y1, y2], 'r', linewidth=2, label="Original Line")

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid()
plt.axis("equal")
plt.legend()
plt.show()

# ------------------------------------------------------------
#  IMAGE 2 → CLIPPED LINE
# ------------------------------------------------------------
plt.figure(figsize=(8, 6))
plt.title("Clipped Line (Using Liang-Barsky)")

# Draw window
plt.plot([x_min, x_max, x_max, x_min, x_min],
         [y_min, y_min, y_max, y_max, y_min],
         'b', linewidth=2, label="Clipping Window")

if clipped_line is not None:
    x1c, y1c, x2c, y2c = clipped_line
    plt.plot([x1c, x2c], [y1c, y2c], 'g', linewidth=3, label="Clipped Line")
else:
    plt.text(30, 50, "Line is completely outside", fontsize=14, color="red")

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid()
plt.axis("equal")
plt.legend()
plt.show()
```

Output:


Original Line (Before Clipping)
Clipping Window
Original Line
Y-axis
X-axis
80
70
60
50
40
30
20
10
20
30
40
50
60
70
80
90


Clipped Line (Using Liang-Barsky)
Clipping Window
Clipped Line
Y-axis
X-axis
80
70
60
50
40
30
20
10
20
30
40
50
60
70
80
90

Experiment – 8

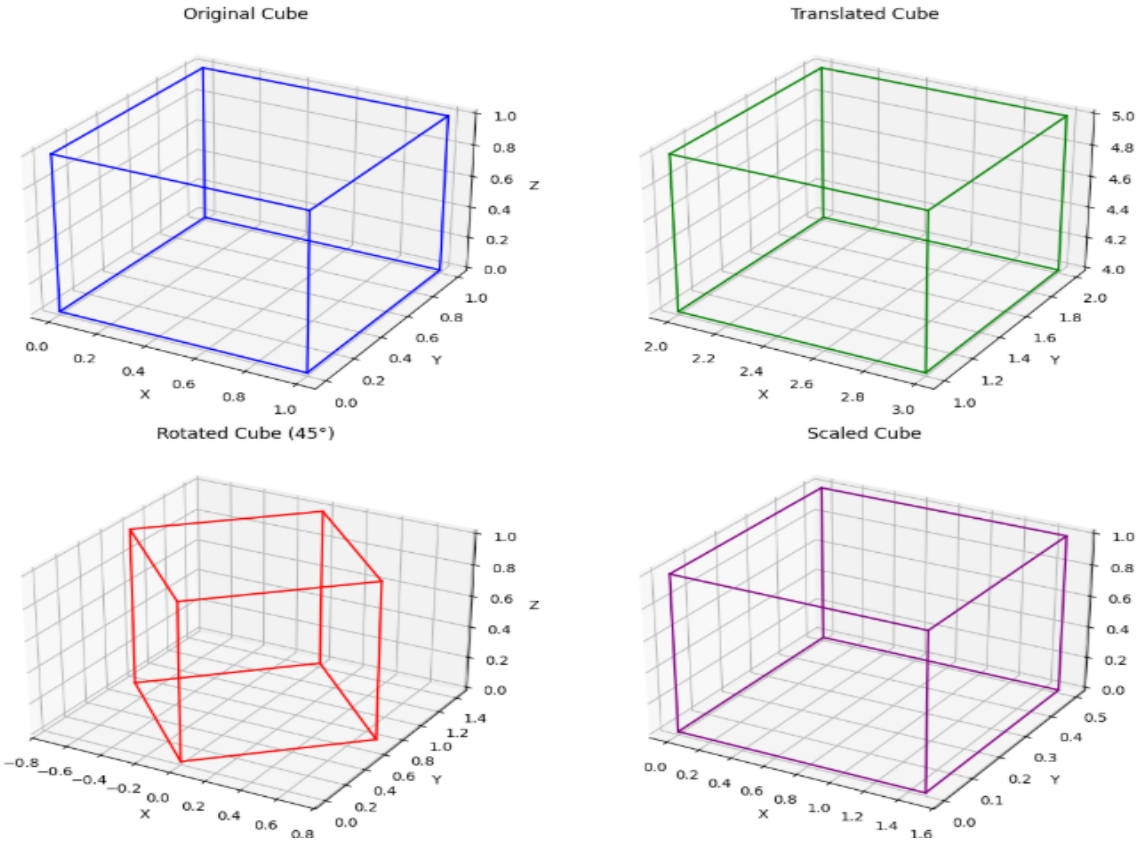
To perform 3D Transformations such as translation, rotation, and scaling.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Simple function to draw cube edges
def plot_cube(ax, points, title, color):
    edges = [
        [0,1],[1,2],[2,3],[3,0],
        [4,5],[5,6],[6,7],[7,4],
        [0,4],[1,5],[2,6],[3,7]
    ]

    for e in edges:
        ax.plot(
            [points[e[0],0], points[e[1],0]],
            [points[e[0],1], points[e[1],1]],
            [points[e[0],2], points[e[1],2]],
            color=color
        )

    ax.set_title(title)
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")

cube = np.array([
    [0,0,0],[1,0,0],[1,1,0],[0,1,0],
    [0,0,1],[1,0,1],[1,1,1],[0,1,1]
], float)

# ---- TRANSLATION ----
translated = cube + np.array([2, 1, 4])

# ---- ROTATION (Z-axis) ----
angle = np.radians(45)
Rz = np.array([
    [np.cos(angle), -np.sin(angle), 0],
    [np.sin(angle),  np.cos(angle), 0],
    [0, 0, 1]
])
```

```
rotated = cube.dot(Rz.T)

# ---- SCALING ----
scaled = cube * np.array([1.5, 0.5, 1])
fig = plt.figure(figsize=(12, 10))

ax1 = fig.add_subplot(221, projection="3d")
plot_cube(ax1, cube, "Original Cube", "blue")

ax2 = fig.add_subplot(222, projection="3d")
plot_cube(ax2, translated, "Translated Cube", "green")

ax3 = fig.add_subplot(223, projection="3d")
plot_cube(ax3, rotated, "Rotated Cube (45°)", "red")

ax4 = fig.add_subplot(224, projection="3d")
plot_cube(ax4, scaled, "Scaled Cube", "purple")

plt.tight_layout()
plt.show()
```

Output:



Experiment – 9

To draw different shapes such as hut, face, kite, fish, etc.

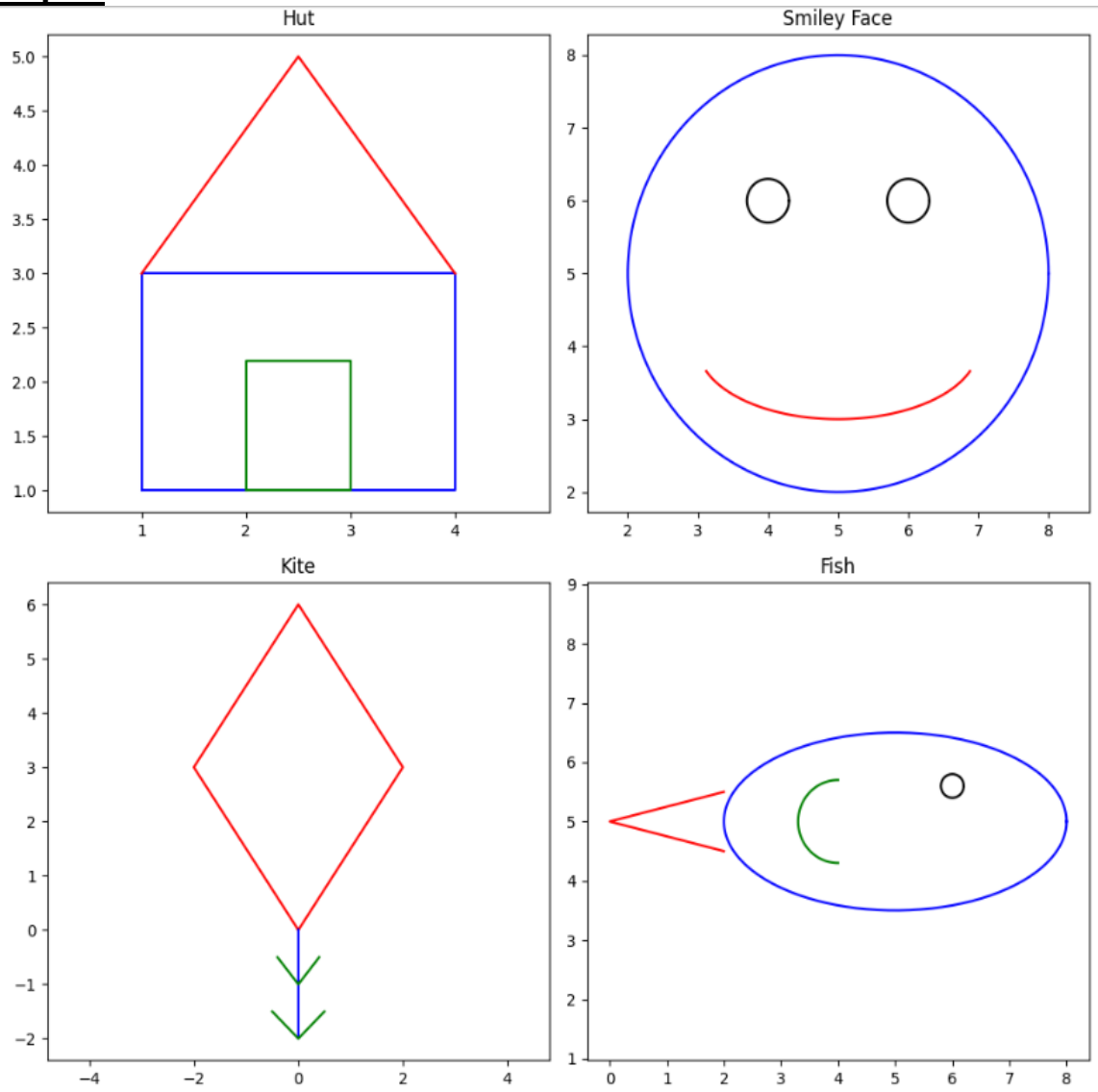
Program:

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

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# ---------------------------------------------------------
# 1) HUT
# ---------------------------------------------------------
ax = axs[0][0]

x_house = [1, 1, 4, 4, 1]
y_house = [1, 3, 3, 1, 1]

x_roof = [1, 2.5, 4]
y_roof = [3, 5, 3]

x_door = [2, 2, 3, 3, 2]
y_door = [1, 2.2, 2.2, 1, 1]

ax.plot(x_house, y_house, 'b')
ax.plot(x_roof, y_roof, 'r')
ax.plot(x_door, y_door, 'g')
ax.set_title("Hut")
ax.axis("equal")

# ---------------------------------------------------------
# 2) FACE
# ---------------------------------------------------------
ax = axs[0][1]

theta = np.linspace(0, 2*np.pi, 200)

# Face
x_face = 5 + 3*np.cos(theta)
y_face = 5 + 3*np.sin(theta)

# Eyes
x_eye1 = 4 + 0.3*np.cos(theta)
y_eye1 = 6 + 0.3*np.sin(theta)

x_eye2 = 6 + 0.3*np.cos(theta)
```

```
y_eye2 = 6 + 0.3*np.sin(theta)

# Smile arc
theta_smile = np.linspace(200*np.pi/180, 340*np.pi/180, 100)
x_smile = 5 + 2*np.cos(theta_smile)
y_smile = 4 + 1*np.sin(theta_smile)

ax.plot(x_face, y_face, 'b')
ax.plot(x_eye1, y_eye1, 'k')
ax.plot(x_eye2, y_eye2, 'k')
ax.plot(x_smile, y_smile, 'r')
ax.set_title("Smiley Face")
ax.axis("equal")

# ----------------------------------------------------------
# 3) KITE
# ----------------------------------------------------------
ax = axs[1][0]

x_kite = [0, 2, 0, -2, 0]
y_kite = [0, 3, 6, 3, 0]

x_tail = [0, 0]
y_tail = [0, -2]

x_b1 = [-0.5, 0, 0.5]
y_b1 = [-1.5, -2, -1.5]

x_b2 = [-0.4, 0, 0.4]
y_b2 = [-0.5, -1, -0.5]

ax.plot(x_kite, y_kite, 'r')
ax.plot(x_tail, y_tail, 'b')
ax.plot(x_b1, y_b1, 'g')
ax.plot(x_b2, y_b2, 'g')
ax.set_title("Kite")
ax.axis("equal")

# ----------------------------------------------------------
# 4) FISH
# ----------------------------------------------------------
ax = axs[1][1]

# Fish body (ellipse)
x_body = 5 + 3*np.cos(theta)
y_body = 5 + 1.5*np.sin(theta)
```

```
# Tail triangle
x_tail = [2, 0, 2]
y_tail = [5.5, 5, 4.5]

# Eye
x_eye = 6 + 0.2*np.cos(theta)
y_eye = 5.6 + 0.2*np.sin(theta)

# Gill (small arc)
theta_g = np.linspace(np.pi/2, 3*np.pi/2, 100)
x_gill = 4 + 0.7*np.cos(theta_g)
y_gill = 5 + 0.7*np.sin(theta_g)

ax.plot(x_body, y_body, 'b')
ax.plot(x_tail, y_tail, 'r')
ax.plot(x_eye, y_eye, 'k')
ax.plot(x_gill, y_gill, 'g')
ax.set_title("Fish")
ax.axis("equal")

plt.suptitle("Experiment 9: Drawing Shapes (Hut, Face, Kite, Fish)", fontsize=16)
plt.tight_layout()
plt.show()
```

Output:



Experiment – 10

To perform basic operations on image using any image editing software.

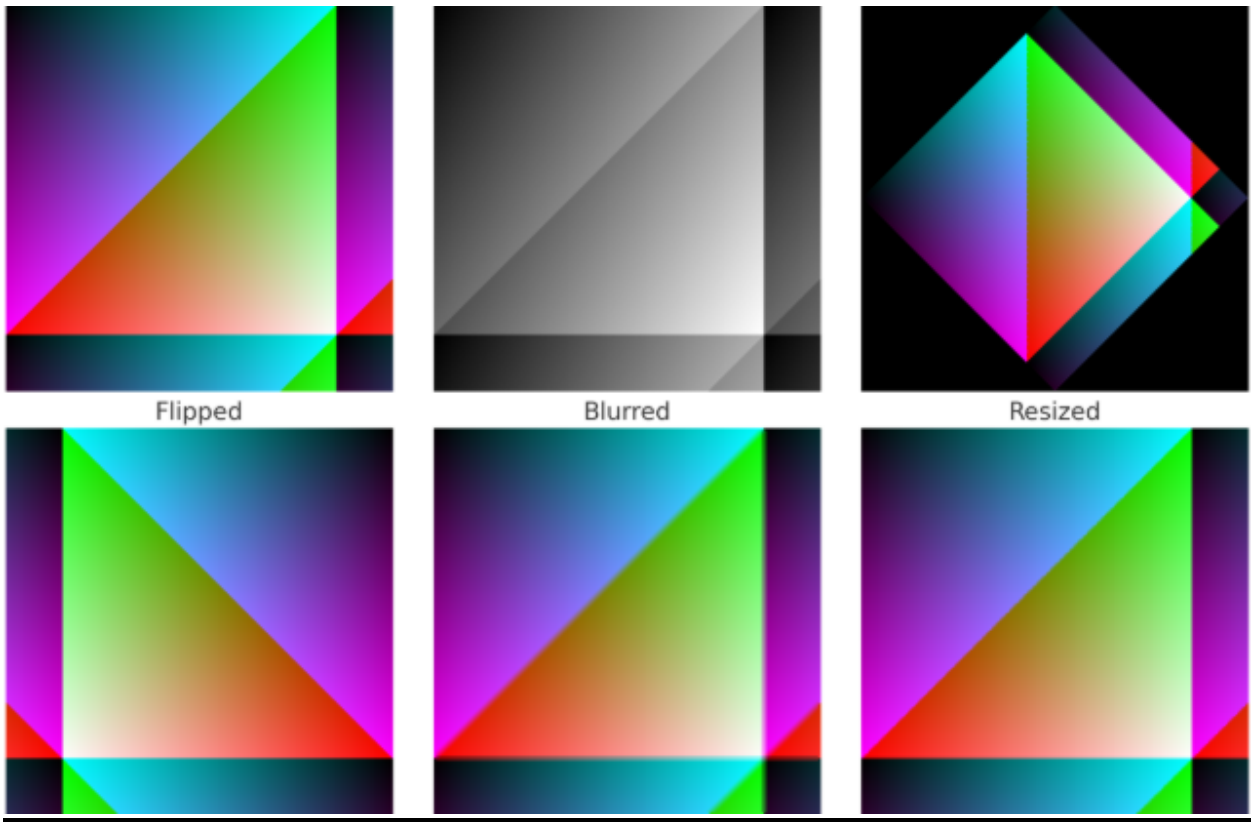
Program:

```
from PIL import Image, ImageOps, ImageFilter
import matplotlib.pyplot as plt
from tkinter import Tk, filedialog

# ---------------------------------------------------------
# 1. OPEN FILE DIALOG (Select your image)
# ---------------------------------------------------------
root = Tk()
root.withdraw()  # Hide the main Tkinter window

print("Select an image file...")
file_path = filedialog.askopenfilename(
    title="Choose Image",
    filetypes=[("Image Files", "*.jpg *.jpeg *.png *.bmp *.gif")]
)

# Load selected image
img = Image.open(file_path)

# Save original
img.save("original.png")


# ---------------------------------------------------------
# 2. Edit Operations
# ---------------------------------------------------------
gray = ImageOps.grayscale(img)
rotated = img.rotate(45, expand=True)
flipped = ImageOps.mirror(img)
blurred = img.filter(ImageFilter.BLUR)
resized = img.resize((300, 300))


# ---------------------------------------------------------
# 3. Save all results
# ---------------------------------------------------------
gray.save("gray.png")
rotated.save("rotated.png")
flipped.save("flipped.png")
blurred.save("blurred.png")
```

```
resized.save("resized.png")


# ---------------------------------------------------------
# 4. SHOW RESULTS ON SCREEN
# ---------------------------------------------------------
fig, axs = plt.subplots(2, 3, figsize=(12, 8))

axs[0,0].imshow(img)
axs[0,0].set_title("Original")
axs[0,0].axis("off")

axs[0,1].imshow(gray, cmap="gray")
axs[0,1].set_title("Grayscale")
axs[0,1].axis("off")

axs[0,2].imshow(rotated)
axs[0,2].set_title("Rotated 45°")
axs[0,2].axis("off")

axs[1,0].imshow(flipped)
axs[1,0].set_title("Flipped")
axs[1,0].axis("off")

axs[1,1].imshow(blurred)
axs[1,1].set_title("Blurred")
axs[1,1].axis("off")

axs[1,2].imshow(resized)
axs[1,2].set_title("Resized (300×300)")
axs[1,2].axis("off")

plt.tight_layout()
plt.show()

print("All images saved successfully!")
```

Output:

All images saved successfully!



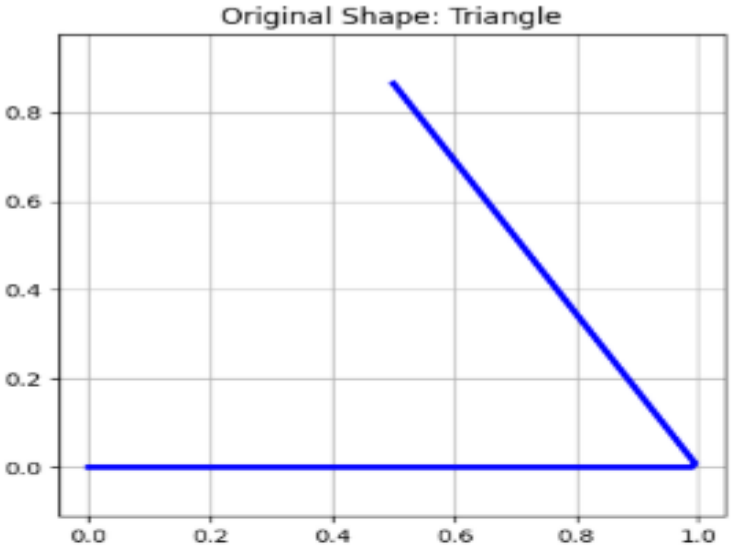
Experiment – 11

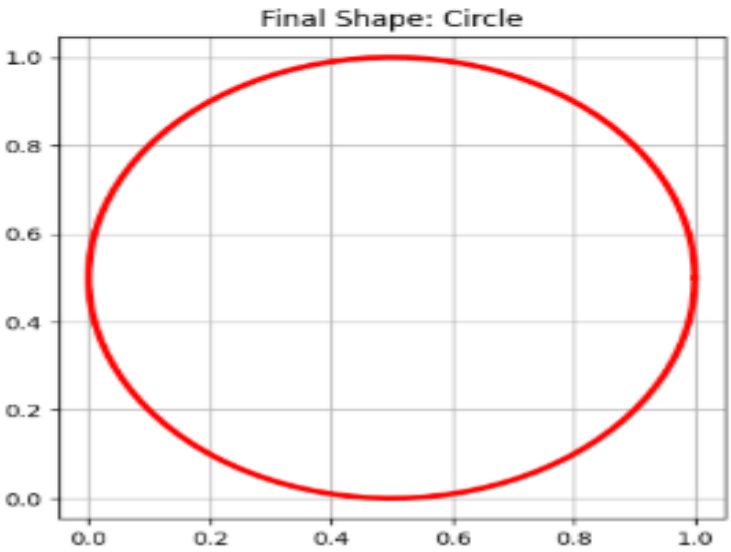
To produce animation effect of triangle transformed into square and then circle.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
n = 100

# TRIANGLE
# ----------------------------
triangle_x = np.interp(np.linspace(0, 1, n), [0, 0.5, 1], [0, 1, 0.5])
triangle_y = np.interp(np.linspace(0, 1, n), [0, 0.5, 1], [0, 0, np.sqrt(3)/2])

plt.figure(figsize=(5,5))
plt.plot(triangle_x, triangle_y, linewidth=3, color='blue')
plt.title("Original Shape: Triangle")
plt.axis("equal")
plt.grid(True)
plt.savefig("triangle.png")
plt.show()
# CIRCLE
# ----------------------------
theta = np.linspace(0, 2*np.pi, n)
circle_x = 0.5 + 0.5 * np.cos(theta)
circle_y = 0.5 + 0.5 * np.sin(theta)

plt.figure(figsize=(5,5))
plt.plot(circle_x, circle_y, linewidth=3, color='red')
plt.title("Final Shape: Circle")
plt.axis("equal")
plt.grid(True)
plt.savefig("circle.png")
plt.show()
```

Output:


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


Final Shape: Circle
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Experiment – 12

To create an animation of an arrow embedded into a circle revolving around its center.

Program:

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

def draw_arrow(ax, angle_deg, title):
    ax.set_title(title)
    ax.set_aspect('equal')

    # Show axis
    ax.set_xlim(0, 10)
    ax.set_ylim(0, 10)
    ax.grid(True)

    # Draw circle
    theta = np.linspace(0, 2*np.pi, 300)
    ax.plot(5 + 4*np.cos(theta), 5 + 4*np.sin(theta), linewidth=4)

    # Arrow base (horizontal to the right initially)
    start = np.array([2, 5])
    end = np.array([8, 5])

    # Rotation matrix
    angle = np.radians(angle_deg)
    R = np.array([[np.cos(angle), -np.sin(angle)],
                  [np.sin(angle),  np.cos(angle)]])

    # Rotate points around center (5,5)
    start_rot = R @ (start - np.array([5, 5])) + np.array([5, 5])
    end_rot = R @ (end - np.array([5, 5])) + np.array([5, 5])

    # Draw arrow
    ax.annotate("",
                xy=end_rot, xytext=start_rot,
                arrowprops=dict(arrowstyle="->", lw=4))

# -------------------------
# Plot two images
# -------------------------

fig = plt.figure(figsize=(10,5))
```

```
# 1. Normal arrow
ax1 = fig.add_subplot(121)
draw_arrow(ax1, angle_deg=0, title="Normal Arrow (With Axis)")

# 2. Rotated arrow (45 degrees)
ax2 = fig.add_subplot(122)
draw_arrow(ax2, angle_deg=45, title="Rotated Arrow (With Axis)")

plt.tight_layout()
plt.show()
```

Output: