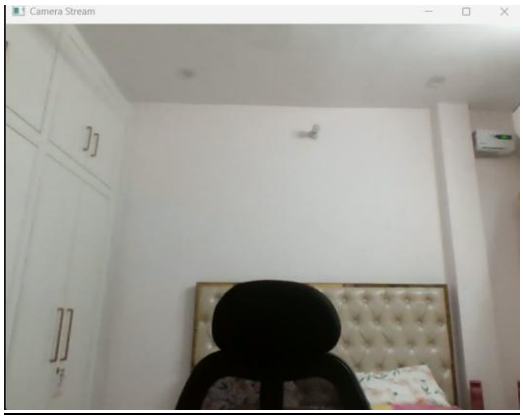
COMPUTER VISION

CODE 1:

```
import cv2

# For USB webcam (index 0 = first camera)
cap = cv2.VideoCapture(0)

# For IP camera (replace with your IP stream URL)
# cap = cv2.VideoCapture("rtsp://username:password@ip:554/stream")

while True:
    ret, frame = cap.read()
    if not ret:
        break

    cv2.imshow("Camera Stream", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

cap.release()
cv2.destroyAllWindows()
```

OUTPUT: Live stream



OBSERVATIONS:

1 The program turns on a camera

2 It keeps taking pictures from the camera continuously

3 Each picture is shown on your screen in a window

4 You can stop the program by pressing a key like q

5 After stopping it closes the camera and the window properly

---

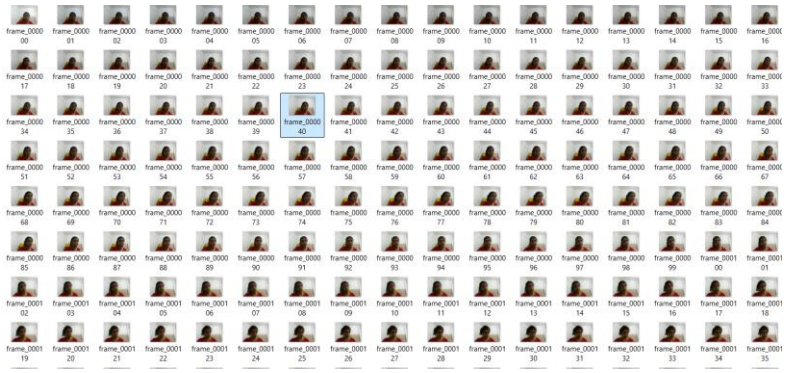
CODE 2

```
import cv2
import os

cap = cv2.VideoCapture(0)

# Create output folder
os.makedirs("frames", exist_ok=True)
frame_count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Show stream
    cv2.imshow("Camera Stream", frame)

    # Save frame
    filename = f"frames/frame_{frame_count:06d}.jpg"
    cv2.imwrite(filename, frame)
    frame_count += 1

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
```

cap.release()

cv2.destroyAllWindows()

OUTPUT: Live camera feed + saved frames



OBSERVATIONS:

1 The program turns on a camera
2 It captures frames continuously from the camera
3 Each frame is shown on screen
4 Each frame is saved in a folder named frames
5 You can stop the program by pressing a key like q
6 After stopping it releases the camera and closes windows

CODE 3

from PIL import Image

import cv2

import numpy as np

Open AVIF image using Pillow

pil_img = Image.open("C://Users//HP-PC//OneDrive//Documents//apple.avif")

Convert to OpenCV format

img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)

cv2.imshow("My Image", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

OUTPUT: Display a single image

OBSERVATIONS:

1 The program loads an image from disk
2 It checks if the image exists
3 Shows the image in a window
4 Waits until a key is pressed to close
5 Closes the window after key press

CODE 4

```
from PIL import Image

import pillow_avif  # make sure you installed pillow-avif-plugin

import cv2

import numpy as np


# Load AVIF image using Pillow

pil_img = Image.open(r"C:\Users\HP-PC\OneDrive\Documents\apple.avif")
```

```
# Convert to OpenCV format
img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)

if img is None:
    print("Error: Could not read image.")
else:
    # Flip vertically (0), horizontally (1), or both (-1)
    flip_vertical = cv2.flip(img, 0)
    flip_horizontal = cv2.flip(img, 1)
    flip_both = cv2.flip(img, -1)

    # Show results
    cv2.imshow("Original", img)
    cv2.imshow("Flipped Vertically", flip_vertical)
    cv2.imshow("Flipped Horizontally", flip_horizontal)
    cv2.imshow("Flipped Both", flip_both)

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

OUTPUT: Original and flipped images

OBSERVATIONS:

1 The program loads an image

2 It creates vertical, horizontal, and both-direction flips
3 Shows original and flipped images
4 Waits for a key press to close
5 Closes all windows after key press

CODE 5

```
from PIL import Image

import pillow_avif  # make sure pillow-avif-plugin is installed

import cv2

import numpy as np


# Load AVIF image using Pillow

pil_img = Image.open(r"C:\Users\HP-PC\OneDrive\Documents\apple.avif")


# Convert to OpenCV format

img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)


# Check if the image loaded correctly

if img is None:

   print("Error: Could not read image.")

   exit()


# Resize image (width=300, height=300)

resized = cv2.resize(img, (300, 300))


# Show both

cv2.imshow("Original", img)

cv2.imshow("Resized", resized)
```

```
# Wait for a key press
cv2.waitKey(0)
cv2.destroyAllWindows()

# Optional: Save the resized image
cv2.imwrite("resized_output.jpg", resized)
```

OUTPUT: Original and resized images

OBSERVATIONS:

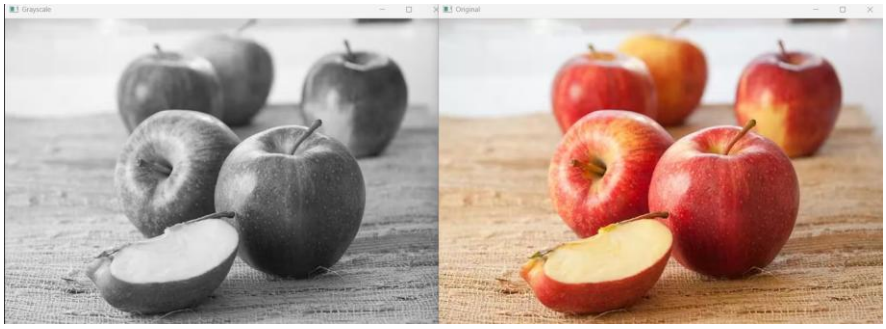
1 Loads an image from disk
2 Checks if image loaded correctly
3 Resizes the image to 300x300 pixels
4 Shows original and resized images
5 Saves the resized image to disk
6 Closes windows after key press

CODE 6

```
from PIL import Image
import pillow_avif  # make sure pillow-avif-plugin is installed
import cv2
import numpy as np
```

```
# Load AVIF image using Pillow
pil_img = Image.open(r"C:\Users\HP-PC\OneDrive\Documents\apple.avif")

# Convert to OpenCV format
img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)

# Check if image loaded correctly
if img is None:
    print("Error: Could not read image.")
    exit()

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Show both
cv2.imshow("Original", img)
cv2.imshow("Grayscale", gray)

# Wait until a key is pressed
cv2.waitKey(0)
cv2.destroyAllWindows()

# Optional: Save grayscale image
cv2.imwrite("grayscale_output.jpg", gray)
```

OUTPUT: grayscale images



OBSERVATIONS:

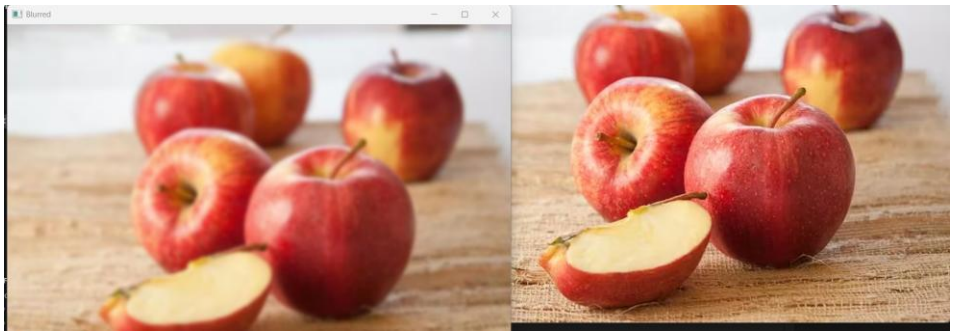
1 Loads an image from disk
2 Converts the image to grayscale
3 Shows original and grayscale images
4 Optionally saves the grayscale image
5 Waits for a key press then closes all windows

CODE 7

```
from PIL import Image

import pillow_avif  # make sure pillow-avif-plugin is installed

import cv2

import numpy as np


# Load AVIF image using Pillow

pil_img = Image.open(r"C:\Users\HP-PC\OneDrive\Documents\apple.avif")


# Convert to OpenCV format

img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)


# Check if image loaded correctly

if img is None:

   print("Error: Could not read image.")

   exit()
```

```
# Apply Gaussian Blur (15x15 kernel)
blur = cv2.GaussianBlur(img, (15, 15), 0)

# Show both
cv2.imshow("Original", img)
cv2.imshow("Blurred", blur)

# Wait for key press
cv2.waitKey(0)
cv2.destroyAllWindows()

# Optional: Save the blurred image
cv2.imwrite("blurred_output.jpg", blur)
```

OUTPUT: Original and blurred images



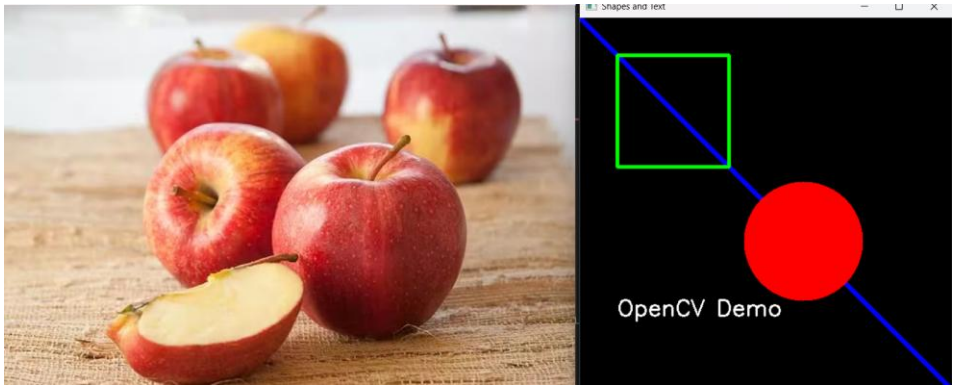
OBSERVATIONS:

1 Loads an image
2 Applies Gaussian blur with a 15x15 kernel
3 Shows original and blurred images
4 Optionally saves the blurred image
5 Waits for a key press then closes all windows

CODE 8

```
from PIL import Image
```

```
import pillow_avif  # make sure pillow-avif-plugin is installed
import cv2
import numpy as np

# Load AVIF image using Pillow
pil_img = Image.open(r"C:\Users\HP-PC\OneDrive\Documents\apple.avif")

# Convert to OpenCV format
img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)

# Create a blank canvas (500x500) if you want to draw shapes on top
canvas = np.zeros((500, 500, 3), dtype="uint8")

# Draw shapes on canvas
cv2.line(canvas, (0, 0), (500, 500), (255, 0, 0), 5)
cv2.rectangle(canvas, (50, 50), (200, 200), (0, 255, 0), 3)
cv2.circle(canvas, (300, 300), 80, (0, 0, 255), -1)

# Add text
cv2.putText(canvas, "OpenCV Demo", (50, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)

# Show results
cv2.imshow("Original Image", img)
cv2.imshow("Shapes and Text", canvas)

cv2.waitKey(0)
cv2.destroyAllWindows()
```
OUTPUT: Image with shapes and text



OBSERVATIONS:

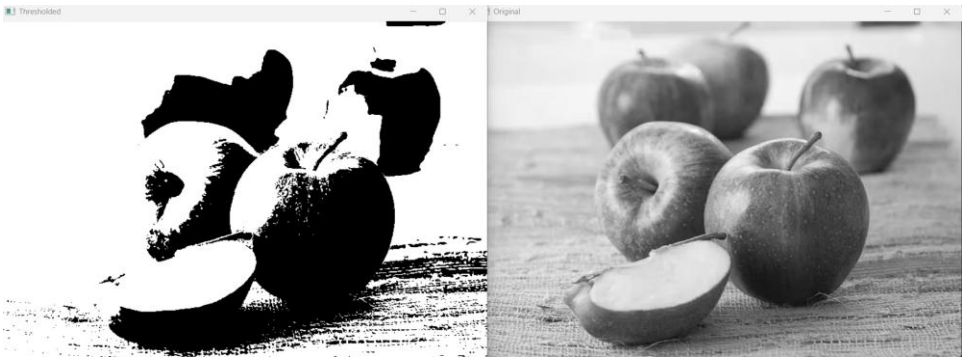
1 Creates a black blank image
2 Draws a line, rectangle, and circle
3 Adds text on the image
4 Displays the image
5 Closes window after key press

CODE 9

```
from PIL import Image

import pillow_avif  # make sure pillow-avif-plugin is installed

import cv2

import numpy as np


# Load AVIF image using Pillow

pil_img = Image.open(r"C:\Users\HP-PC\OneDrive\Documents\apple.avif")


# Convert to OpenCV format

img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)


# Convert to grayscale

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


# Apply binary threshold

_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
```

```
# Show results
cv2.imshow("Original", gray)
cv2.imshow("Thresholded", thresh)

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

OUTPUT: Original and thresholded images



OBSERVATIONS:

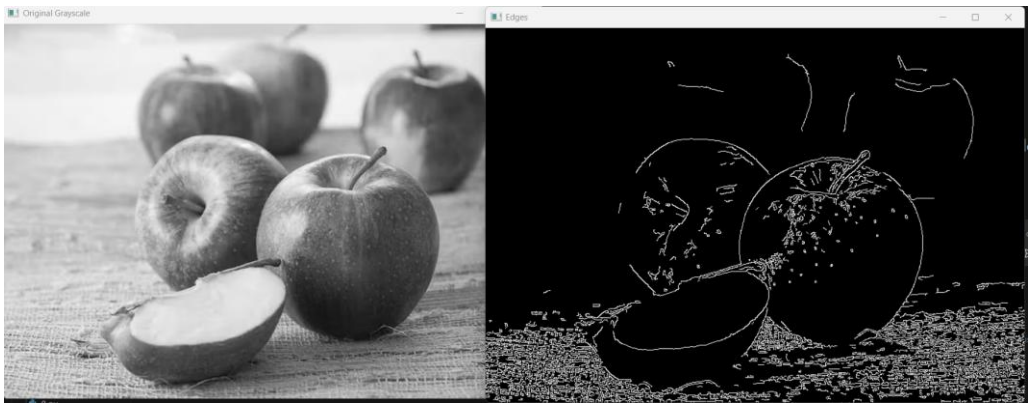
1 Loads a grayscale image
2 Applies binary thresholding to convert image into black and white
3 Shows original and thresholded images
4 Waits for a key press then closes all windows

CODE 10

```
from PIL import Image
import pillow_avif  # make sure pillow-avif-plugin is installed
import cv2
import numpy as np

# Load AVIF image using Pillow
pil_img = Image.open(r"C:\Users\HP-PC\OneDrive\Documents\apple.avif")
```

```
# Convert to OpenCV format
img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply Canny edge detection
edges = cv2.Canny(gray, 100, 200)

# Show results
cv2.imshow("Original Grayscale", gray)
cv2.imshow("Edges", edges)

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

OUTPUT: Edge-detected image



OBSERVATIONS:

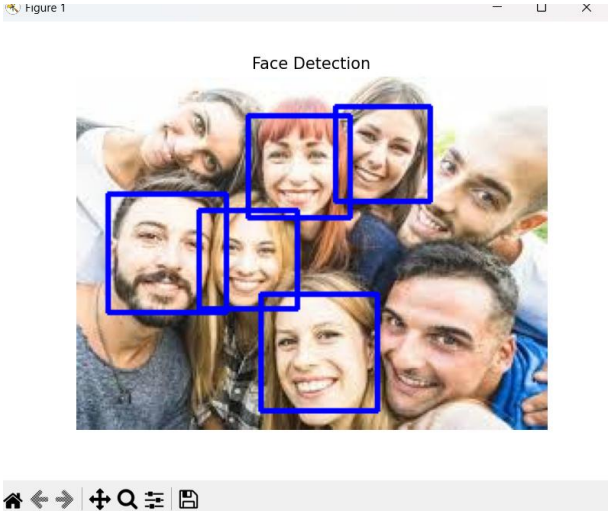
1 Loads a grayscale image
2 Detects edges using Canny edge detector
3 Shows edges in a window
4 Waits for a key press then closes all windows

CODE 11

```
import cv2
import matplotlib.pyplot as plt

# Load pre-trained classifier
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")

# Load image (use your file path)
file_path = r"C:\Users\HP-PC\OneDrive\Documents\faces.jpeg"
img = cv2.imread(file_path)

if img is None:
   print("Error: Could not read image.")
   exit()

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect faces
faces = face_cascade.detectMultiScale(gray, 1.1, 4)

# Draw rectangles around faces
for (x, y, w, h) in faces:
   cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Convert BGR to RGB for Matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
# Display the image
plt.imshow(img_rgb)
plt.axis('off')
plt.title("Face Detection")
plt.show()
```

OUTPUT: Image with faces detected



OBSERVATIONS:

1 Loads an image and Haar cascade classifier for face detection
2 Converts image to grayscale
3 Detects faces in the image
4 Draws rectangles around detected faces
5 Shows the result and closes window after key press

CODE 12

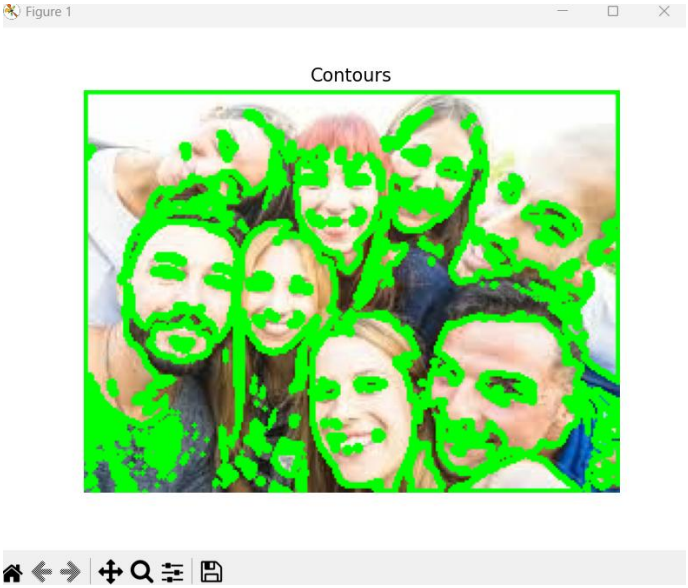
```
import cv2
import matplotlib.pyplot as plt
import os

# Load image
file_path = r"C:\Users\HP-PC\OneDrive\Documents\faces.jpeg"
```

```
if not os.path.exists(file_path):
    print("Error: File does not exist!")
    exit()

img = cv2.imread(file_path)

if img is None:
    print("Error: Could not read image.")
    exit()

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Threshold
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Find contours
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Draw contours on original image
cv2.drawContours(img, contours, -1, (0, 255, 0), 2)

# Convert BGR to RGB for Matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display the image
plt.imshow(img_rgb)
plt.axis('off')
```

plt.title("Contours")

plt.show()

OUTPUT: Image with contours drawn



OBSERVATIONS:

1 Loads an image and converts it to grayscale
2 Applies thresholding to get binary image
3 Finds contours in the image
4 Draws contours on original image
5 Shows the result and closes window after key press

CODE 13

import cv2

import matplotlib.pyplot as plt

import numpy as np

import os

Load image

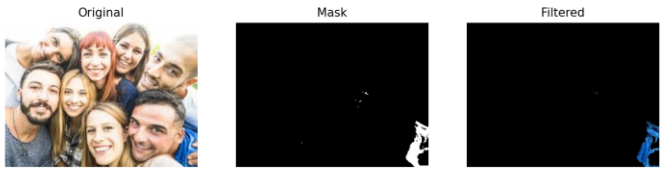
file_path = r"C:\Users\HP-PC\OneDrive\Documents\faces.jpeg"

if not os.path.exists(file_path):

```
    print("Error: File does not exist!")
    exit()

img = cv2.imread(file_path)

if img is None:
    print("Error: Could not read image.")
    exit()

# Convert to HSV color space
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Define blue color range
lower_blue = (100, 150, 0)
upper_blue = (140, 255, 255)

# Create mask for blue color
mask = cv2.inRange(hsv, lower_blue, upper_blue)

# Apply mask to original image
result = cv2.bitwise_and(img, img, mask=mask)

# Convert BGR to RGB for Matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
result_rgb = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)

# Display images
plt.figure(figsize=(12, 6))
```

```
plt.subplot(1, 3, 1)
plt.imshow(img_rgb)
plt.title("Original")
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(mask, cmap='gray')
plt.title("Mask")
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(result_rgb)
plt.title("Filtered")
plt.axis('off')

plt.show()
```

OUTPUT: Color detection and masking



OBSERVATIONS:

1 Loads an image and converts it to HSV color space
2 Defines blue color range
3 Creates mask for blue color

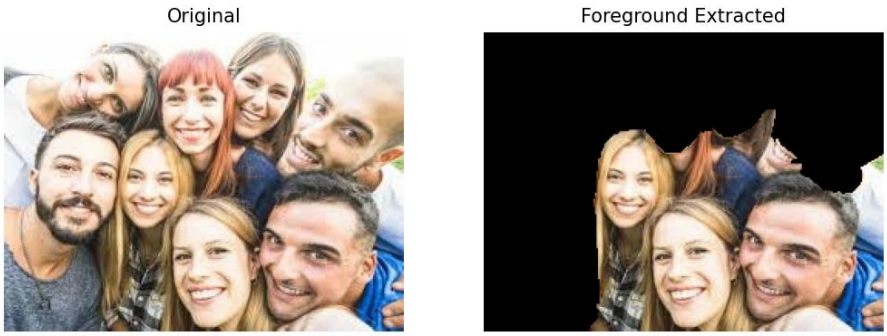
4 Applies mask to original image to filter blue objects
5 Shows original, mask, and filtered images
6 Waits for key press then closes all windows

CODE 14

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

import os


# Load image

file_path = r"C:\Users\HP-PC\OneDrive\Documents\faces.jpeg"

if not os.path.exists(file_path):

   print("Error: File does not exist!")

   exit()


img = cv2.imread(file_path)


if img is None:

   print("Error: Could not read image.")

   exit()


# Initialize mask and models for grabCut

mask = np.zeros(img.shape[:2], np.uint8)

bgdModel = np.zeros((1, 65), np.float64)

fgdModel = np.zeros((1, 65), np.float64)


# Define rectangle around foreground object

rect = (50, 50, img.shape[1]-50, img.shape[0]-50)  # Adjust rectangle to image size
```

```
# Apply grabCut algorithm
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)

# Create mask for foreground
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype("uint8")
result = img * mask2[:, :, np.newaxis]

# Convert BGR to RGB for Matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
result_rgb = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)

# Display original and foreground-extracted images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img_rgb)
plt.title("Original")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(result_rgb)
plt.title("Foreground Extracted")
plt.axis('off')

plt.show()
```

OUTPUT: Foreground extraction from image



OBSERVATIONS:

1 Loads an image and creates mask for GrabCut
2 Initializes background and foreground models
3 Defines rectangular ROI for GrabCut
4 Applies GrabCut algorithm to extract foreground
5 Shows original and foreground images
6 Waits for key press then closes all windows

CODE 15

```
import cv2
import numpy as np

# Open webcam
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Error: Could not open webcam.")
    exit()

while True:
    ret, frame = cap.read()
    if not ret:
        print("Failed to grab frame")
        break
```

```
    # Convert frame to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Define blue color range
    lower_blue = (100, 150, 0)
    upper_blue = (140, 255, 255)

    # Create mask and apply it
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
    result = cv2.bitwise_and(frame, frame, mask=mask)

    # Show frames
    cv2.imshow("Original Frame", frame)
    cv2.imshow("Mask", mask)
    cv2.imshow("Tracked Blue Objects", result)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
cap.release()
cv2.destroyAllWindows()
```

OUTPUT: Live blue object tracking

OBSERVATIONS:

1 Turns on the camera
2 Captures frames continuously
3 Converts frames to HSV color space
4 Detects and masks blue objects
5 Shows original frame, mask, and tracked output
6 Stops when key q is pressed
7 Releases camera and closes all windows

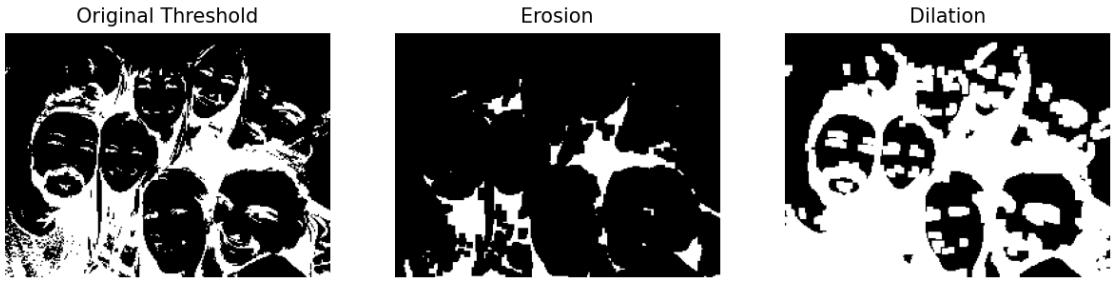
CODE 16

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

import os


# Load image

file_path = r"C:\Users\HP-PC\OneDrive\Documents\faces.jpeg"

if not os.path.exists(file_path):

   print("Error: File does not exist!")

   exit()


# Read in grayscale
```

```
img = cv2.imread(file_path, 0)

if img is None:
   print("Error: Could not read image.")
   exit()

# Apply binary inverse threshold
_, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)

# Create kernel for morphological operations
kernel = np.ones((5, 5), np.uint8)

# Apply erosion and dilation
erosion = cv2.erode(thresh, kernel, iterations=1)
dilation = cv2.dilate(thresh, kernel, iterations=1)

# Display results using Matplotlib
plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)
plt.imshow(thresh, cmap='gray')
plt.title("Original Threshold")
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(erosion, cmap='gray')
plt.title("Erosion")
plt.axis('off')
```

```
plt.subplot(1, 3, 3)

plt.imshow(dilation, cmap='gray')

plt.title("Dilation")

plt.axis('off')


plt.show()
```

OUTPUT: Morphological operations on text image



OBSERVATIONS:

1 Loads a grayscale text image
2 Applies binary inverse thresholding
3 Performs erosion to shrink white regions
4 Performs dilation to enlarge white regions
5 Shows original, erosion, and dilation results
6 Waits for key press then closes all windows