1.

```
import turtle

def bresenham_line(x1, y1, x2, y2):
    # Calculate the deltas
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)

    # Determine the step direction for each axis
    x_step = 1 if x1 < x2 else -1
    y_step = 1 if y1 < y2 else -1

    # Initialize the error term
    error = 2 * dy - dx

    # Initialize the line points
    line_points = []

    # Start at the first point
    x, y = x1, y1

    # Draw the line
    for _ in range(dx + 1):
        # Add the current point to the line
        line_points.append((x, y))

        # Update the error term and adjust the coordinates
        if error > 0:
            y += y_step
            error -= 2 * dx
        error += 2 * dy
        x += x_step

    return line_points

# Example usage
turtle.setup(500, 500)
turtle.speed(0)  # Fastest drawing speed

x1, y1 = 100, 100
x2, y2 = 400, 300

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

# Draw the line
turtle.penup()
turtle.goto(x1, y1)
turtle.pendown()
for x, y in line_points:
    turtle.goto(x, y)

turtle.exitonclick()
```

2. import turtle
import math

```
# Set up the turtle screen
screen = turtle.Screen()
screen.bgcolor("white")

# Create a turtle instance
t = turtle.Turtle()
t.speed(1)  # Set the drawing speed (1 is slowest, 10 is fastest)
t.pensize(2)  # Set the pen size

# Define a function to draw a rectangle
def draw_rectangle(x, y, width, height, color):
    t.penup()
    t.goto(x, y)
    t.pendown()
    t.color(color)
    for _ in range(2):
        t.forward(width)
        t.left(90)
        t.forward(height)
        t.left(90)

# Define a function to draw a circle
def draw_circle(x, y, radius, color):
    t.penup()
    t.goto(x, y - radius)
    t.pendown()
    t.color(color)
    t.circle(radius)

# Define a function to translate a 2D object
def translate(x, y, dx, dy):
    t.penup()
    t.goto(x + dx, y + dy)
    t.pendown()

# Define a function to rotate a 2D object
def rotate(x, y, angle):
    t.penup()
    t.goto(x, y)
    t.setheading(angle)
    t.pendown()

# Define a function to scale a 2D object
def scale(x, y, sx, sy):
    t.penup()
    t.goto(x * sx, y * sy)
    t.pendown()

# Draw a rectangle
draw_rectangle(-200, 0, 100, 50, "blue")

# Translate the rectangle
translate(-200, 0, 200, 0)
draw_rectangle(0, 0, 100, 50, "blue")

# Rotate the rectangle
rotate(0, 0, 45)
draw_rectangle(0, 0, 100, 50, "blue")

# Scale the rectangle
scale(0, 0, 2, 2)
draw_rectangle(0, 0, 100, 50, "blue")

# Draw a circle
draw_circle(100, 100, 50, "red")

# Translate the circle
translate(100, 100, 200, 0)
draw_circle(300, 100, 50, "red")

# Rotate the circle
rotate(300, 100, 45)
draw_circle(300, 100, 50, "red")

# Scale the circle
scale(300, 100, 2, 2)
draw_circle(600, 200, 50, "red")

# Keep the window open until it's closed
turtle.done()
```

3.
```
from vpython import canvas, box, cylinder, vector, color, rate

# Create a 3D canvas
scene = canvas(width=800, height=600, background=color.white)

# Define a function to draw a cuboid
def draw_cuboid(pos, length, width, height, color):
    cuboid = box(pos=vector(*pos), length=length, width=width, height=height, color=color)
    return cuboid

# Define a function to draw a cylinder
def draw_cylinder(pos, radius, height, color):
    cyl = cylinder(pos=vector(*pos), radius=radius, height=height, color=color)
    return cyl

# Define a function to translate a 3D object
def translate(obj, dx, dy, dz):
    obj.pos += vector(dx, dy, dz)

# Define a function to rotate a 3D object
def rotate(obj, angle, axis):
    obj.rotate(angle=angle, axis=vector(*axis))

# Define a function to scale a 3D object
def scale(obj, sx, sy, sz):
    obj.size = vector(obj.size.x * sx, obj.size.y * sy, obj.size.z * sz)

# Draw a cuboid
cuboid = draw_cuboid((-2, 0, 0), 2, 2, 2, color.blue)

# Translate the cuboid
translate(cuboid, 4, 0, 0)

# Rotate the cuboid
rotate(cuboid, angle=45, axis=(0, 1, 0))

# Scale the cuboid
scale(cuboid, 1.5, 1.5, 1.5)

# Draw a cylinder
cylinder = draw_cylinder((2, 2, 0), 1, 10, color.red)

# Translate the cylinder
translate(cylinder, 0, -2, 0)

# Rotate the cylinder
rotate(cylinder, angle=30, axis=(1, 0, 0))

# Scale the cylinder
scale(cylinder, 1.5, 1.5, 1.5)

while True:
    rate(30)  # Set the frame rate to 30 frames per second
```

5.
```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *

pygame.init()
display = (800, 600)
pygame.display.set_mode(display, DOUBLEBUF | OPENGL)

gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
glTranslatef(0.0, 0.0, -5)

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()

    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT)

    glBegin(GL_QUADS)
    glColor3f(1, 0, 0)
    glVertex3f(1, 1, -1)
    glVertex3f(-1, 1, -1)
    glVertex3f(-1, 1, 1)
    glVertex3f(1, 1, 1)

    glColor3f(0, 1, 0)
    glVertex3f(1, -1, 1)
    glVertex3f(-1, -1, 1)
    glVertex3f(-1, -1, -1)
    glVertex3f(1, -1, -1)

    glColor3f(0, 0, 1)
    glVertex3f(1, 1, 1)
    glVertex3f(-1, 1, 1)
    glVertex3f(-1, -1, 1)
    glVertex3f(1, -1, 1)

    glColor3f(1, 1, 0)
    glVertex3f(1, -1, -1)
    glVertex3f(-1, -1, -1)
    glVertex3f(-1, 1, -1)
    glVertex3f(1, 1, -1)

    glColor3f(0, 1, 1)
    glVertex3f(-1, 1, 1)
    glVertex3f(-1, 1, -1)
    glVertex3f(-1, -1, -1)
    glVertex3f(-1, -1, 1)

    glColor3f(1, 0, 0)
    glVertex3f(1, 1, -1)
    glVertex3f(-1, 1, -1)
    glVertex3f(-1, 1, 1)
    glVertex3f(1, 1, 1)

    glColor3f(0, 1, 0)
    glVertex3f(1, -1, 1)
    glVertex3f(-1, -1, 1)
    glVertex3f(-1, -1, -1)
    glVertex3f(1, -1, -1)

    glColor3f(0, 0, 1)
    glVertex3f(1, 1, 1)
    glVertex3f(-1, 1, 1)
    glVertex3f(-1, -1, 1)
    glVertex3f(1, -1, 1)

    glColor3f(1, 1, 0)
    glVertex3f(1, -1, -1)
    glVertex3f(-1, -1, -1)
    glVertex3f(-1, 1, -1)
    glVertex3f(1, 1, -1)

    glColor3f(0, 1, 1)
    glVertex3f(-1, 1, 1)
    glVertex3f(-1, 1, -1)
    glVertex3f(-1, -1, -1)
    glVertex3f(-1, -1, 1)

    glVertex3f(1, -1, -1)
    glVertex3f(1, -1, 1)
    glVertex3f(1, 1, 1)
    glVertex3f(1, 1, -1)

    glEnd()

    # Swap the buffers
    pygame.display.flip()
    pygame.time.wait(10)
```

6.

```
import pygame
import sys

# Initialize Pygame
pygame.init()

# Set up the screen
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Simple Animation")

# Colors
WHITE = (255, 255, 255)
RED = (255, 0, 0)

# Set initial position and velocity of the square
x = 50
y = 50
velocity = 5

# Main loop
while True:
    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    # Clear the screen
    screen.fill(WHITE)

    # Move the square
    x += velocity

    # If the square goes off the screen, wrap around
    if x > width:
        x = 0

    # Draw the square
    pygame.draw.rect(screen, RED, (x, y, 50, 50))

    # Update the display
    pygame.display.flip()

    # Add a small delay to control the speed of animation
    pygame.time.delay(50)
```

4.

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

def apply_transformation(vertices,
transformation_matrix):
    return np.dot(transformation_matrix, vertices.T).T

def plot_object(vertices, title, ax):
    ax.plot(np.vstack([vertices, vertices[0]])[:, 0],
np.vstack([vertices, vertices[0]])[:, 1], marker='o')
    ax.set_title(title)
    ax.set_aspect('equal')

rectangle = np.array([[1, 1], [3, 1], [3, 2], [1, 2]])
ones = np.ones((rectangle.shape[0], 1))
rectangle_homogeneous = np.hstack([rectangle, ones])

translation_matrix = np.array([[1, 0, 2], [0, 1, 3], [0, 0, 1]])
rotation_matrix = np.array([[np.cos(np.radians(45)), -
np.sin(np.radians(45)), 0], [np.sin(np.radians(45)),
np.cos(np.radians(45)), 0], [0, 0, 1]])
scaling_matrix = np.array([[2, 0, 0], [0, 1.5, 0], [0, 0, 1]])

translated_rectangle =
apply_transformation(rectangle_homogeneous,
translation_matrix)
rotated_rectangle =
apply_transformation(rectangle_homogeneous,
rotation_matrix)
scaled_rectangle =
apply_transformation(rectangle_homogeneous,
scaling_matrix)

fig, axs = plt.subplots(2, 2, figsize=(10, 10))
plot_object(rectangle, "Original Rectangle", axs[0, 0])
plot_object(translated_rectangle[:, :2], "Translated
Rectangle", axs[0, 1])
plot_object(rotated_rectangle[:, :2], "Rotated Rectangle",
axs[1, 0])
plot_object(scaled_rectangle[:, :2], "Scaled Rectangle",
axs[1, 1])
plt.tight_layout()
plt.show()
```