# INDEX

| Sl No. | Name of The Experiment | Page No. |
|---|---|---|
| 01 | A program to implement the DDA (Digital Differential Analyzer) line drawing algorithm. | 2 - 3 |
| 02 | A program to implement Bresenham's line drawing algorithm. | 4 - 5 |
| 03 | A program to implement Bresenham's circle drawing algorithm. | 6 |
| 04 | A program to implement the Midpoint circle drawing algorithm. | 7 |
| 05 | A program to implement the Midpoint ellipse drawing algorithm. | 8 - 9 |
| 06 | A program to implement the Polygon (Rectangle) filling algorithm. | 10 |
| 07 | A program to implement the 2D transformation (Translation, Rotation). | 11 -12 |
| 08 | A program to implement the 2D transformation (Scaling, Translation). | 13 |
| 09 | A program to implement the composite (Translation, Rotation) transformation. | 14 |
| 10 | A program to implement the general point-to-point rotation of a triangle. | 15 |

## **01. Experiment Name:** A program to implement the DDA (Digital Differential Analyzer) line drawing algorithm.

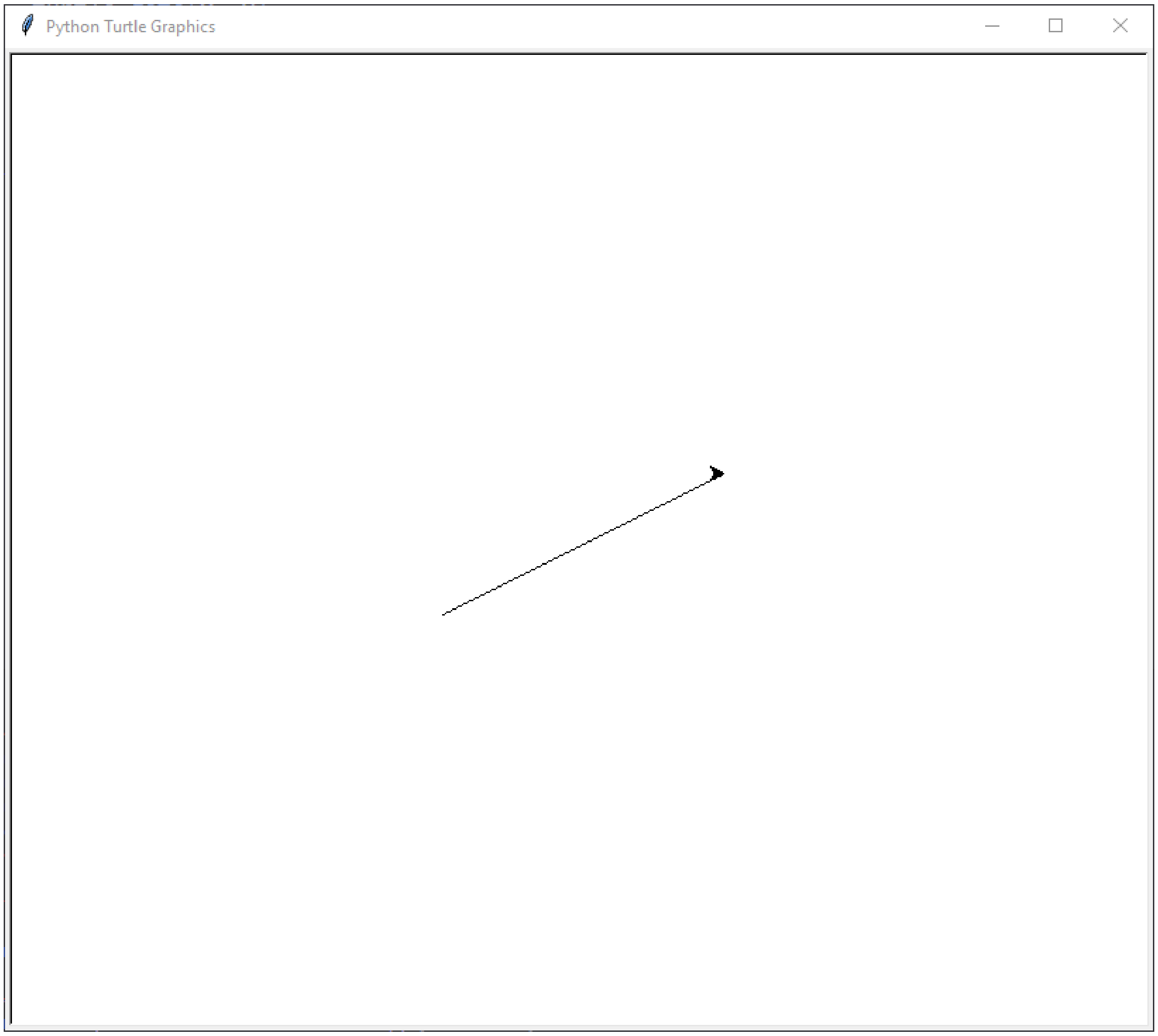
### Objective:

To implement the Digital Differential Analyzer (DDA) line drawing algorithm using Python and the Turtle graphics library, allowing accurate and incremental plotting of a straight line between two points on a graphical interface.

### Program:

```
import turtle

def dda(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    steps = int(max(abs(dx), abs(dy)))
    x_inc = dx / steps
    y_inc = dy / steps
    x, y = x1, y1

    turtle.penup()
    turtle.goto(x, y)
    turtle.pendown()
    for _ in range(steps):
        turtle.goto(round(x), round(y))
        x += x_inc
        y += y_inc

turtle.speed(0)
dda(-100, -50, 100, 50)
turtle.done()
```

Output:



## **02. Experiment Name:** A program to implement Bresenham's line drawing algorithm.

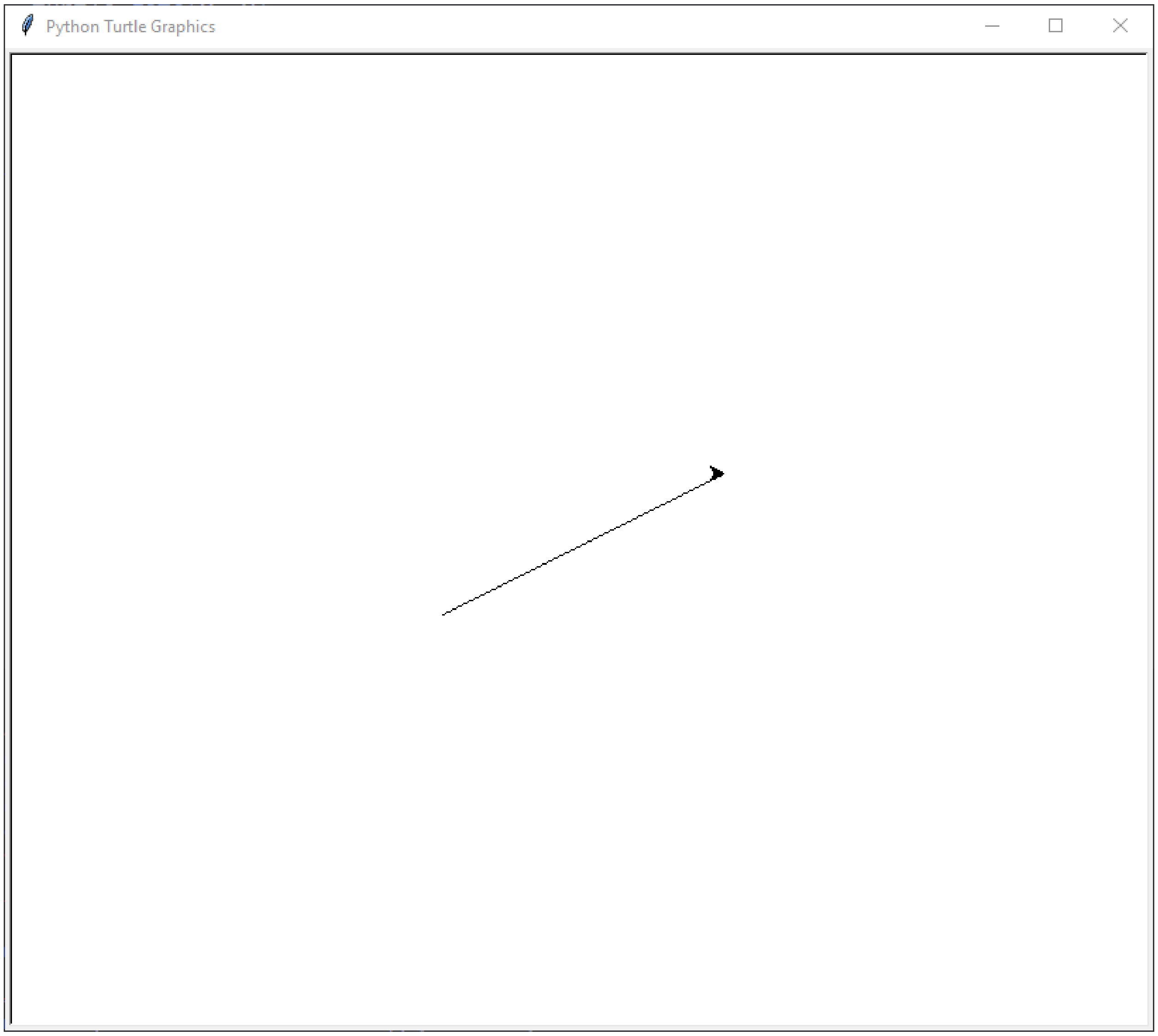
### Objective:

To implement Bresenham's line drawing algorithm using Python and the Turtle graphics library, enabling efficient and accurate rasterization of a straight line between two points using only integer calculations.

### Program:

```
import turtle

def bresenham(x1, y1, x2, y2):
    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
    x, y = x1, y1

    turtle.penup()
    turtle.goto(x, y)
    turtle.pendown()
    while True:
        turtle.goto(x, y)
        if x == x2 and y == y2:
            break
        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x += sx
        if e2 < dx:
            err += dx
            y += sy

turtle.speed(0)
bresenham(-100, 0, 100, 80)
turtle.done()
```

Output:

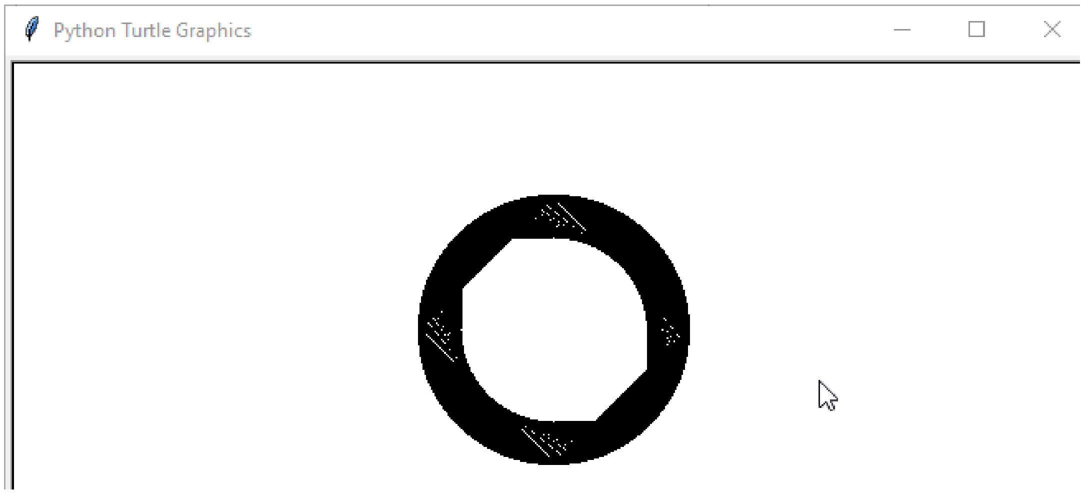


## **03. Experiment Name:** A program to implement Bresenham's circle drawing algorithm.

Program:

```
import turtle

def plot8(cx, cy, x, y):
    for dx, dy in
[(x,y),(y,x),(-x,y),(-y,x),(-x,-y),(-y,-x),(x,-y),(y,-x)]:
        turtle.goto(cx+dx, cy+dy); turtle.dot()

def bresenham_circle(cx, cy, r):
    x, y, d = 0, r, 3 - 2*r
    plot8(cx, cy, x, y)
    while x < y:
        if d < 0:
            d += 4*x + 6
        else:
            d += 4*(x-y) + 10
            y -= 1
        x += 1
        plot8(cx, cy, x, y)

turtle.speed(0)
bresenham_circle(0, 0, 80)
turtle.done()
```

Output:

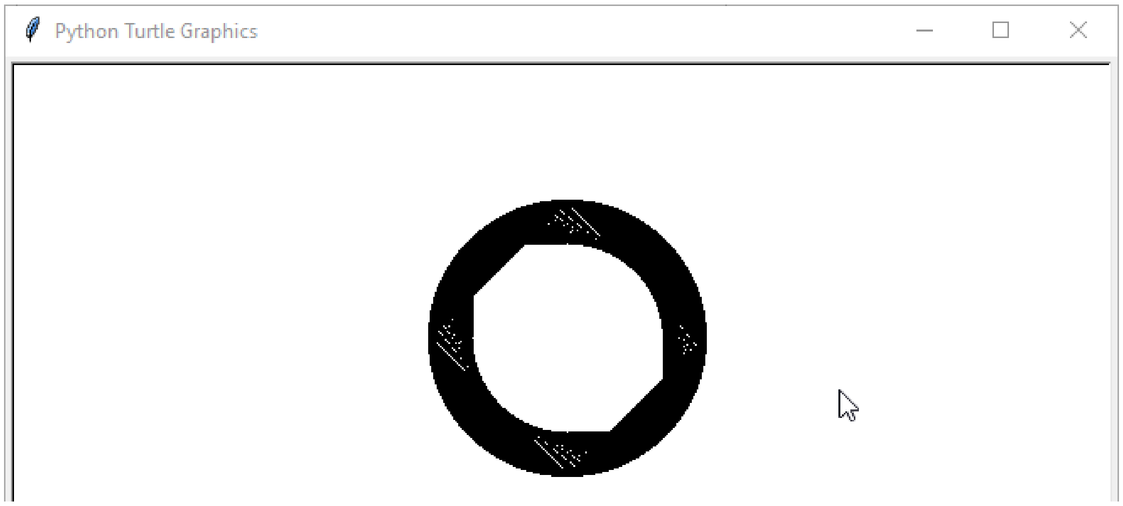


## **04. Experiment Name:** A program to implement the Midpoint circle drawing algorithm.

Program:

```
import turtle

def plot8(cx, cy, x, y):
    for dx, dy in
[(x,y),(y,x),(-x,y),(-y,x),(-x,-y),(-y,-x),(x,-y),(y,-x)]:
        turtle.goto(cx+dx, cy+dy); turtle.dot()

def bresenham_circle(cx, cy, r):
    x, y, d = 0, r, 3 - 2*r
    plot8(cx, cy, x, y)
    while x < y:
        if d < 0:
            d += 4*x + 6
        else:
            d += 4*(x-y) + 10
            y -= 1
        x += 1
        plot8(cx, cy, x, y)

turtle.speed(0)
bresenham_circle(0, 0, 80)
turtle.done()
```

Output:

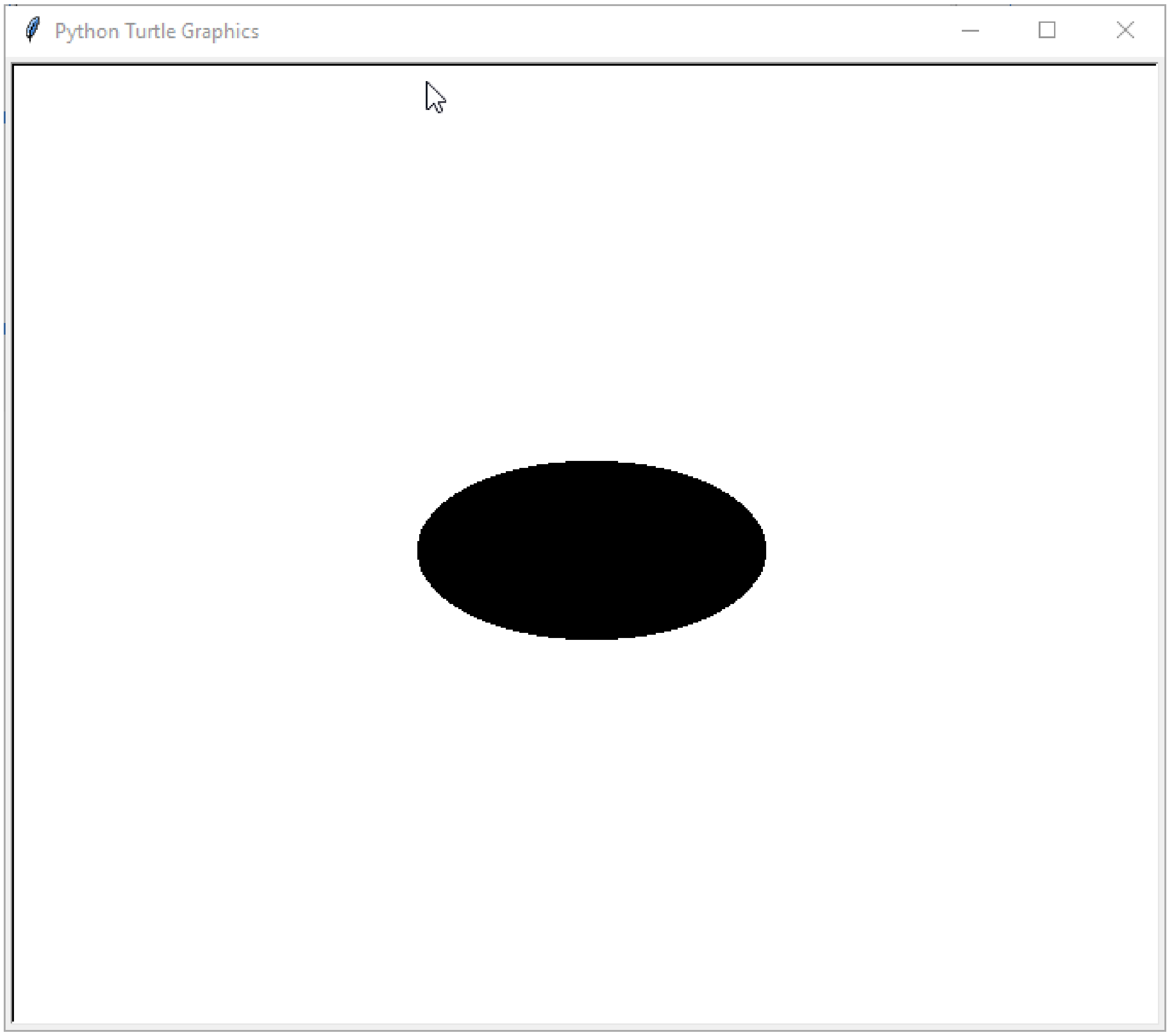


## **05. Experiment Name:** A program to implement the Midpoint ellipse drawing algorithm.

Program:

```
import turtle

def plot4(cx, cy, x, y):
    for dx, dy in [( x, y),(-x, y),(-x,-y),( x,-y)]:
        turtle.goto(cx+dx, cy+dy)
        turtle.dot()

def midpoint_ellipse(cx, cy, a, b):
    x, y = 0, b
    a2, b2 = a*a, b*b
    d1 = b2 - a2*b + 0.25*a2
    dx, dy = 2*b2*x, 2*a2*y

    turtle.penup()
    plot4(cx, cy, x, y)
    turtle.pendown()
    # Region 1
    while dx < dy:
        x += 1
        dx += 2*b2
        if d1 < 0:
            d1 += dx + b2
        else:
            y -= 1
            dy -= 2*a2
            d1 += dx - dy + b2
        plot4(cx, cy, x, y)
    # Region 2
    d2 = b2*(x+0.5)**2 + a2*(y-1)**2 - a2*b2
    while y > 0:
        y -= 1
        dy -= 2*a2
        if d2 > 0:
            d2 += a2 - dy
        else:
            x += 1
            dx += 2*b2
            d2 += dx - dy + a2
        plot4(cx, cy, x, y)

turtle.speed(0)
midpoint_ellipse(0, 0, 100, 50)
turtle.done()
```

Output:

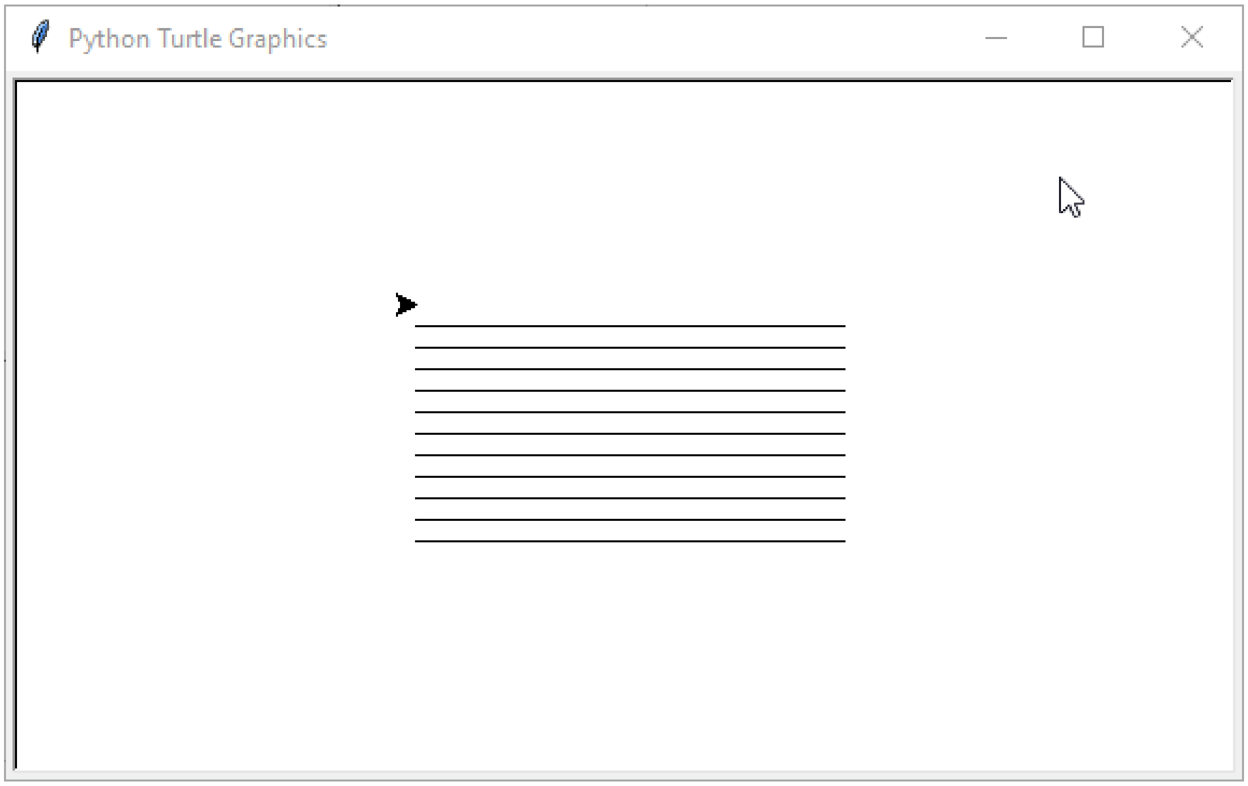


## **06. Experiment Name:** A program to implement the Polygon (Rectangle) filling algorithm.

Program:

```python
import turtle

def fill_rect(x0, y0, x1, y1, gap=10):
    y = min(y0, y1)
    while y <= max(y0, y1):
        turtle.penup(); turtle.goto(x0, y)
        turtle.pendown(); turtle.goto(x1, y)
        y += gap

turtle.speed(0)
fill_rect(-100, -50, 100, 50)
turtle.done()
```

Output:

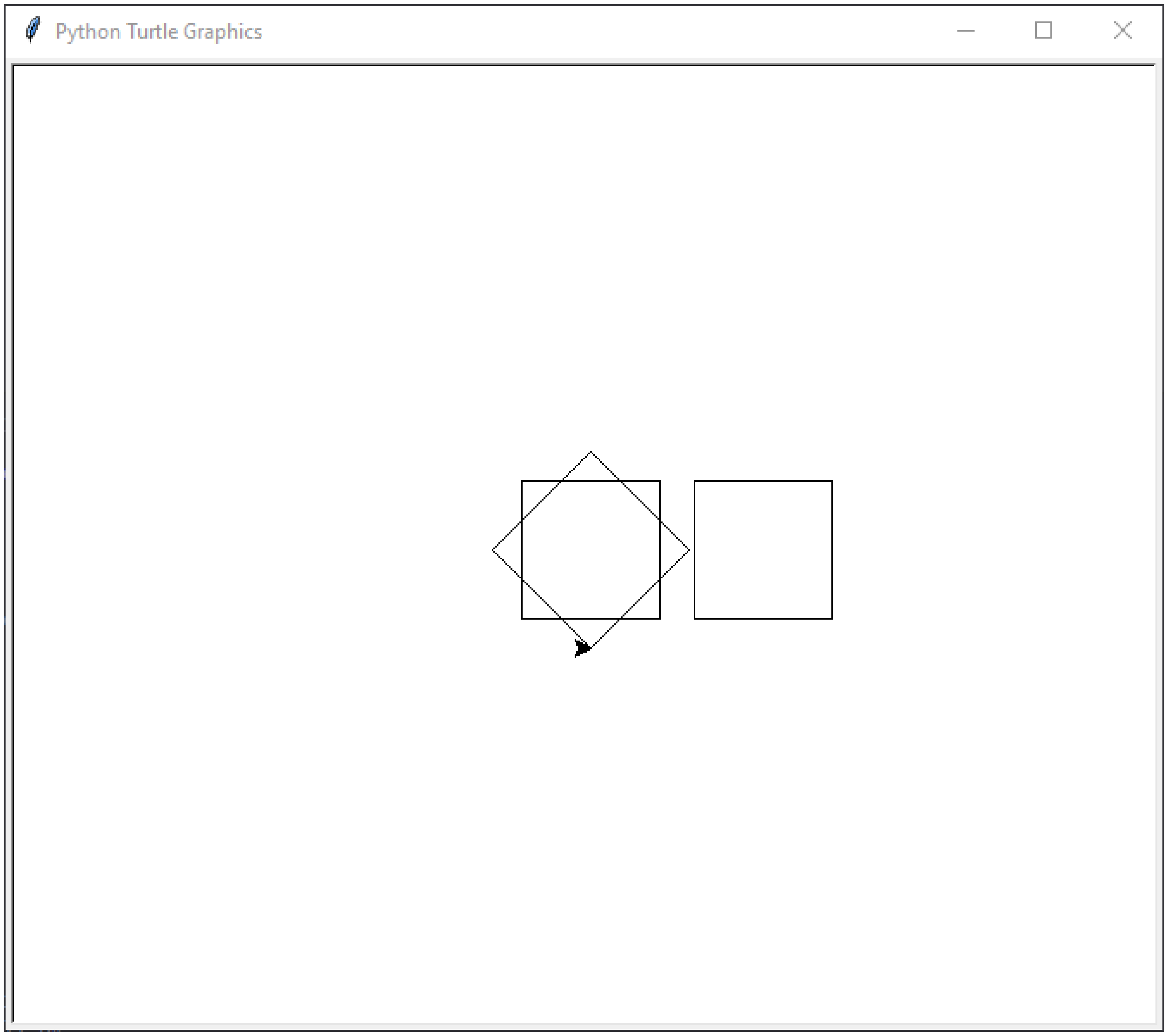


## 07. Experiment Name: A program to implement the 2D transformation (Translation, Rotation).

Program:

```python
import turtle, math

def draw(pts):
    turtle.penup()
    turtle.goto(pts[0])
    turtle.pendown()
    for x, y in pts[1:]+[pts[0]]:
        turtle.goto(x, y)

def translate(pts, dx, dy):
    return [(x+dx, y+dy) for x, y in pts]

def rotate(pts, ang, cx=0, cy=0):
    θ = math.radians(ang)
    out = []
    for x, y in pts:
        x0, y0 = x-cx, y-cy
        out.append((x0*math.cos(θ)-y0*math.sin(θ)+cx,
                    x0*math.sin(θ)+y0*math.cos(θ)+cy))
    return out

turtle.speed(0)
square = [(-40,-40),(40,-40),(40,40),(-40,40)]
draw(square)
draw(translate(square, 100, 0))
draw(rotate(square, 45))
turtle.done()
```

Output:

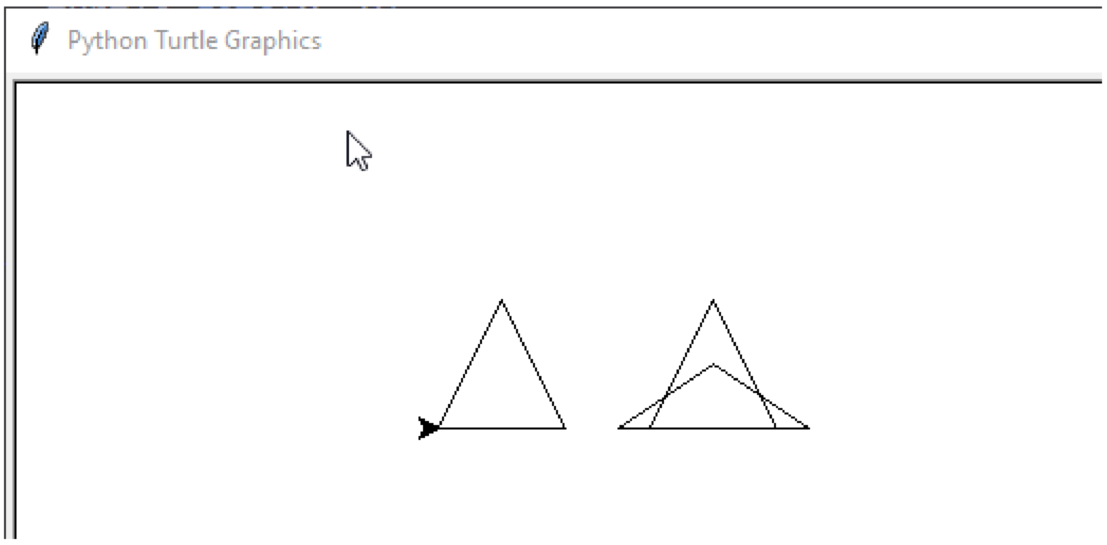


## 08. Experiment Name: A program to implement the 2D transformation (Scaling, Translation).

Program:

```
import turtle

def draw(pts):
    turtle.penup()
    turtle.goto(pts[0])
    turtle.pendown()
    for x, y in pts[1:]+[pts[0]]:
        turtle.goto(x, y)

def scale(pts, sx, sy, cx=0, cy=0):
    return [((x-cx)*sx+cx, (y-cy)*sy+cy) for x, y in pts]

def translate(pts, dx, dy):
    return [(x+dx, y+dy) for x, y in pts]

turtle.speed(0)
tri = [(-30,0),(30,0),(0,60)]
draw(tri)
draw(scale(tri, 1.5, 0.5))
draw(translate(tri, -100, 0))
turtle.done()
```

Output:

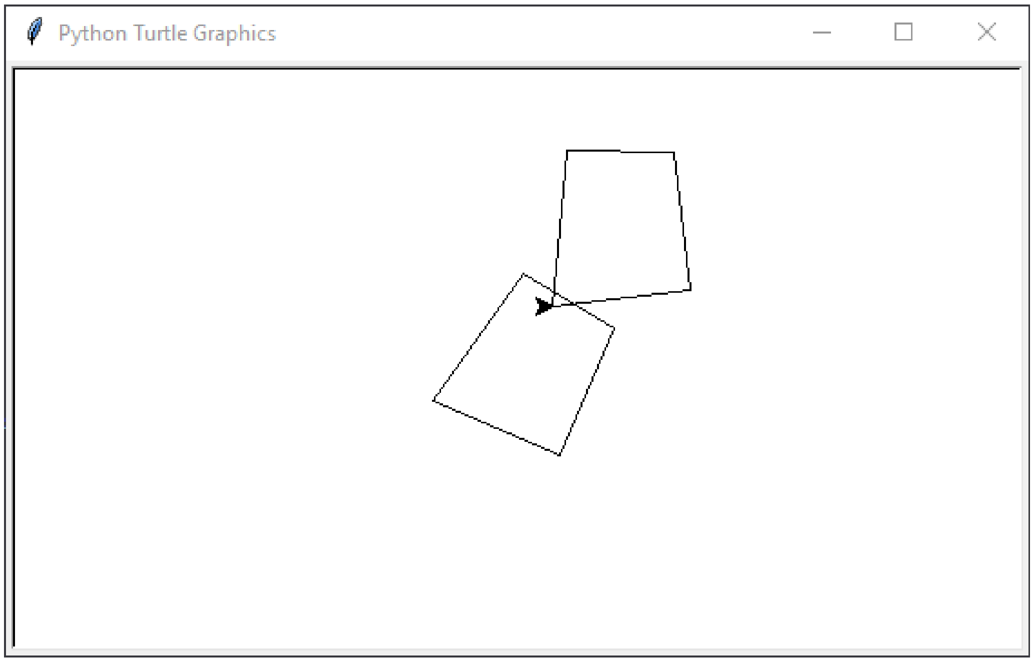


## 09. Experiment Name: A program to implement the composite (Translation, Rotation) transformation.

Program:

```
import turtle, math

def draw(pts):
    turtle.penup(); turtle.goto(pts[0]); turtle.pendown()
    for p in pts[1:]+[pts[0]]: turtle.goto(p)

def composite(pts, dx, dy, ang):
    moved = [(x+dx, y+dy) for x,y in pts]
    θ = math.radians(ang)
    return [(x*math.cos(θ)-y*math.sin(θ),
             x*math.sin(θ)+y*math.cos(θ))
            for x,y in moved]

turtle.speed(0)
poly = [(-50,-20),(0,50),(50,20),(20,-50)]
draw(poly)
draw(composite(poly, 80, 40, 30))
turtle.done()
```

Output:

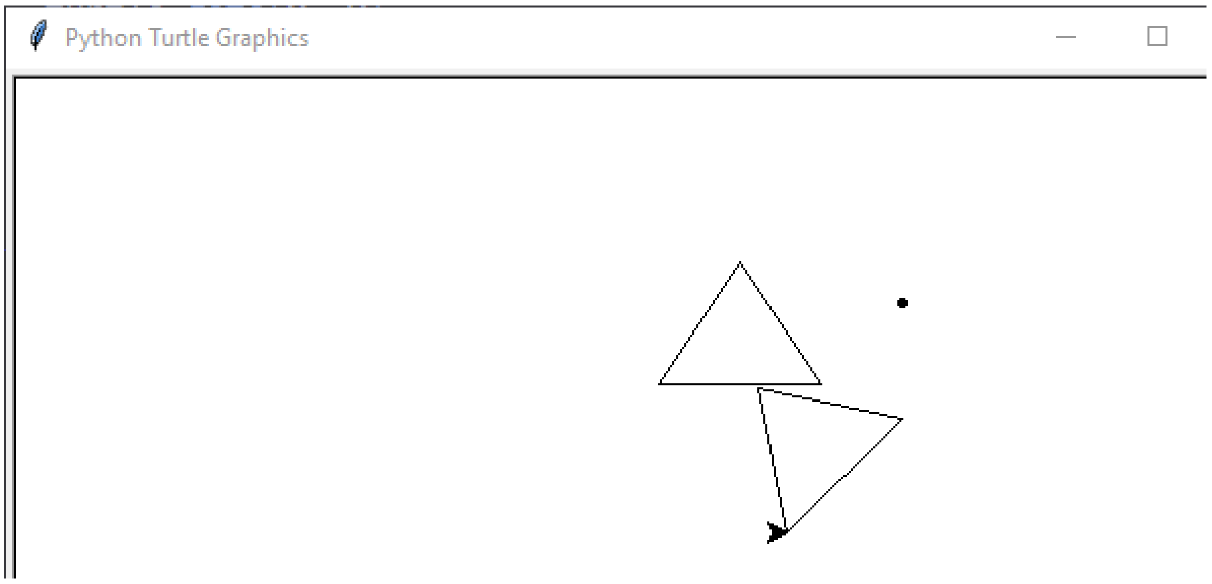


**10. Experiment Name:** A program to implement the general point-to-point rotation of a triangle.

Program:

```
import turtle, math

def draw(pts):
    turtle.penup(); turtle.goto(pts[0]); turtle.pendown()
    for p in pts[1:]+[pts[0]]: turtle.goto(p)

def rotate_about(pts, ang, px, py):
    θ = math.radians(ang)
    return [(
        (x-px)*math.cos(θ)-(y-py)*math.sin(θ)+px,
        (x-px)*math.sin(θ)+(y-py)*math.cos(θ)+py
    ) for x,y in pts]

turtle.speed(0)
tri = [(0,0),(80,0),(40,60)]
draw(tri)
pivot = (120, 40)
turtle.goto(pivot); turtle.dot()
draw(rotate_about(tri, 45, *pivot))
turtle.done()
```

Output:


Python Turtle Graphics