Q1. Develop a program to draw a line using Bresenham's line drawing technique

```python
import matplotlib.pyplot as plt
def bresenham_line(x0, y0, x1, y1):
    dx, dy = x1 - x0, y1 - y0
    p = 2 * dy - dx
    points = [(x0, y0)]
    while x0 < x1:
        x0 += 1
        if p < 0:
            p += 2 * dy
        else:
            y0 += 1
            p += 2 * dy - 2 * dx
            points.append((x0, y0))
    x_points, y_points = zip(*points)
    plt.plot(x_points, y_points, 'o-')
    plt.title("Bresenham's Line Algorithm")
    plt.xlabel("X coordinate")
    plt.ylabel("Y coordinate")
    plt.grid(True)
    plt.show()
bresenham_line(20, 10, 30, 18)
```

Q2. Develop a program to demonstrate basic geometric operations on the 2D object

```python
import numpy as np
import matplotlib.pyplot as plt
triangle = np.array([[0, 0], [1, 0], [0.5, 1], [0, 0]])
translated_triangle = triangle + [1, 2]
plt.plot(*triangle.T, 'r-', label='Original Triangle')
plt.plot(*translated_triangle.T, 'g-', label='Translated Triangle')
plt.axis('equal')
plt.legend()
plt.show()
```

Q3. Develop a program to demonstrate basic geometric operations on the 3D object

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

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

scaled_cube = cube * [0.5, 0.5, 0.5]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(cube[:, 0], cube[:, 1], cube[:, 2], 'r-', label='Original')
ax.plot(scaled_cube[:, 0], scaled_cube[:, 1], scaled_cube[:, 2], 'g-',
label='Scaled')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()
plt.show()
```

Q4. Develop a program to demonstrate 2D transformation on basic objects

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

def translate(coords, tx, ty):
    translation_matrix = np.array([[1, 0, tx],
                                   [0, 1, ty],
                                   [0, 0, 1]])
    return np.dot(translation_matrix, coords)

def rotate(coords, theta):
    rad = np.radians(theta)
    rotation_matrix = np.array([[np.cos(rad), -np.sin(rad), 0],
                                [np.sin(rad), np.cos(rad), 0],
                                [0, 0, 1]])
    return np.dot(rotation_matrix, coords)

def scale(coords, sx, sy):
    scaling_matrix = np.array([[sx, 0, 0],
                               [0, sy, 0],
                               [0, 0, 1]])
    return np.dot(scaling_matrix, coords)

def plot_polygon(coords, color='blue', label='Original'):
    xs = coords[0, :]
    ys = coords[1, :]
    xs = np.append(xs, xs[0])
    ys = np.append(ys, ys[0])
    plt.plot(xs, ys, marker='o', color=color, label=label)

triangle = np.array([[0, 1, 0.5, 0],
                     [0, 0, 1, 0],
                     [1, 1, 1, 1]])

triangle_translated = translate(triangle, 1, 1)
triangle_rotated = rotate(triangle, 90)
triangle_scaled = scale(triangle, 1, 2)

plt.figure(figsize=(10, 8))
plot_polygon(triangle, color='blue', label='Original')
plot_polygon(triangle_translated, color='green', label='Translated')
plot_polygon(triangle_rotated, color='red', label='Rotated')
plot_polygon(triangle_scaled, color='purple', label='Scaled')
plt.axis('equal')
plt.legend()
plt.grid(True)
plt.title('2D Geometric Transformations')
plt.show()
```

Q5. Develop a program to demonstrate 3D transformation on 3D objects

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

cube = np.array([[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 0, 0], [0, 0, 1], [0, 1,
1], [1, 1, 1], [1, 0, 1]])
faces = [[0, 1, 2, 3], [4, 5, 6, 7], [0, 1, 5, 4], [2, 3, 7, 6], [0, 3, 7, 4],
[1, 2, 6, 5]]

def plot_cube(ax, vertices, color='blue', alpha=0.25):
    ax.add_collection3d(Poly3DCollection([vertices[f] for f in faces],
facecolors=color, alpha=alpha, linewidths=1, edgecolors='r'))
    ax.set(xlim=[0, 2], ylim=[0, 2], zlim=[0, 2], box_aspect=[1, 1, 1])

def transform(v, t=[0, 0, 0], s=[1, 1, 1], angle=0, axis='z'):
    v = (v + t) * s
    c, s = np.cos(angle), np.sin(angle)
    R = np.array({'x': [[1, 0, 0], [0, c, -s], [0, s, c]], 'y': [[c, 0, s],
[0, 1, 0], [-s, 0, c]], 'z': [[c, -s, 0], [s, c, 0], [0, 0, 1]]}[axis])
    return v @ R.T

fig, (ax1, ax2) = plt.subplots(1, 2, subplot_kw={'projection': '3d'})
plot_cube(ax1, cube)
plot_cube(ax2, transform(cube, [1, 1, 1], [0.5, 0.5, 0.5], np.pi/4), 'green')
ax1.set_title('Original Cube')
ax2.set_title('Transformed Cube')
plt.show()
```

Q6. Develop a program to demonstrate Animation effects on simple objects.

```python
import turtle, colorsys

t = turtle.Turtle()
turtle.bgcolor('black')
t.speed(0)

for i in range(360):
    t.color(colorsys.hsv_to_rgb(i/70, 1, 0.8))
    t.left(1)
    t.forward(1)
    for _ in range(2):
        t.left(2)
        t.circle(90)
```

Q7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

```python
import cv2
image = cv2.imread(r"image.jpg")
h, w = image.shape[:2]
cv2.imshow('Upper Left', image[:h//2, :w//2])
cv2.imshow('Upper Right', image[:h//2, w//2:])
cv2.imshow('Lower Left', image[h//2:, :w//2])
cv2.imshow('Lower Right', image[h//2:, w//2:])
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q8. Write a program to show rotation, scaling, and translation on an image.

```python
import cv2
import imutils
def transform_image(image_path):
    img = cv2.imread(image_path)
    rotated = imutils.rotate_bound(img, 45)
    scaled = cv2.resize(img, None, fx=0.5, fy=0.5)
    translated = imutils.translate(img, 100, 50)
    transformations = {'Original': img, 'Rotated': rotated, 'Scaled': scaled,
'Translated': translated}
    for title, image in transformations.items():
        cv2.imshow(title, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
transform_image(r"image.jpg")
```

Q9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.

```
import cv2
from skimage.feature import local_binary_pattern
import numpy as np

image = cv2.imread(r"image.jpeg")
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray_image, 100, 300)
lbp = local_binary_pattern(gray_image, 24, 3, method='uniform')
lbp = np.uint8((lbp / np.max(lbp)) * 150)

cv2.imshow('Original Image', image)
cv2.imshow('Edges', edges)
cv2.imshow('Texture (LBP)', lbp)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q10. Write a program to blur and smoothing an image.

```
import cv2
image = cv2.imread(r"image.jpeg")
blurred = cv2.GaussianBlur(image, (21, 21), 0)
bilateral = cv2.bilateralFilter(image, 15, 75, 75)
median = cv2.medianBlur(image, 15)
cv2.imshow('Original', image)
cv2.imshow('Gaussian Blurred Image', blurred)
cv2.imshow('Bilateral Image', bilateral)
cv2.imshow('Median Blurred Image', median)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q11. Write a program to contour an image.

```
import cv2
img = cv2.imread(r"image.jpg")
contours, _ = cv2.findContours(cv2.Canny(cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY), 30, 200), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0, 255, 0), 1)
cv2.imshow('Contours', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q12. Write a program to detect a face/s in an image.

```
import face_recognition
import cv2

def detect_faces(image_path):
    img = face_recognition.load_image_file(image_path)
    for (top, right, bottom, left) in face_recognition.face_locations(img):
        cv2.rectangle(img, (left, top), (right, bottom), (0, 255, 0), 2)

    cv2.imshow('Faces Detected', cv2.cvtColor(img, cv2.COLOR_RGB2BGR))
    cv2.waitKey(0)
    cv2.destroyAllWindows()

detect_faces(r'image.jpg')
```