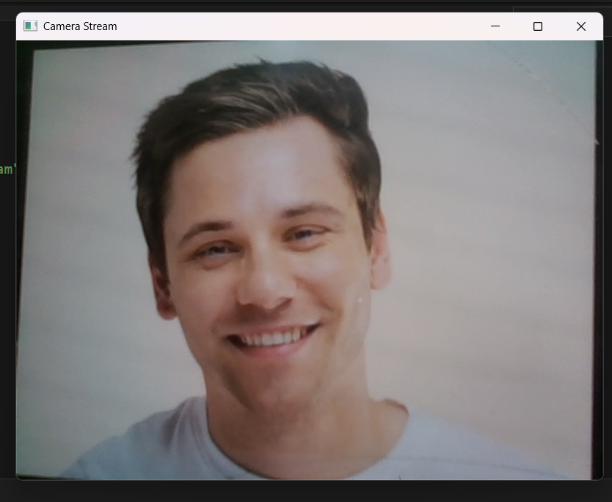
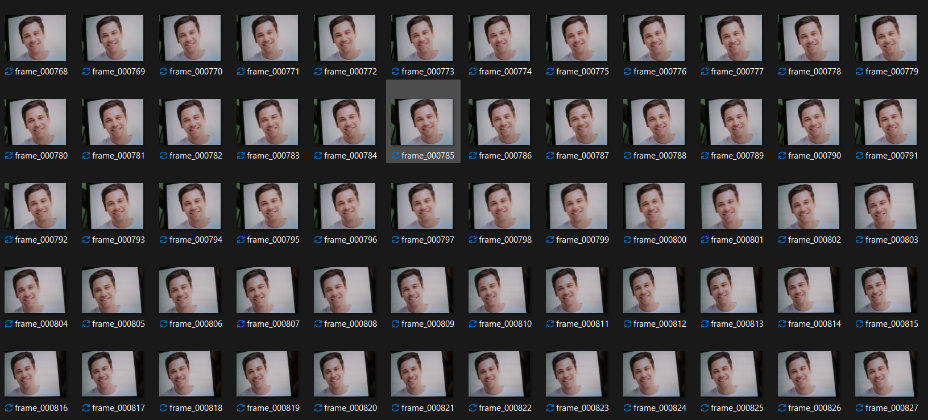
Computer Vision

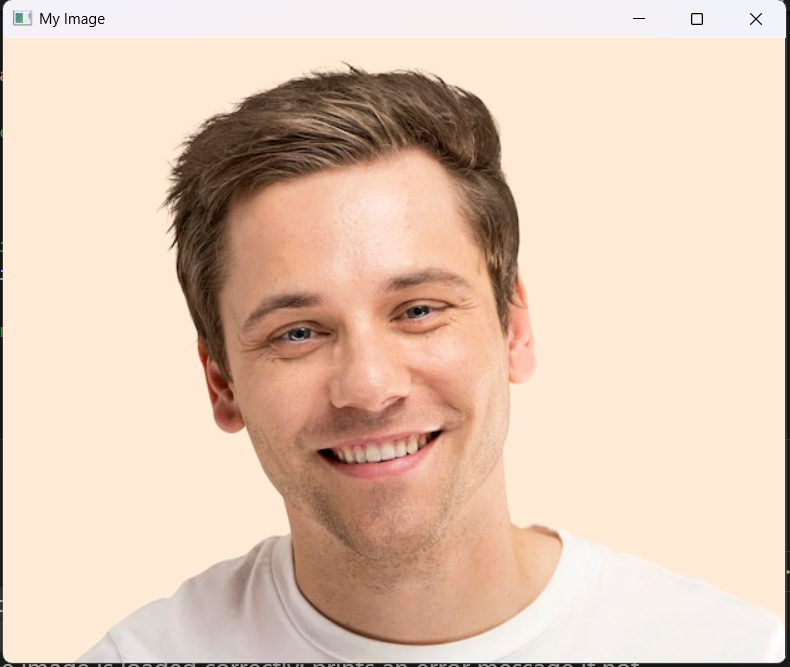
**CODE 1.py**



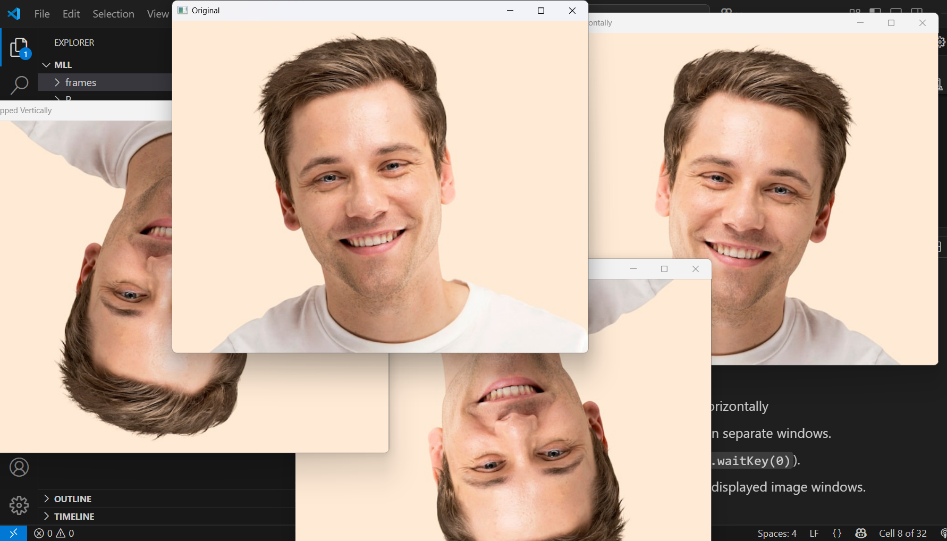
* Opens webcam (cv2.VideoCapture(0)) or IP stream.
* Reads frames in a loop.
* Shows frames in a window.
* Press **q** to quit.
* Releases camera & closes windows.

**CODE 2.py**

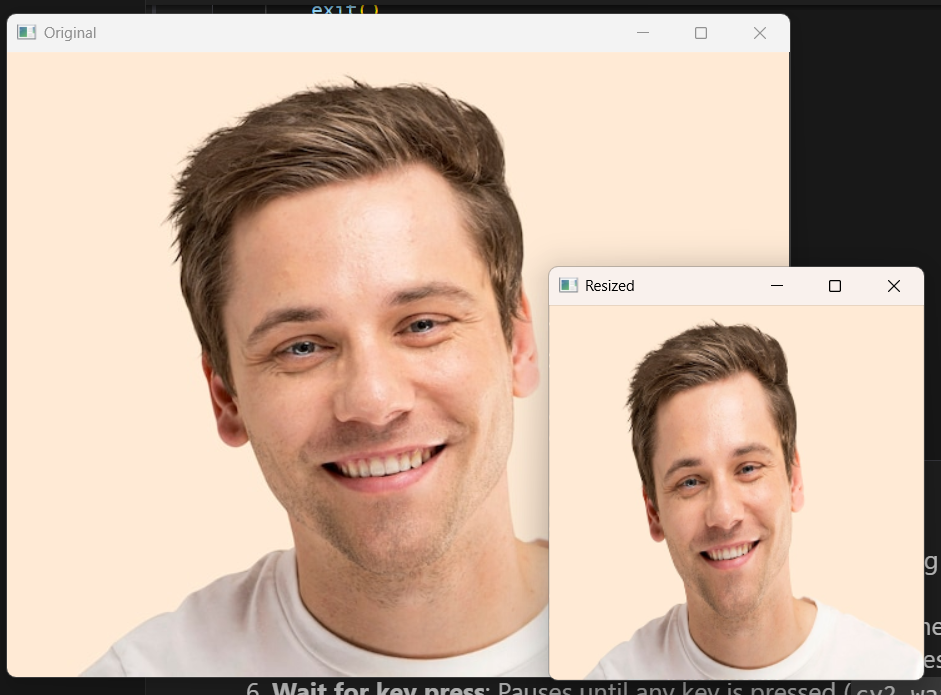
* Opens webcam and starts video capture.
* Creates a folder **frames/** to store images.
* Captures and displays live stream.
* Saves each frame as frame\_000000.jpg, frame\_000001.jpg, etc.
* Stops when **q** is pressed, then releases resources.

**CODE 3.py**

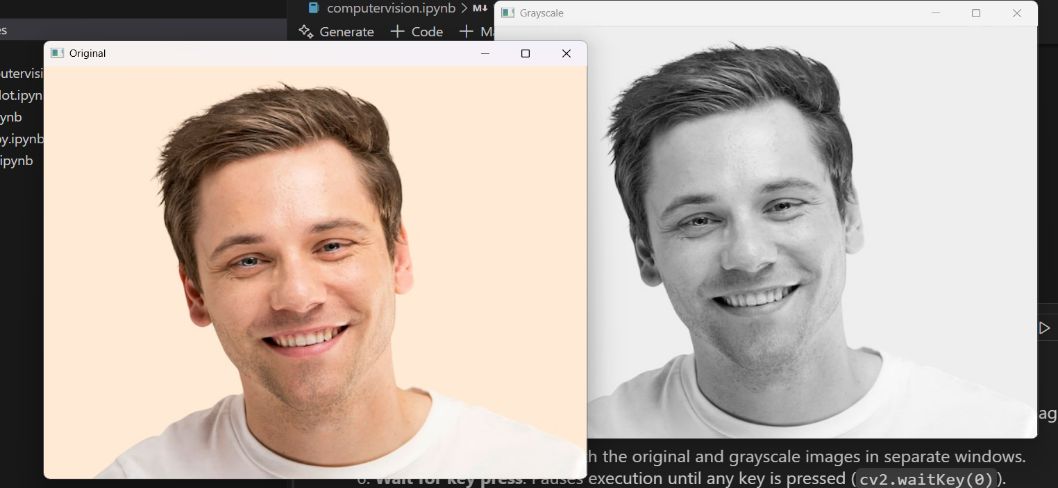
* Loads an image from a given file path.
* Checks if the image is read successfully.
* Displays the image in a window titled **"My Image"**.
* Waits for any key press before closing.
* Closes all OpenCV windows after exit.

**CODE 4.py**

* Reads an image from the given path.
* Checks if the image is loaded correctly.
* Uses cv2.flip() for vertical, horizontal, and both flips.
* Displays original and flipped versions in separate windows.
* Waits for a key press, then closes all windows.

**CODE 5.py**

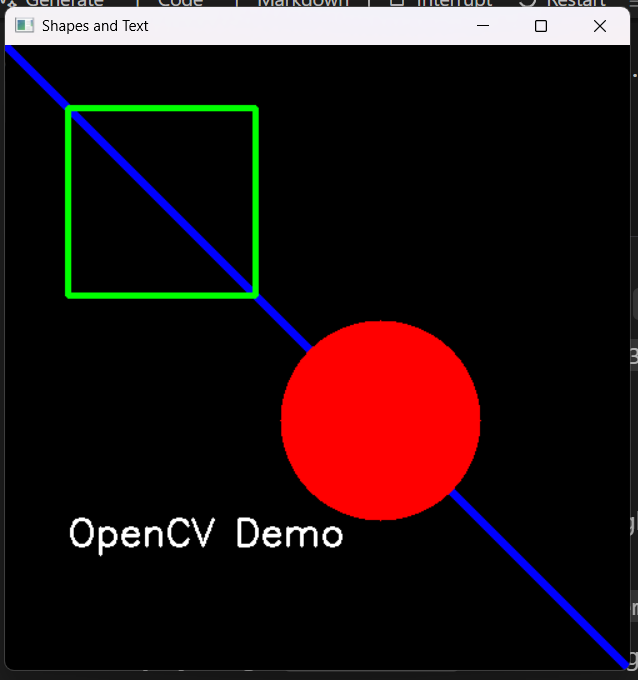
* Loads an image from the given path.
* Checks if the image is read successfully.
* Resizes the image to **300×300 pixels**.
* Displays both original and resized images.
* Optionally saves the resized image as **resized\_output.jpg**.

**CODE 6.py**

* Loads an image from the given path.
* Verifies successful image loading.
* Converts the image to **grayscale** using cv2.cvtColor().
* Displays both original and grayscale images in windows.
* Optionally saves the grayscale image (typo: should be gray, not gqray).

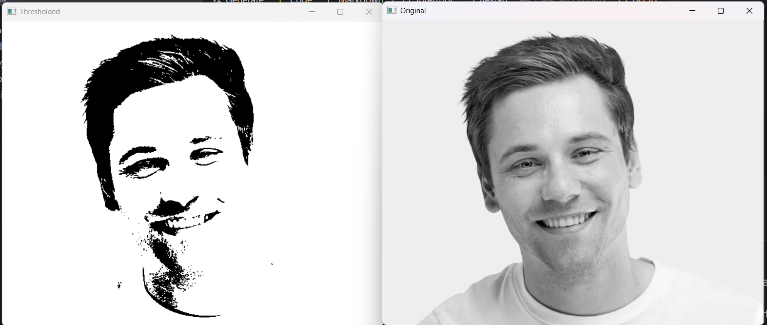
**CODE 7.py**

* Loads an image from the given path.
* Checks if the image is loaded successfully.
* Applies **Gaussian Blur** with a 15×15 kernel.
* Displays original and blurred images in separate windows.
* Optionally saves the blurred image as **blurred\_output.jpg**.

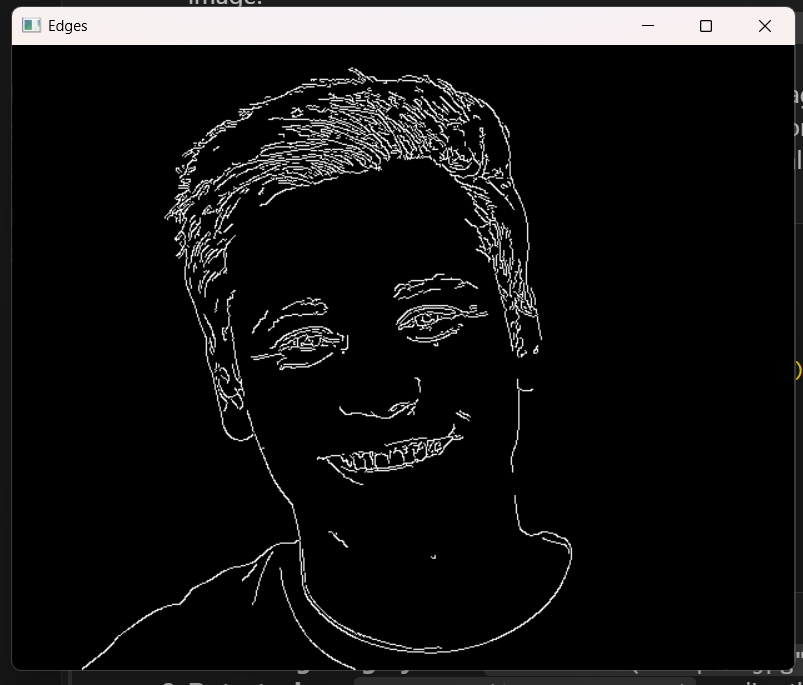
**CODE 8.py**

* Creates a blank **500×500 black image** using NumPy.
* Draws a **blue line** from top-left to bottom-right.
* Draws a **green rectangle** and a **filled red circle**.
* Adds white text "OpenCV Demqo" on the image.
* Displays the result in a window until a key is pressed.

( Small typo: "Demqo" → should likely be "Demo".)

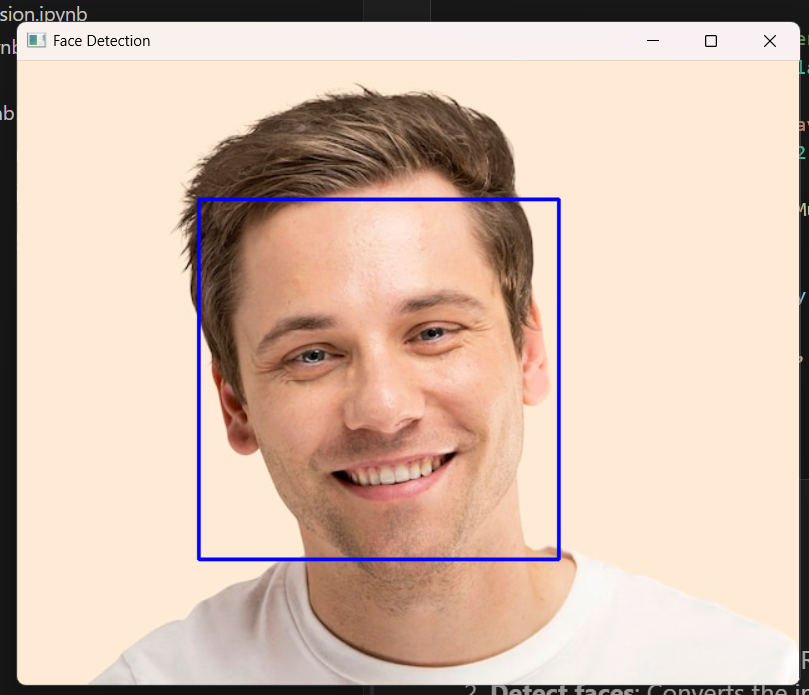
**CODE 9.py**

* Loads an image in **grayscale mode**.
* Applies **binary thresholding** with a threshold value of 127.
* Pixels above 127 become **white (255)**, below become **black (0)**.
* Displays both original grayscale and thresholder images.
* Waits for a key press, then closes all windows.

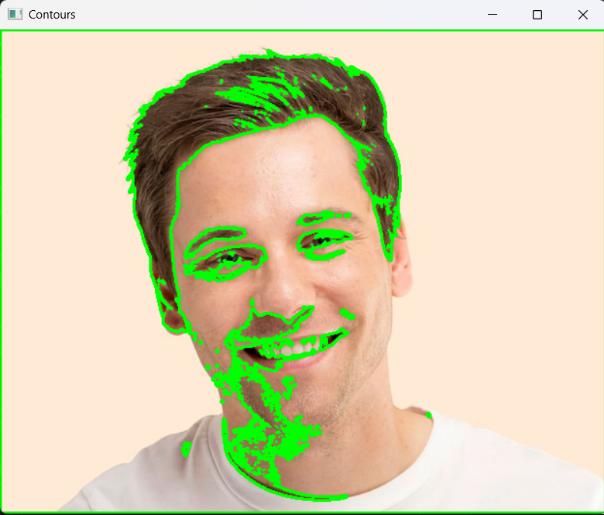
**CODE 10.py**

* Loads an image in **grayscale mode**.
* Applies **Canny edge detection** with thresholds 100 and 200.
* Detects and highlights edges in the image.
* Displays only the edge-detected result.
* Waits for a key press, then closes the window.

**CODE 11.py**



* Loads a **Haar Cascade classifier** for face detection.
* Reads the input image and converts it to **grayscale**.
* Uses detect Multiscale() to detect faces in the image.
* Draws **blue rectangles** around detected faces.
* Displays the result and waits for a key press to close.

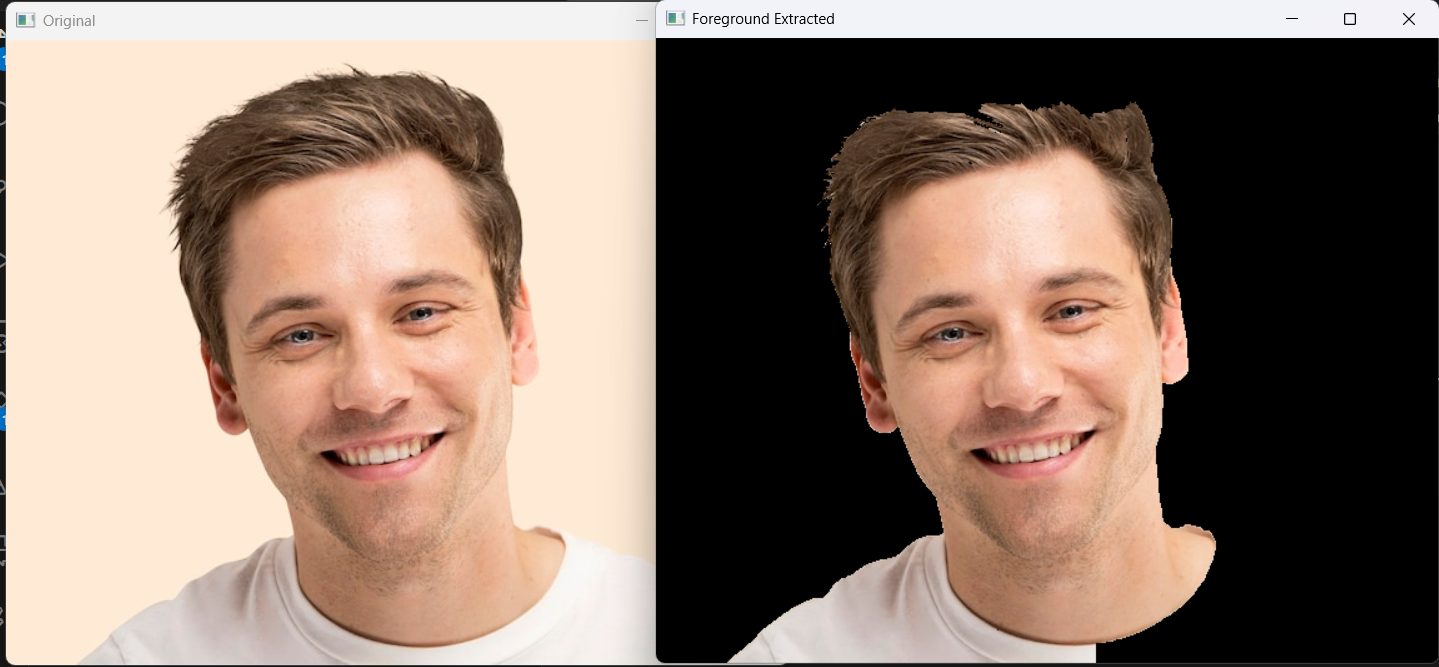
**CODE 12.py**

* Loads an image and converts it to **grayscale**.
* Applies **binary thresholding** to prepare for contour detection.
* Finds contours using cv2.findContours().
* Draws all detected contours in **green** on the original image.
* Displays the result and waits for a key press to close.

**CODE 13.py**

* Loads an image and converts it to **HSV color space**.
* Defines a **white color range** (lower and upper bounds).
* Creates a **mask** that isolates blue regions in the image.
* Uses bitwise\_and() to extract only the blue parts.
* Displays the original image, the mask, and the filtered result.

**CODE 14.py**

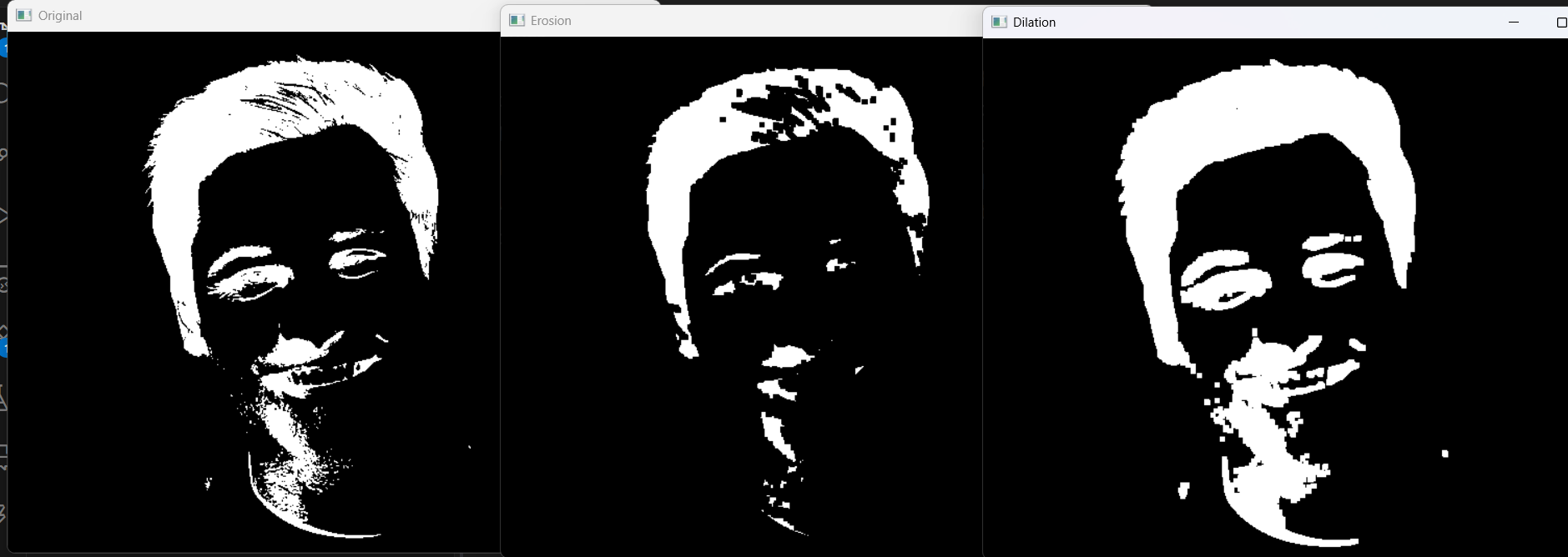


* Loads an image and creates an empty mask for segmentation.
* Initializes background and foreground models required by **GrabCut**.
* Defines a rectangular ROI (50, 50, 400, 500) for segmentation.
* Uses cv2.grabCut() to separate foreground from background.
* Applies the mask to extract and display the **foreground only**.

**CODE 15.py**

* Opens the **webcam** and captures frames in real time.
* Converts each frame from **BGR to HSV** color space.
* Defines a **blue color range** and creates a mask with cv2.inRange().
* Uses bitwise\_and() to highlight only the blue areas in the video.
* Displays the original frame, the mask, and the **tracked output**, stopping when **q** is pressed.

**CODE 16.py**



* **Grayscale Reading:** The image is loaded in grayscale (0), which is required for thresholding.
* **Binary Inversion:** cv2.THRESH\_BINARY\_INV converts pixels above 127 to 0 and below to 255, creating a negative effect.
* **Kernel Creation:** A 5×5 matrix of ones is used as the structuring element for morphological operations.
* **Erosion Effect:** Erosion shrinks white regions, removing small noise but thinning objects.
* **Dilation Effect:** Dilation expands white regions, filling gaps and enlarging features after erosion.