EXP 1:

```
import turtle

screen = turtle.Screen()
screen.setup(700, 700)
screen.title("Simple Line and Circle")
screen.setworldcoordinates(-350, -350, 350, 350)

pen = turtle.Turtle()
pen.hideturtle()
pen.speed(0)
pen.penup()

PIXEL_SIZE = 8

def set_pixel(x, y, color="black"):
    pen.goto(x, y)
    pen.dot(PIXEL_SIZE, color)

def draw_line(x1, y1, x2, y2, color="blue"):
    dx, dy = x2 - x1, y2 - y1
    steps = int(max(abs(dx), abs(dy)))
    x_inc, y_inc = dx / steps, dy / steps
    x, y = x1, y1
    for _ in range(steps + 1):
        set_pixel(round(x), round(y), color)
        x += x_inc
        y += y_inc

def draw_circle(cx, cy, r, color="green"):
    x, y, p = 0, r, 1 - r
```

```
    while x <= y:
        for px, py in [
            (cx + x, cy + y), (cx - x, cy + y),
            (cx + x, cy - y), (cx - x, cy - y),
            (cx + y, cy + x), (cx - y, cy + x),
            (cx + y, cy - x), (cx - y, cy - x)
        ]:
            set_pixel(px, py, color)
        x += 1
        if p < 0:
            p += 2 * x + 1
        else:
            y -= 1
            p += 2 * (x - y) + 1

draw_line(-150, -150, 150, 150)
draw_circle(0, 0, 100)

screen.exitonclick()
```

EXP 2:

```
def plot_line_bresenham(x1, y1, x2, y2, grid_size=10):

    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]



    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1
    err = dx - dy
```

```
    while True:
        if 0 <= x1 < grid_size and 0 <= y1 < grid_size:
            grid[y1][x1] = 'X'

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

        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x1 += sx
        if e2 < dx:
            err += dx
            y1 += sy

    for row in grid[::-1]:
        print(" ".join(row))


def plot_line_dda(x1, y1, x2, y2, grid_size=10):

    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]

    dx = x2 - x1
    dy = y2 - y1
    steps = max(abs(dx), abs(dy))
    x_inc = dx / steps
    y_inc = dy / steps

    x, y = x1, y1
```

```
    for _ in range(steps + 1):
        if 0 <= int(x) < grid_size and 0 <= int(y) < grid_size:
            grid[int(y)][int(x)] = 'X'
        x += x_inc
        y += y_inc

    for row in grid[::-1]:
        print(" ".join(row))


def draw_path(start_points, end_points, algorithm='bresenham', grid_size=10):
    for i in range(len(start_points)):
        x1, y1 = start_points[i]
        x2, y2 = end_points[i]

        print(f"Drawing path from ({x1}, {y1}) to ({x2}, {y2}) using {algorithm.upper()}:\n")

        if algorithm == 'bresenham':
            plot_line_bresenham(x1, y1, x2, y2, grid_size)
        elif algorithm == 'dda':
            plot_line_dda(x1, y1, x2, y2, grid_size)
        else:
            print("Invalid algorithm. Choose 'bresenham' or 'dda'.")

        print("\n" + "-" * 40 + "\n")


start_points = [(1, 1), (7, 3), (3, 7)]
end_points = [(7, 3), (3, 7), (8, 8)]

draw_path(start_points, end_points, algorithm='bresenham')
```

EXP 3:

```
def plot_circle_bresenham(xc, yc, r, grid_size=10):
    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]
    x, y = 0, r
    p = 3 - 2 * r

    while 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:
            if 0 <= px < grid_size and 0 <= py < grid_size:
                grid[py][px] = 'X'

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

    for row in grid[::-1]:
        print(" ".join(row))



def plot_circle_midpoint(xc, yc, r, grid_size=10):
    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]
```

```
    x, y = 0, r
    p = 1 - r

    while 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:
            if 0 <= px < grid_size and 0 <= py < grid_size:
                grid[py][px] = 'X'

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

    for row in grid[::-1]:
        print(" ".join(row))



def draw_circle(center, radius, algorithm='bresenham', grid_size=10):
    xc, yc = center
    print(f"Drawing circle at center ({xc}, {yc}) with radius {radius} using {algorithm.upper()}
algorithm:\n")

    if algorithm == 'bresenham':
```

```
        plot_circle_bresenham(xc, yc, radius, grid_size)
    elif algorithm == 'midpoint':
        plot_circle_midpoint(xc, yc, radius, grid_size)
    else:
        print("Invalid algorithm. Please choose 'bresenham' or 'midpoint'.")

    print("\n" + "-" * 40 + "\n")
center = (5, 5)
radius = 3
draw_circle(center, radius, algorithm='bresenham')
```

EXP 4:

```
import numpy as np

# ---------- 2D TRANSFORMATIONS ----------
def translate_2d(img, tx, ty):
    M = np.array([[1, 0, tx], [0, 1, ty], [0, 0, 1]])
    return apply_2d(img, M)

def scale_2d(img, sx, sy):
    M = np.array([[sx, 0, 0], [0, sy, 0], [0, 0, 1]])
    return apply_2d(img, M)

def rotate_2d(img, angle):
    a = np.radians(angle)
    M = np.array([[np.cos(a), -np.sin(a), 0],
                  [np.sin(a), np.cos(a), 0],
                  [0, 0, 1]])
    return apply_2d(img, M)

def shear_2d(img, shx, shy):
```

```
    M = np.array([[1, shx, 0], [shy, 1, 0], [0, 0, 1]])
    return apply_2d(img, M)

def apply_2d(img, M):
    out = np.zeros_like(img)
    rows, cols = img.shape
    for x in range(rows):
        for y in range(cols):
            p = np.dot(M, [x, y, 1])
            nx, ny = int(p[0]), int(p[1])
            if 0 <= nx < rows and 0 <= ny < cols:
                out[nx, ny] = img[x, y]
    return out

# ---------- 3D TRANSFORMATIONS ----------
def translate_3d(obj, tx, ty, tz):
    M = np.array([[1, 0, 0, tx],
                  [0, 1, 0, ty],
                  [0, 0, 1, tz],
                  [0, 0, 0, 1]])
    return apply_3d(obj, M)

def scale_3d(obj, sx, sy, sz):
    M = np.array([[sx, 0, 0, 0],
                  [0, sy, 0, 0],
                  [0, 0, sz, 0],
                  [0, 0, 0, 1]])
    return apply_3d(obj, M)

def rotate_3d(obj, ax, ay, az):
    ax, ay, az = np.radians([ax, ay, az])
```

```
    Rx = np.array([[1, 0, 0, 0],
                   [0, np.cos(ax), -np.sin(ax), 0],
                   [0, np.sin(ax), np.cos(ax), 0],
                   [0, 0, 0, 1]])
    Ry = np.array([[np.cos(ay), 0, np.sin(ay), 0],
                   [0, 1, 0, 0],
                   [-np.sin(ay), 0, np.cos(ay), 0],
                   [0, 0, 0, 1]])
    Rz = np.array([[np.cos(az), -np.sin(az), 0, 0],
                   [np.sin(az), np.cos(az), 0, 0],
                   [0, 0, 1, 0],
                   [0, 0, 0, 1]])
    M = Rx @ Ry @ Rz
    return apply_3d(obj, M)

def apply_3d(obj, M):

    return np.copy(obj)

# ---------- DEMO ----------
img = np.array([[0,0,0,0,0],
                [0,1,1,1,0],
                [0,1,1,1,0],
                [0,0,0,0,0]])

print("Original:\n", img)
print("\nTranslated:\n", translate_2d(img, 1, 1))
print("\nScaled:\n", scale_2d(img, 2, 2))
print("\nRotated 45°:\n", rotate_2d(img, 45))
print("\nSheared:\n", shear_2d(img, 1, 0))
```

EXP 5:

```
import matplotlib.pyplot as plt

X_MIN, Y_MIN, X_MAX, Y_MAX = 0, 0, 10, 10
INSIDE, LEFT, RIGHT, BOTTOM, TOP = 0, 1, 2, 4, 8

def compute_code(x, y):
    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):
    c1, c2 = compute_code(x1, y1), compute_code(x2, y2)
    while True:
        if not (c1 | c2): return (x1, y1, x2, y2)
        elif c1 & c2: return None
        c_out = c1 or c2
        if c_out & TOP:    x, y = x1 + (x2-x1)*(Y_MAX-y1)/(y2-y1), Y_MAX
        elif c_out & BOTTOM: x, y = x1 + (x2-x1)*(Y_MIN-y1)/(y2-y1), Y_MIN
        elif c_out & RIGHT:  y, x = y1 + (y2-y1)*(X_MAX-x1)/(x2-x1), X_MAX
        elif c_out & LEFT:   y, x = y1 + (y2-y1)*(X_MIN-x1)/(x2-x1), X_MIN
        if c_out == c1: x1, y1, c1 = x, y, compute_code(x, y)
        else: x2, y2, c2 = x, y, compute_code(x, y)

def plot_line_clip(x1, y1, x2, y2):
    plt.figure()
    plt.xlim(-1, 11); plt.ylim(-1, 11)
```

```
    plt.plot([X_MIN, X_MAX, X_MAX, X_MIN, X_MIN],
             [Y_MIN, Y_MIN, Y_MAX, Y_MAX, Y_MIN], 'k-', lw=2)

    plt.plot([x1, x2], [y1, y2], 'b--', label="Original Line")

    res = cohen_sutherland_clip(x1, y1, x2, y2)
    if res:
        x1, y1, x2, y2 = res
        plt.plot([x1, x2], [y1, y2], 'r-', lw=2, label="Clipped Line")
    else:
        print("Line is outside the window.")
    plt.title("Cohen–Sutherland Line Clipping")
    plt.grid(True); plt.legend(); plt.show()

plot_line_clip(2, 3, 12, 5)
```

EXP 6:

```
import pygame, sys, math
pygame.init()

WIDTH, HEIGHT = 600, 400
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Interactive Bézier Curve")
WHITE, RED, BLUE = (255,255,255), (255,0,0), (0,0,255)

points = [(100, 300), (150, 50), (450, 50), (500, 300)]
drag = None

def C(n, k): return math.comb(n, k)

def bezier(t):
```

```
    n = len(points) - 1
    x = sum(C(n,i)*(t**i)*((1-t)**(n-i))*points[i][0] for i in range(n+1))
    y = sum(C(n,i)*(t**i)*((1-t)**(n-i))*points[i][1] for i in range(n+1))
    return int(x), int(y)

def draw():
    screen.fill(WHITE)
    pygame.draw.lines(screen, RED, False, points, 2)
    for p in points: pygame.draw.circle(screen, RED, p, 6)
    for i in range(101):
        t = i / 100
        pygame.draw.circle(screen, BLUE, bezier(t), 2)
    pygame.display.flip()

while True:
    for e in pygame.event.get():
        if e.type == pygame.QUIT: pygame.quit(); sys.exit()
        if e.type == pygame.MOUSEBUTTONDOWN:
            for i, p in enumerate(points):
                if abs(e.pos[0]-p[0]) < 8 and abs(e.pos[1]-p[1]) < 8:
                    drag = i
        if e.type == pygame.MOUSEBUTTONUP: drag = None
        if e.type == pygame.MOUSEMOTION and drag is not None:
            points[drag] = e.pos
    draw()
```