```
1)
import turtle
def bresenham_line(x1, y1, x2, y2):
   dx = abs(x2 - x1)
   dy = abs(y2 - y1)
   x_step = 1 if x1 < x2 else -1
   y_step = 1 if y1 < y2 else -1
   error = 2 * dy - dx
   line_points = []
   x, y = x1, y1
 for _ in range(dx + 1):
      line_points.append((x, y))
      if error > 0:
         y += y_step
         error -= 2 * dx
      error += 2 * dy
      x += x_step
   return line_points
turtle.setup(500, 500)
turtle.speed(0)
x1, y1 = 100, 100
x2, y2 = 400, 300
line_points = bresenham_line(x1, y1, x2, y2)
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
screen = turtle.Screen()
screen.bgcolor("white")
t = turtle.Turtle()
t.speed(1)
t.pensize(2)
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)
def draw_circle(x, y, radius, color):
   t.penup()
   t.goto(x, y - radius)
   t.pendown()
   t.color(color)
   t.circle(radius)
def translate(x, y, dx, dy):
   t.penup()
   t.goto(x + dx, y + dy)
   t.pendown()
def rotate(x, y, angle):
   t.penup()
   t.goto(x, y)
   t.setheading(angle)
   t.pendown()
def scale(x, y, sx, sy):
   t.penup()
   t.goto(x * sx, y * sy)
   t.pendown()
draw_rectangle(-200, 0, 100, 50, "blue")
translate(-200, 0, 200, 0)
draw_rectangle(0, 0, 100, 50, "blue")
rotate(0, 0, 45)
draw_rectangle(0, 0, 100, 50, "blue")
scale(0, 0, 2, 2)
draw_rectangle(0, 0, 100, 50, "blue")
draw_circle(100, 100, 50, "red")
translate(100, 100, 200, 0)
draw_circle(300, 100, 50, "red")
rotate(300, 100, 45)
draw_circle(300, 100, 50, "red")
scale(300, 100, 2, 2)
draw_circle(600, 200, 50, "red")
turtle.done()
```

```
3)
from vpython import canvas, box, cylinder, vector,
color, rate
scene = canvas(width=800, height=600,
background=color.white)
def draw_cuboid(pos, length, width, height, color):
   cuboid = box(pos=vector(*pos), length=length,
width=width, height=height, color=color)
   return cuboid
def draw_cylinder(pos, radius, height, color):
   cyl = cylinder(pos=vector(*pos), radius=radius,
height=height, color=color)
   return cyl
def translate(obj, dx, dy, dz):
   obj.pos += vector(dx, dy, dz)
def rotate(obj, angle, axis):
   obj.rotate(angle=angle, axis=vector(*axis))
def scale(obj, sx, sy, sz):
   obj.size = vector(obj.size.x * sx, obj.size.y * sy,
obj.size.z * sz)
cuboid = draw_cuboid((-2, 0, 0), 2, 2, 2, color.blue)
translate(cuboid, 4, 0, 0)
rotate(cuboid, angle=45, axis=(0, 1, 0))
scale(cuboid, 1.5, 1.5, 1.5)
cyl = draw_cylinder((2, 2, 0), 1, 10, color.red)
translate(cyl, 0, -2, 0)
rotate(cyl, angle=30, axis=(1, 0, 0))
scale(cyl, 1.5, 1.5, 1.5)
while True:
   rate(30)
```

```
4)
import cv2
import numpy as np
canvas_width = 500
canvas_height = 500
canvas = np.ones((canvas_height, canvas_width, 3),
dtype=np.uint8) * 255
obj_points = np.array([[100, 100], [200, 100], [200,
200], [100, 200]], dtype=np.int32)
translation_matrix = np.float32([[1, 0, 100], [0, 1,
50]])
rotation_matrix = cv2.getRotationMatrix2D((150,
150), 45, 1)
scaling_matrix = np.float32([[1.5, 0, 0], [0, 1.5, 0]])
translated_obj =
np.array([np.dot(translation_matrix, [x, y, 1])[:2] for
x, y in obj_points], dtype=np.int32)
rotated_obj = np.array([np.dot(rotation_matrix, [x,
y, 1])[:2] for x, y in translated_obj], dtype=np.int32)
scaled_obj = np.array([np.dot(scaling_matrix, [x, y,
1])[:2] for x, y in rotated_obj], dtype=np.int32)
cv2.polylines(canvas, [obj_points], True, (0, 0, 0), 2)
cv2.polylines(canvas, [translated_obj], True, (0, 255,
0), 2)
cv2.polylines(canvas, [rotated_obj], True, (255, 0, 0),
2)
cv2.polylines(canvas, [scaled_obj], True, (0, 0, 255),
2)
cv2.imshow("2D Transformations", canvas)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
6)import pygame
import random
pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Animation Effects")
BLACK, WHITE, RED, GREEN, BLUE = (0, 0, 0), (255,
255, 255), (255, 0, 0), (0, 255, 0), (0, 0, 255)
objects = [{"x": random.randint(50, 750), "y":
random.randint(50, 550), "radius":
random.randint(10, 30),
        "color": random.choice([RED, GREEN, BLUE]),
"speed_x": random.randint(-5, 5), "speed_y":
random.randint(-5, 5)}
        for _ in range(10)]
clock = pygame.time.Clock()
running = True
while running:
   for event in pygame.event.get():
      if event.type == pygame.QUIT:
         running = False
    screen.fill(WHITE)
    for obj in objects:
      obj["x"] += obj["speed_x"]
      obj["y"] += obj["speed_y"]
      if obj["x"] - obj["radius"] < 0 or obj["x"] +
obj["radius"] > 800:
         obj["speed_x"] = -obj["speed_x"]
      if obj["y"] - obj["radius"] < 0 or obj["y"] +
obj["radius"] > 600:
         obj["speed_y"] = -obj["speed_y"]
      pygame.draw.circle(screen, obj["color"],
(obj["x"], obj["y"]), obj["radius"])
pygame.display.flip()
   clock.tick(60)
pygame.quit()
```

```
5)import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
import numpy as np
pygame.init()
display_width = 800
display_height = 600
display = pygame.display.set_mode((display_width,
display_height), DOUBLEBUF | OPENGL)
pygame.display.set_caption("3D Transformations")
glClearColor(0.0, 0.0, 0.0, 1.0)
glEnable(GL_DEPTH_TEST)
glMatrixMode(GL_PROJECTION)
gluPerspective(45, (display_width / display_height),
0.1, 50.0)
glMatrixMode(GL_MODELVIEW)
vertices = np.array([
   [-1, -1, -1],
   [1, -1, -1],
   [1, 1, -1],
   [-1, 1, -1],
   [-1, -1, 1],
   [1, -1, 1],
   [1, 1, 1],
   [-1, 1, 1]
], dtype=np.float32)
edges = np.array([
   [0, 1], [1, 2], [2, 3], [3, 0],
   [4, 5], [5, 6], [6, 7], [7, 4],
   [0, 4], [1, 5], [2, 6], [3, 7]
], dtype=np.uint32)
translation_matrix = np.eye(4, dtype=np.float32)
translation_matrix[3, :3] = [0, 0, -5]
rotation_matrix = np.eye(4, dtype=np.float32)
scaling_matrix = np.eye(4, dtype=np.float32)
scaling_matrix[0, 0] = 1.5
scaling_matrix[1, 1] = 1.5
scaling_matrix[2, 2] = 1.5
running = True
angle = 0
while running:
   for event in pygame.event.get():
      if event.type == pygame.QUIT:
         running = False
   glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT)
   glLoadIdentity()
   glMultMatrixf(translation_matrix)
   glRotatef(angle, 1, 1, 0)
   glMultMatrixf(rotation_matrix)
   glMultMatrixf(scaling_matrix)
   glBegin(GL_LINES)
   for edge in edges:
      for vertex in edge:
         glVertex3fv(vertices[vertex])
   glEnd()
    angle += 1
   pygame.display.flip()
   pygame.time.wait(10)
 pygame.quit()
```