COMPUTER VISION

CODE 1: (code_1.py)

Concept used: Real-time video capture and display using OpenCV

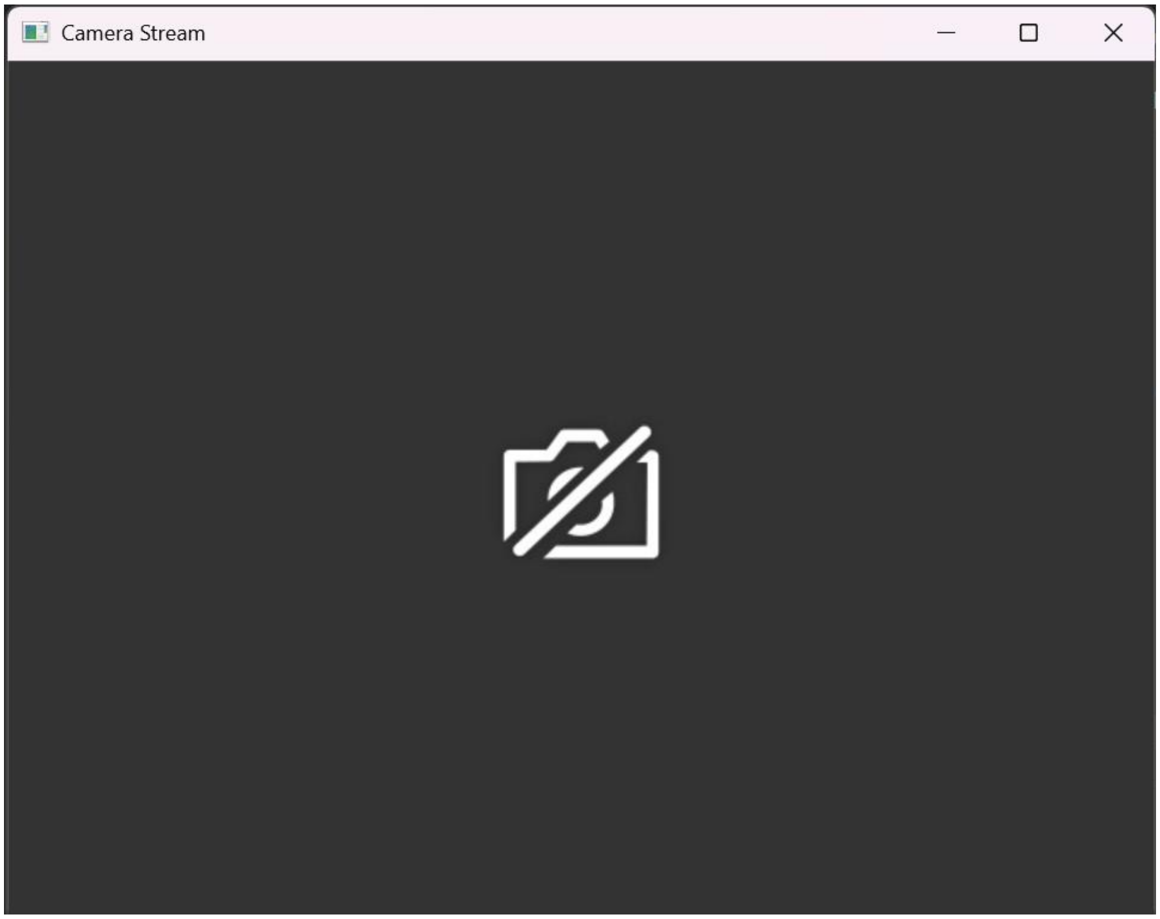
Observation: A live camera feed window opens and displays continuous video frames until 'q' is pressed.

Code:

```
import cv2

cap = cv2.VideoCapture(0)
while True:
   ret, frame = cap.read()
   cv2.imshow("Camera Stream", frame)
   if cv2.waitKey(1) & 0xFF == ord("q"):
      break
cap.release()
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 2: (code_2.py)

Concept used: Video capture with automatic frame saving

Observation: Shows live camera feed while automatically saving each frame as numbered JPEG files in a 'frames' folder.

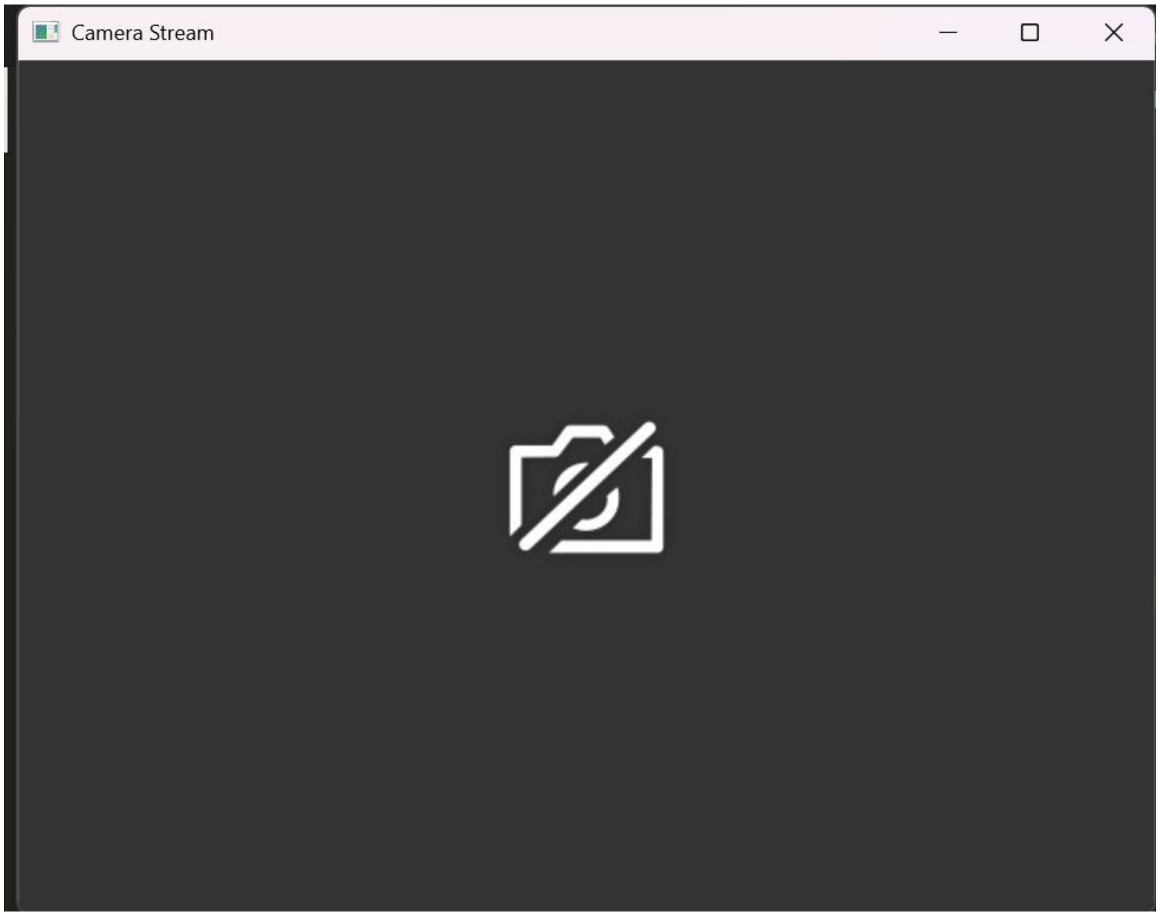
Code:

```
import cv2
import os

cap = cv2.VideoCapture(0)
os.makedirs("frames", exist_ok=True)
frame_count = 0
while True:
  ret, frame = cap.read()
```

```
cv2.imshow("Camera Stream", frame)
filename = f"frames/frame_{frame_count:06d}.jpg"
cv2.imwrite(filename, frame)
frame_count += 1
```

Output Screenshot:



CODE 3: (code_3.py)

Concept used: Basic image loading and display

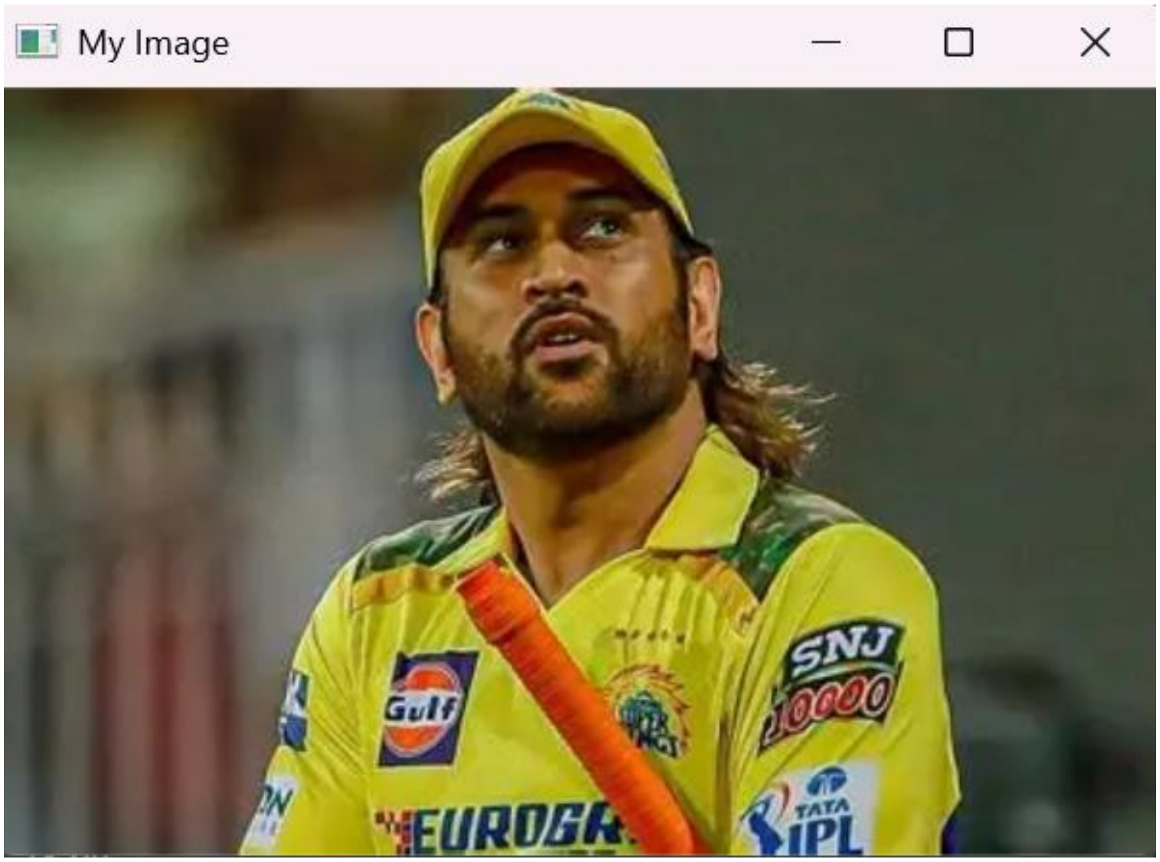
Observation: Displays an image in a window until a key is pressed. Shows error if image file is not found.

Code:

```
import cv2
```

```
img = cv2.imread("image.png")
if img is None:
   print("Error: Could not read image.")
else:
   cv2.imshow("My Image", img)
   cv2.waitKey(0)
   cv2.destroyAllWindows()
```

Output Screenshot:



CODE 4: (code_4.py)

Concept used: Image flipping transformations

Observation: Displays original image and three flipped versions: vertical, horizontal, and both.

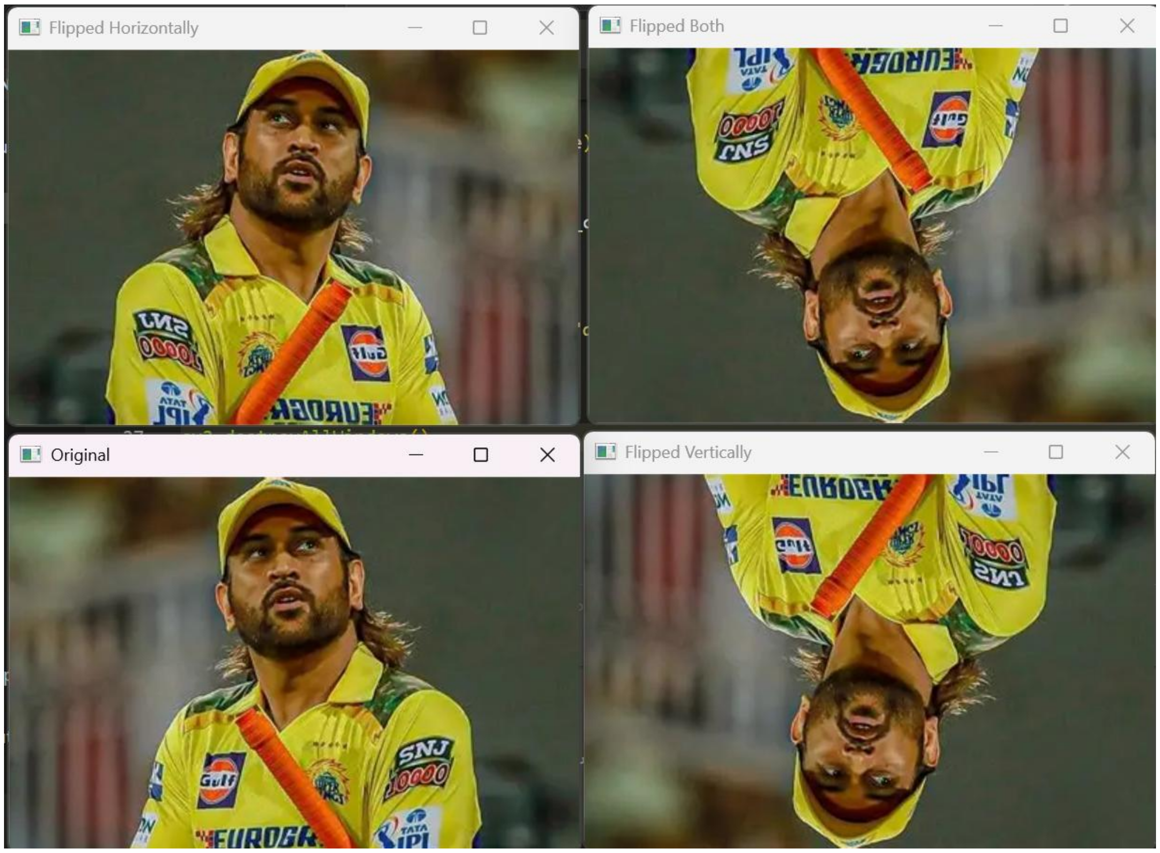
Code:

```
import cv2
```

```
img = cv2.imread("image.png")
flip_vertical = cv2.flip(img, 0)
flip_horizontal = cv2.flip(img, 1)
flip_both = cv2.flip(img, -1)

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 Screenshot:



CODE 5: (code_5.py)

Concept used: Image resizing

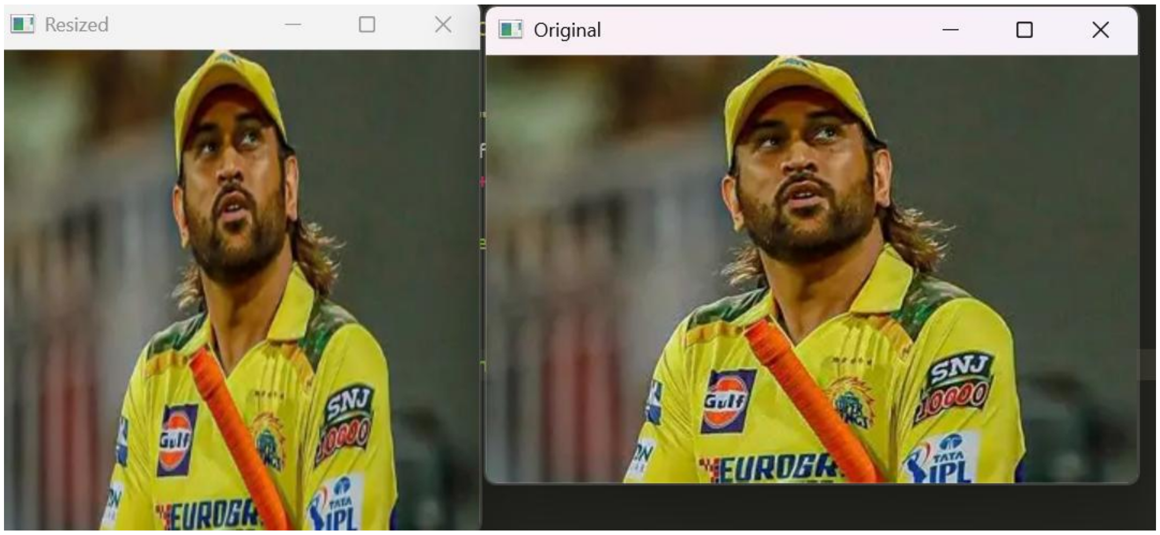
Observation: Resizes image to 300x300 and displays both original and resized versions. Also saves the resized image.

Code:

```
import cv2

img = cv2.imread("image.png")
resized = cv2.resize(img, (300, 300))
cv2.imshow("Original", img)
cv2.imshow("Resized", resized)
cv2.imwrite("resized_output.jpg", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 6: (code_6.py)

Concept used: Converting color images to grayscale

Observation: Converts color image to grayscale and displays both. Grayscale image is also saved.

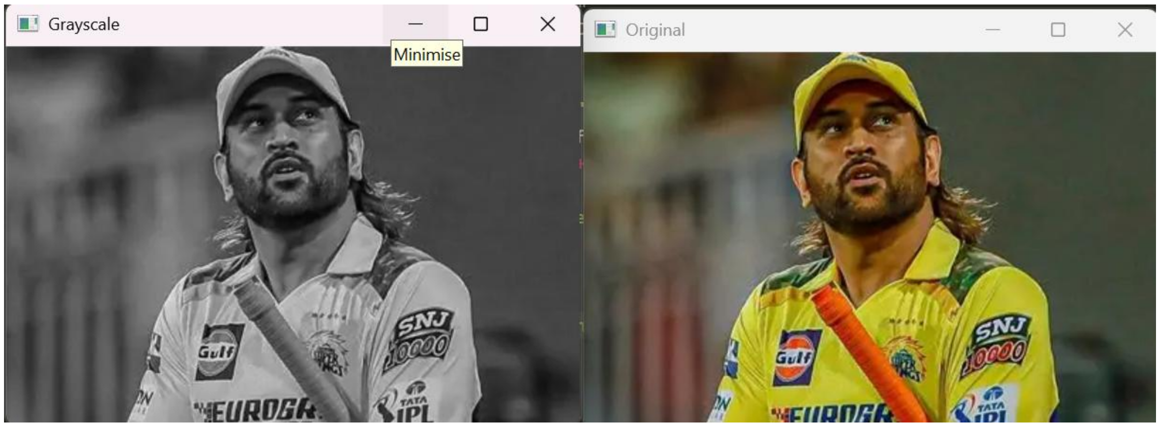
Code:

```
import cv2

img = cv2.imread("image.png")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow("Original", img)
cv2.imshow("Grayscale", gray)
cv2.imwrite("grayscale_output.jpg", gray)
```

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

Output Screenshot:



CODE 7: (code_7.py)

Concept used: Image blurring using Gaussian filter

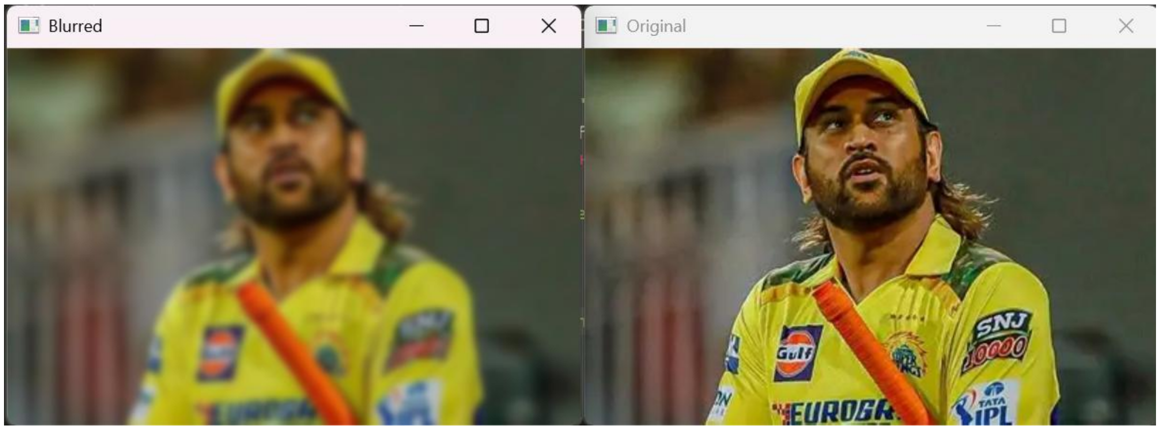
Observation: Applies Gaussian blur, making the image appear softer.

Code:

```
import cv2

img = cv2.imread("image.png")
blur = cv2.GaussianBlur(img, (15, 15), 0)
cv2.imshow("Original", img)
cv2.imshow("Blurred", blur)
cv2.imwrite("blurred_output.jpg", blur)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 8: (code_8.py)

Concept used: Drawing shapes and text on images

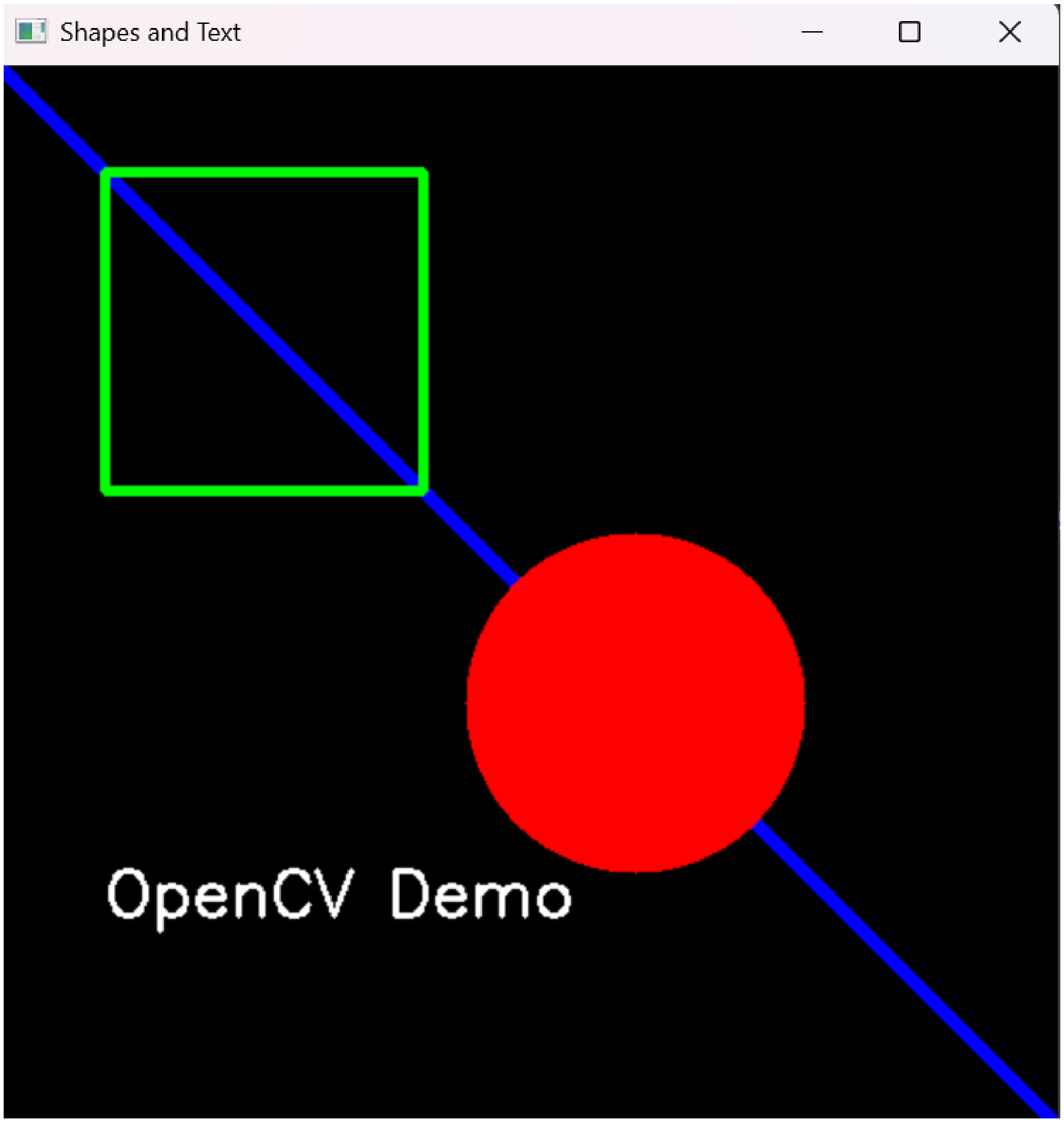
Observation: Creates a black canvas and draws line, rectangle, circle, and text.

Code:

```
import cv2
import numpy as np

img = np.zeros((500, 500, 3), dtype="uint8")
cv2.line(img, (0, 0), (500, 500), (255, 0, 0), 5)
cv2.rectangle(img, (50, 50), (200, 200), (0, 255, 0), 3)
cv2.circle(img, (300, 300), 80, (0, 0, 255), -1)
cv2.putText(img, "OpenCV Demo", (50, 400),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
cv2.imshow("Shapes and Text", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 9: (code_9.py)

Concept used: Binary thresholding

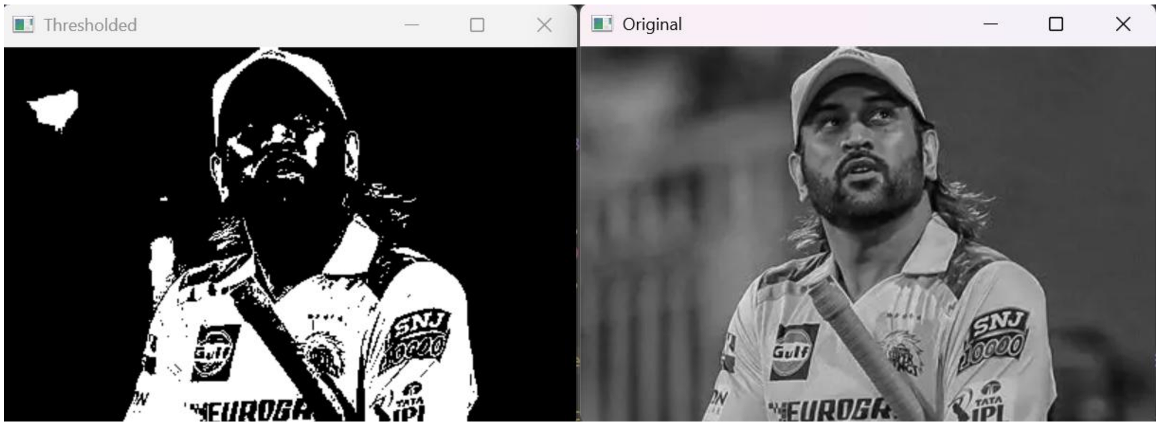
Observation: Converts grayscale image to black and white using threshold 127.

Code:

```
import cv2

img = cv2.imread("image.png", 0)
_, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
cv2.imshow("Original", img)
cv2.imshow("Thresholded", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 10: (code_10.py)

Concept used: Edge detection using Canny algorithm

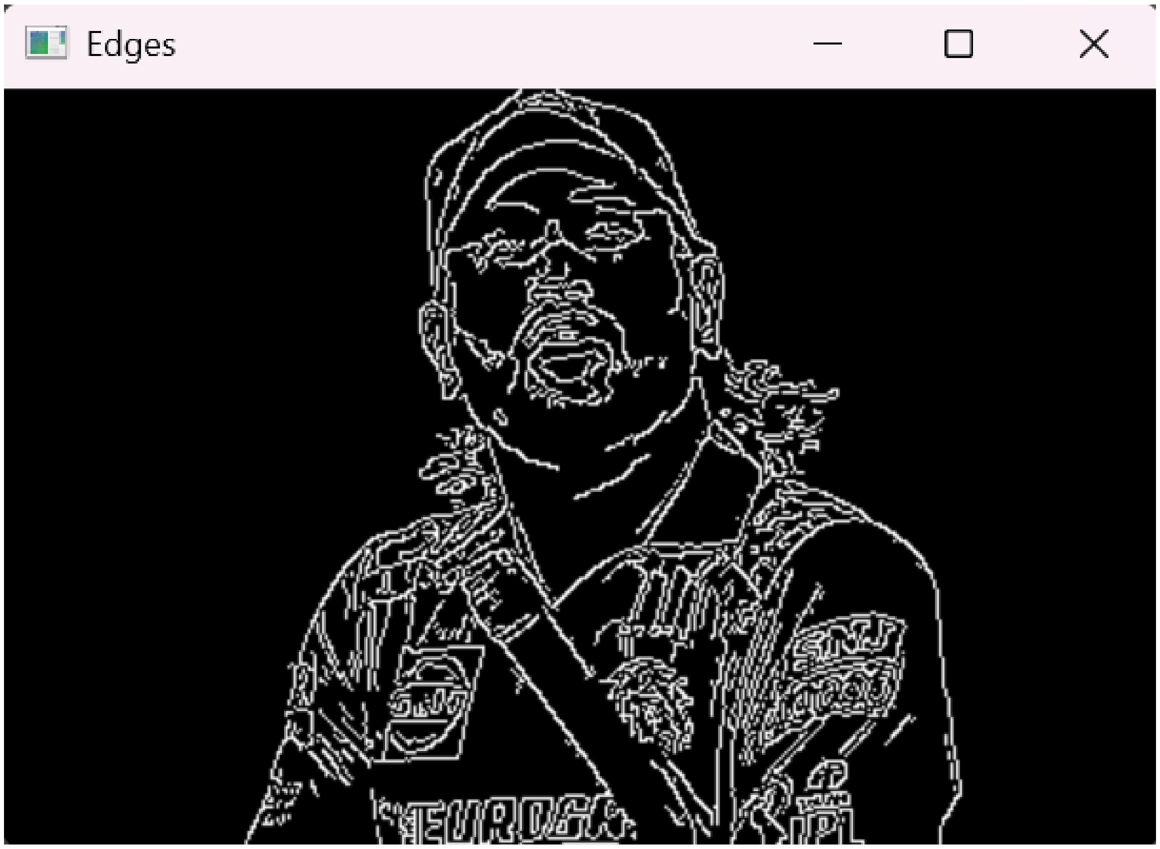
Observation: Detects and highlights edges, giving a sketch-like appearance.

Code:

```
import cv2

img = cv2.imread("image.png", 0)
edges = cv2.Canny(img, 100, 200)
cv2.imshow("Edges", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 11: (code_11.py)

Concept used: Automatic face detection

Observation: Detects faces using Haar cascade and draws blue rectangles.

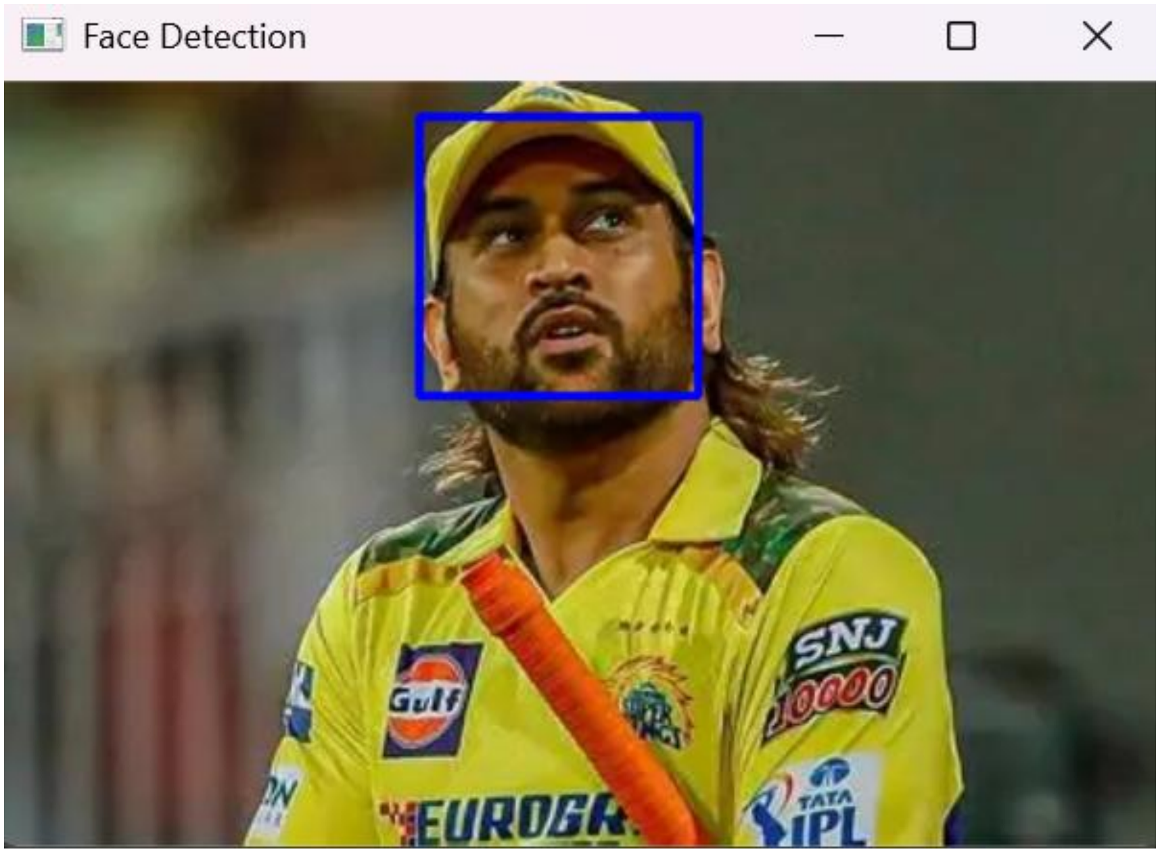
Code:

```
import cv2

img = cv2.imread("image.png")
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
for (x, y, w, h) in faces:
  cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
cv2.imshow("Face Detection", img)
```

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

Output Screenshot:



CODE 12: (code_12.py)

Concept used: Finding and drawing contours

Observation: Finds object boundaries and draws green contours.

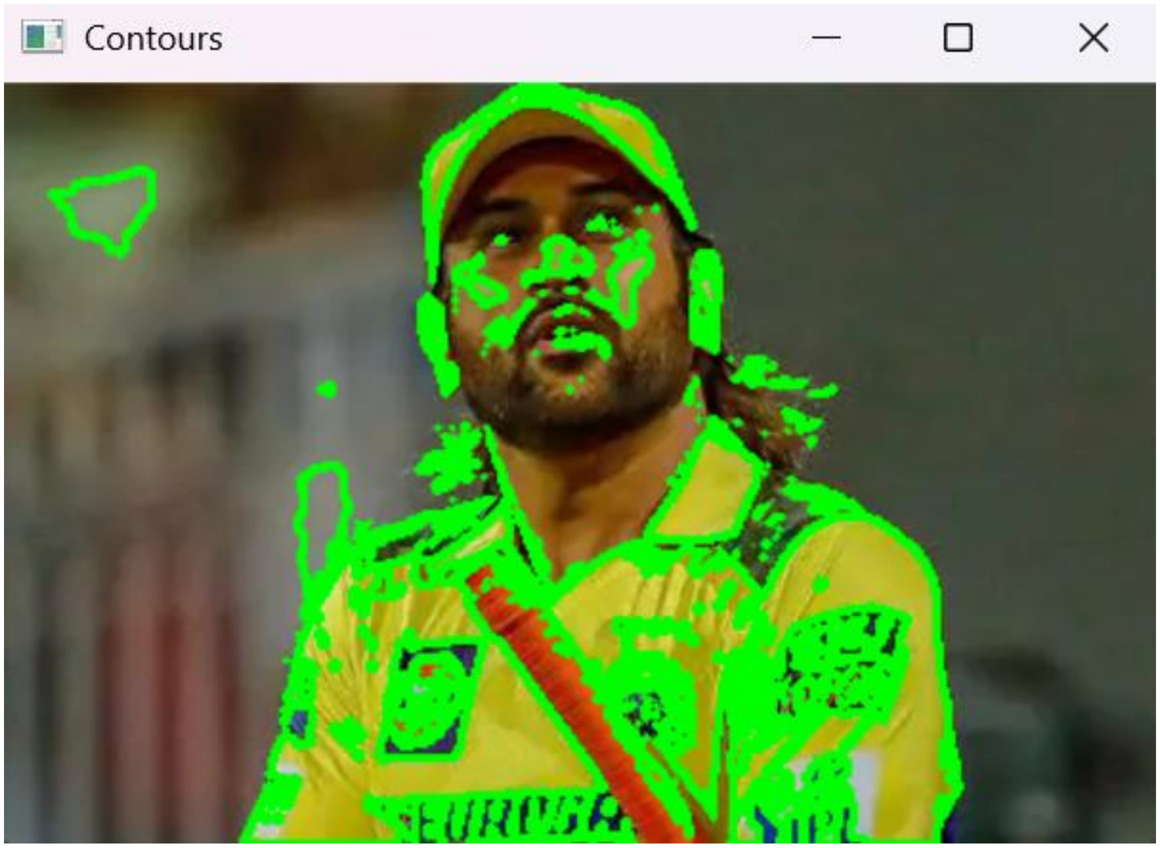
Code:

```
import cv2

img = cv2.imread("image.png")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

```
cv2.drawContours(img, contours, -1, (0, 255, 0), 2)
cv2.imshow("Contours", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 13: (code_13.py)

Concept used: Color-based filtering using HSV color space

Observation: Filters image to show only blue-colored objects.

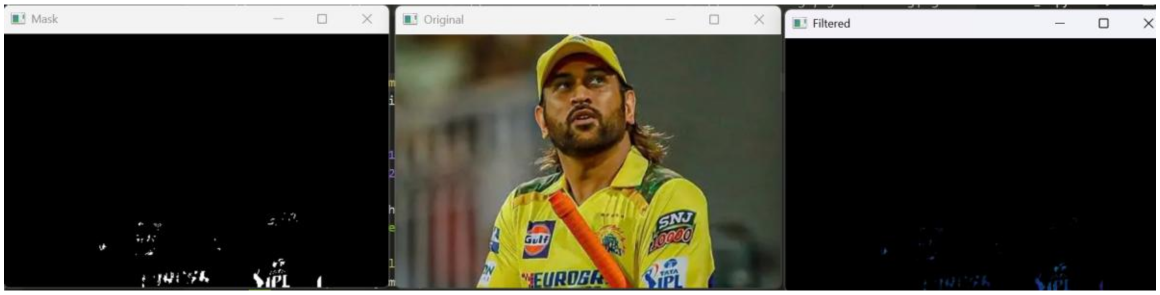
Code:

```
import cv2

img = cv2.imread("image.png")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
lower_blue = (100, 150, 0)
upper_blue = (140, 255, 255)
mask = cv2.inRange(hsv, lower_blue, upper_blue)
```

```
result = cv2.bitwise_and(img, img, mask=mask)
cv2.imshow("Original", img)
cv2.imshow("Mask", mask)
cv2.imshow("Filtered", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 14: (code_14.py)

Concept used: Advanced background removal using GrabCut

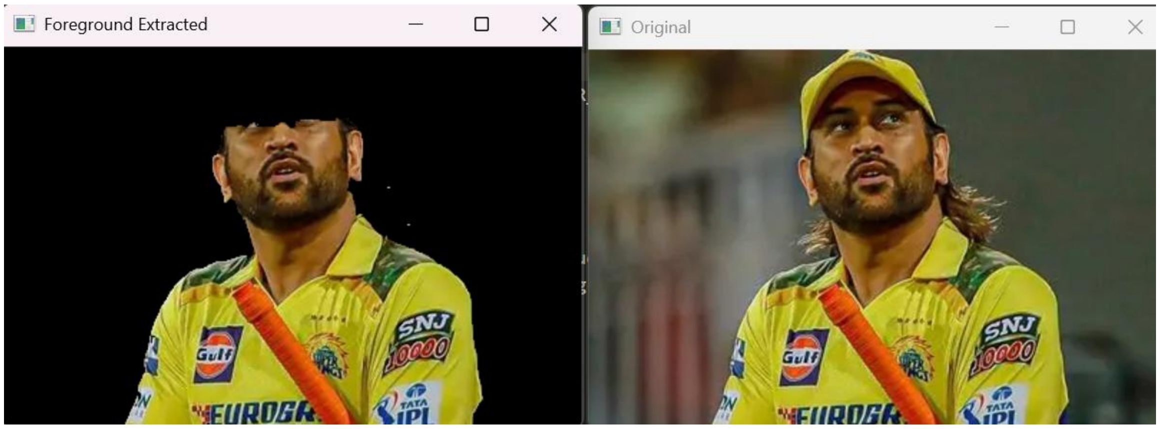
Observation: Separates foreground from background using rectangular region.

Code:

```
import cv2
import numpy as np

img = cv2.imread("image.png")
mask = np.zeros(img.shape[:2], np.uint8)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
rect = (50, 50, 400, 500)
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5,
cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype("uint8")
result = img   mask2[:, :, np.newaxis]
cv2.imshow("Original", img)
cv2.imshow("Foreground Extracted", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 15: (code_15.py)

Concept used: Real-time color tracking in live video

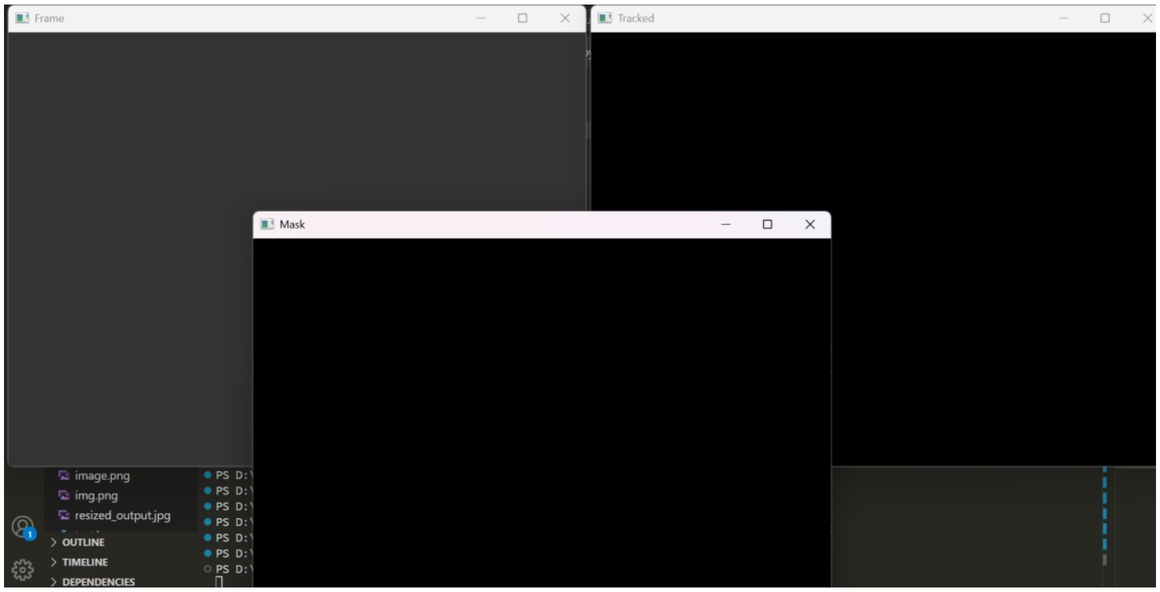
Observation: Tracks blue objects in real-time using webcam.

Code:

```
import cv2

cap = cv2.VideoCapture(0)
while True:
   ret, frame = cap.read()
   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
   lower_blue = (100, 150, 0)
   upper_blue = (140, 255, 255)
   mask = cv2.inRange(hsv, lower_blue, upper_blue)
   result = cv2.bitwise_and(frame, frame, mask=mask)
   cv2.imshow("Frame", frame)
   cv2.imshow("Mask", mask)
   cv2.imshow("Tracked", result)
   if cv2.waitKey(1) & 0xFF == ord("q"):
      break
cap.release()
cv2.destroyAllWindows()
```

Output Screenshot:



CODE 16: (code_16.py)

Concept used: Morphological operations for image processing

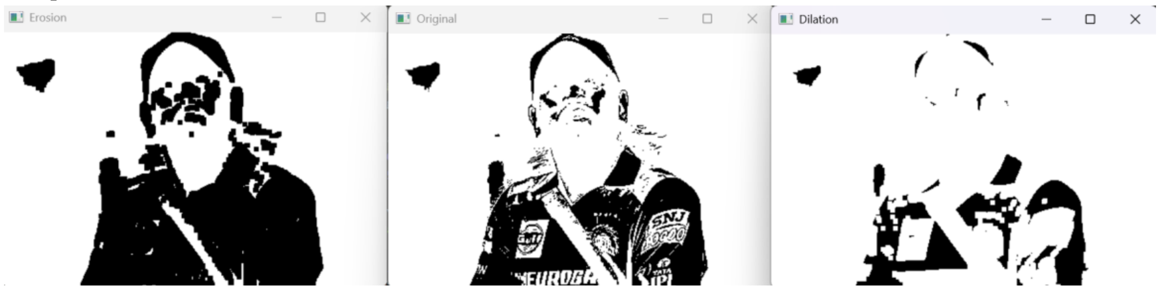
Observation: Demonstrates erosion (shrinks white) and dilation (expands white) on binary images.

Code:

```
import cv2
import numpy as np

img = cv2.imread("image.png", 0)
_, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
kernel = np.ones((5, 5), np.uint8)
erosion = cv2.erode(thresh, kernel, iterations=1)
dilation = cv2.dilate(thresh, kernel, iterations=1)
cv2.imshow("Original", thresh)
cv2.imshow("Erosion", erosion)
cv2.imshow("Dilation", dilation)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output Screenshot:


Erosion
Original
Dilation