



**ORMEDIAN RESEARCH INSTITUTE**

TOPIC: : **OpenCV Tutorials**  
**(part eight)**

By:

**ADEKOLA OLUWASEUN O.**

**February 26, 2023**

# TABLE OF CONTENT

- ❑ What Histograms are in OpenCV
- ❑ Computing Histograms for Grayscale Images in OpenCV
- ❑ Computing Histograms for BGR Images
- ❑ Thresholding in OpenCV
- ❑ Implementation of a Simple Thresholding in OpenCV
- ❑ Inversion of a Simple Thresholded Image in OpenCV
- ❑ Implementation of an Adaptive Thresholding by Mean Method in OpenCV
- ❑ Inversion of an Adaptive Thresholded Image in OpenCV
- ❑ Implementation of an Adaptive Thresholding by Gaussian Method
- ❑ Fine Tuning Adaptive Thresholded Images
- ❑ Edge Detection Techniques in OpenCV
- ❑ Implementation of Laplacian and Sobel Edge Detector

# Computing Histograms in OpenCV

In OpenCV, histograms are very essential when there is a need for visualization of the distribution of the pixel intensities in an image. Histograms can be computed for RGB images and grayscale images. In defining a syntax for computing histograms, some arguments are very essential and must be well stated for the histogram to be successfully computed for the image of interest. Upon giving the required arguments to the `calcHist()` function, then we can use the matplotlib to plot the result.

## SYNTAX

```
cv2.calcHist([list of images], [channels], mask, highest intensity, [intensity range])
```

# Computing Histograms in OpenCV

Cont'd

## What the Algorithm is all about

**images:** Represents the list of images to use to compute the histogram.

**channels:** Represents the list of indexes of channels we want to compute histogram for.

**mask:** This specifies the mask to use.

**histSize:** This is used to specify the number of bins to use during histogram computation.

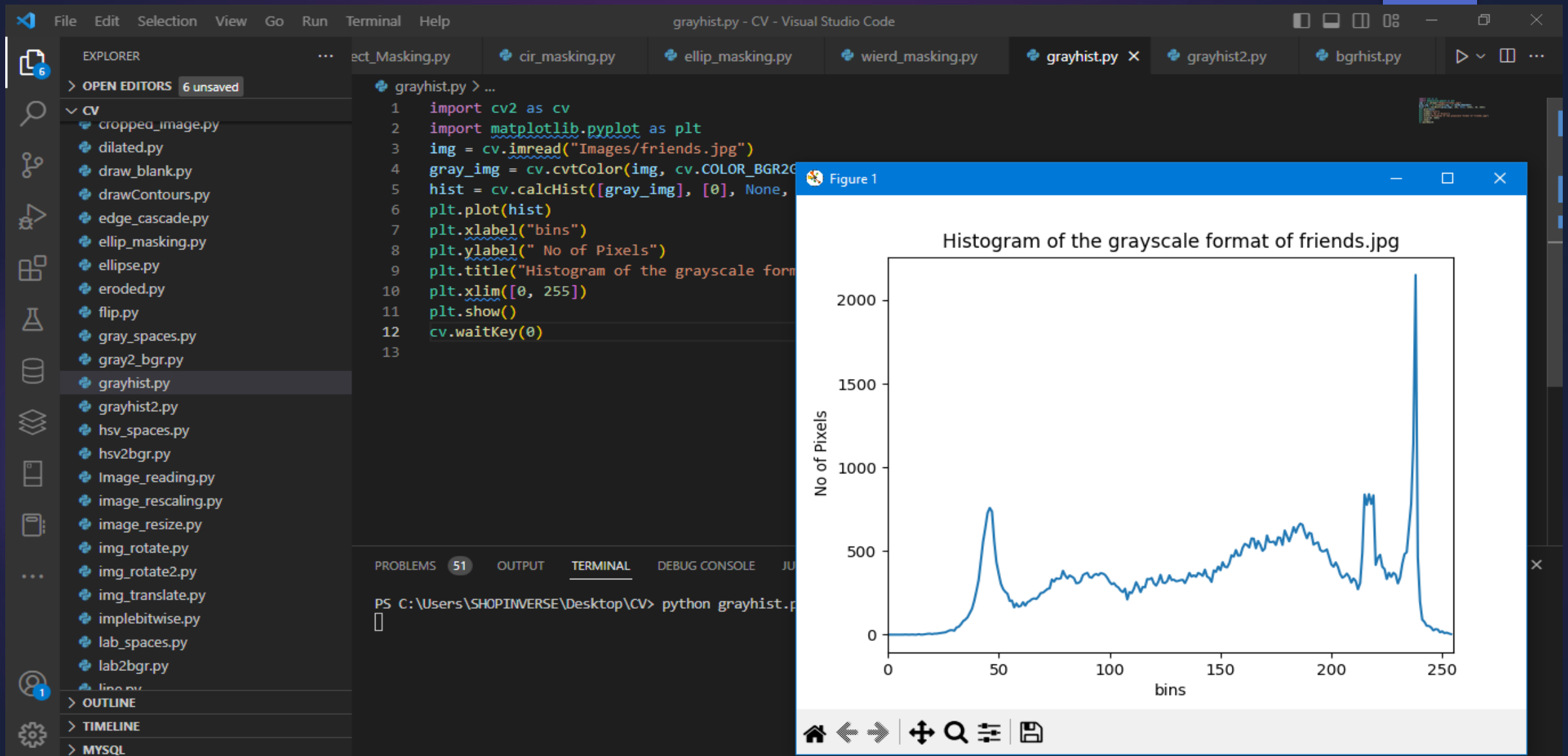
# Computing Histograms in OpenCV

## Cont'd

**Example 1:** Compute the histogram for the grayscale image version of “friends.jpg”.

```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
hist = cv.calcHist([gray_img], [0], None, [255], [0, 255])
plt.plot(hist)
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the grayscale format of friends.jpg")
plt.xlim([0, 255])
plt.show()
cv.waitKey(0)
```

# OUTPUT



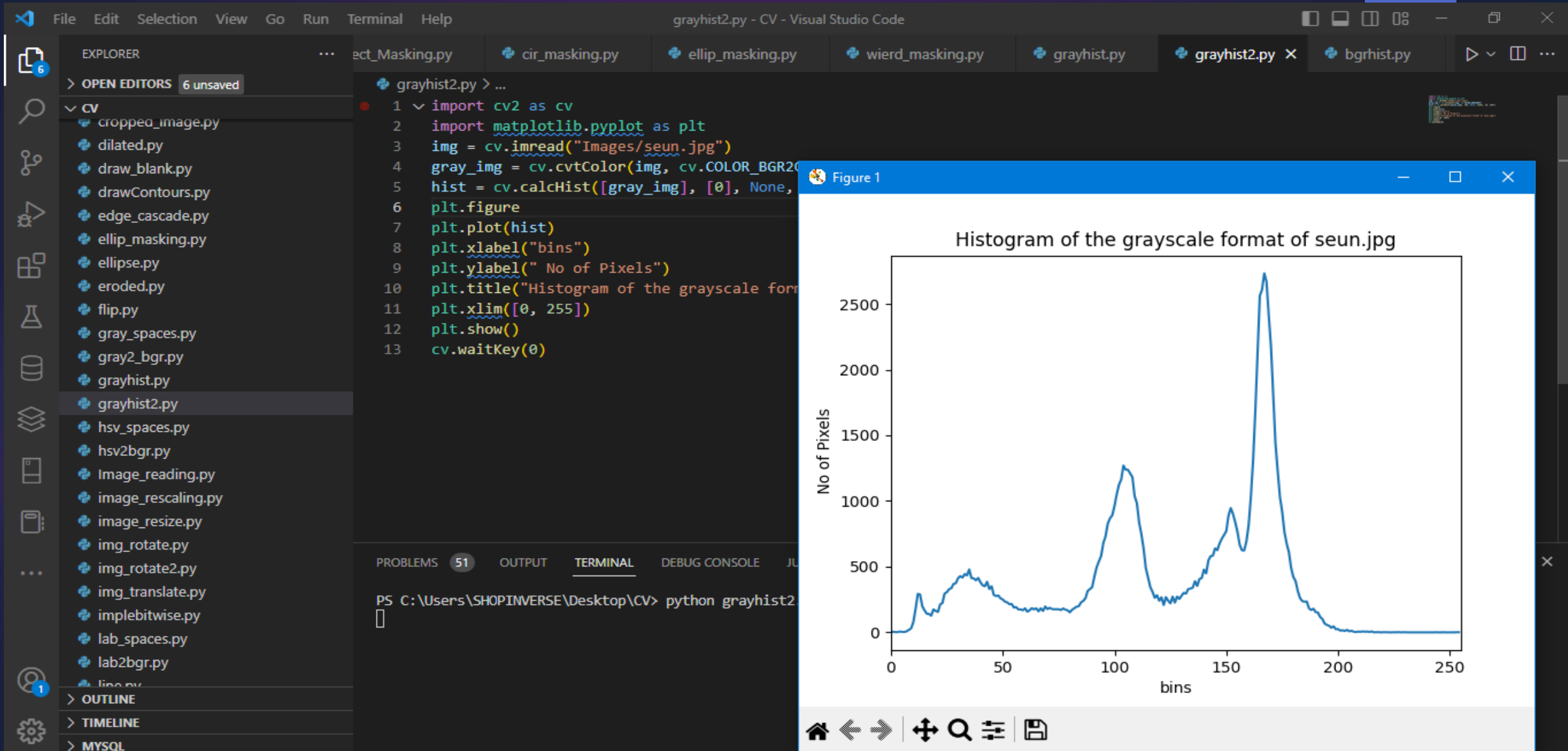
# Computing Histograms in OpenCV

## Cont'd

**Example 2:** Compute the histogram for the grayscale image version of “seun.jpg”.

```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("Images/seun.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
hist = cv.calcHist([gray_img], [0], None, [256], [0, 256])
plt.figure
plt.plot(hist)
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the grayscale format of seun.jpg")
plt.xlim([0, 256])
plt.show()
cv.waitKey(0)
```

# OUTPUT





# Computing Histograms in OpenCV

## Cont'd

Histogram Computation is not only limited to grayscale images, it can also be computed for BGR images.

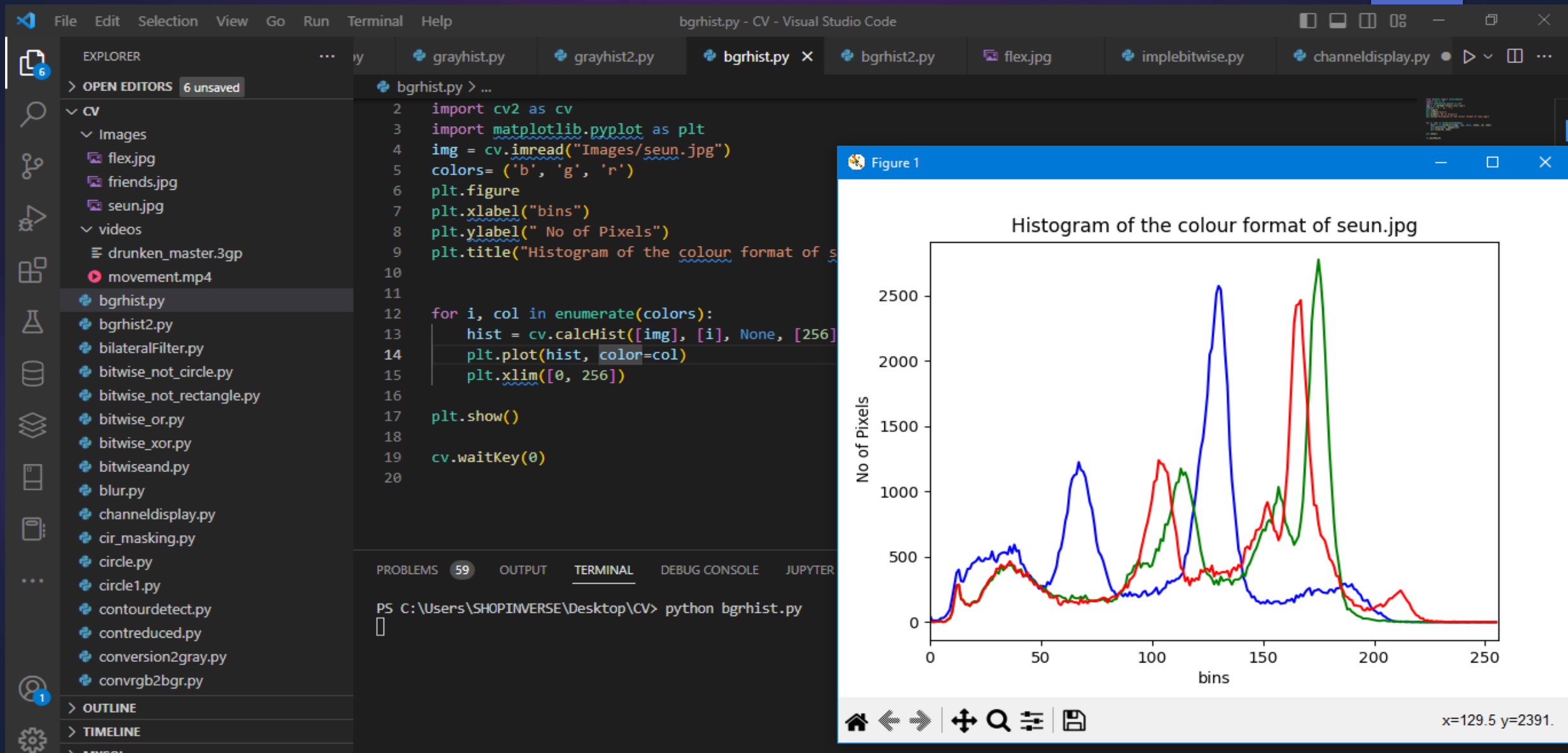
**Example 2:** Compute the histogram for the BGR image version of “seun.jpg”.

```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("Images/seun.jpg")
colors= ('b', 'g', 'r')
plt.figure
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the colour format of seun.jpg")

for i, col in enumerate(colors):
    hist = cv.calcHist([img], [i], None, [256], [0, 256])
    plt.plot(hist, color=col)
    plt.xlim([0, 256])

plt.show()
cv.waitKey(0)
```

# OUTPUT



# Computing Histograms in OpenCV

## Cont'd

**Example 2:** Compute the histogram for the BGR image version of “friend.jpg”.

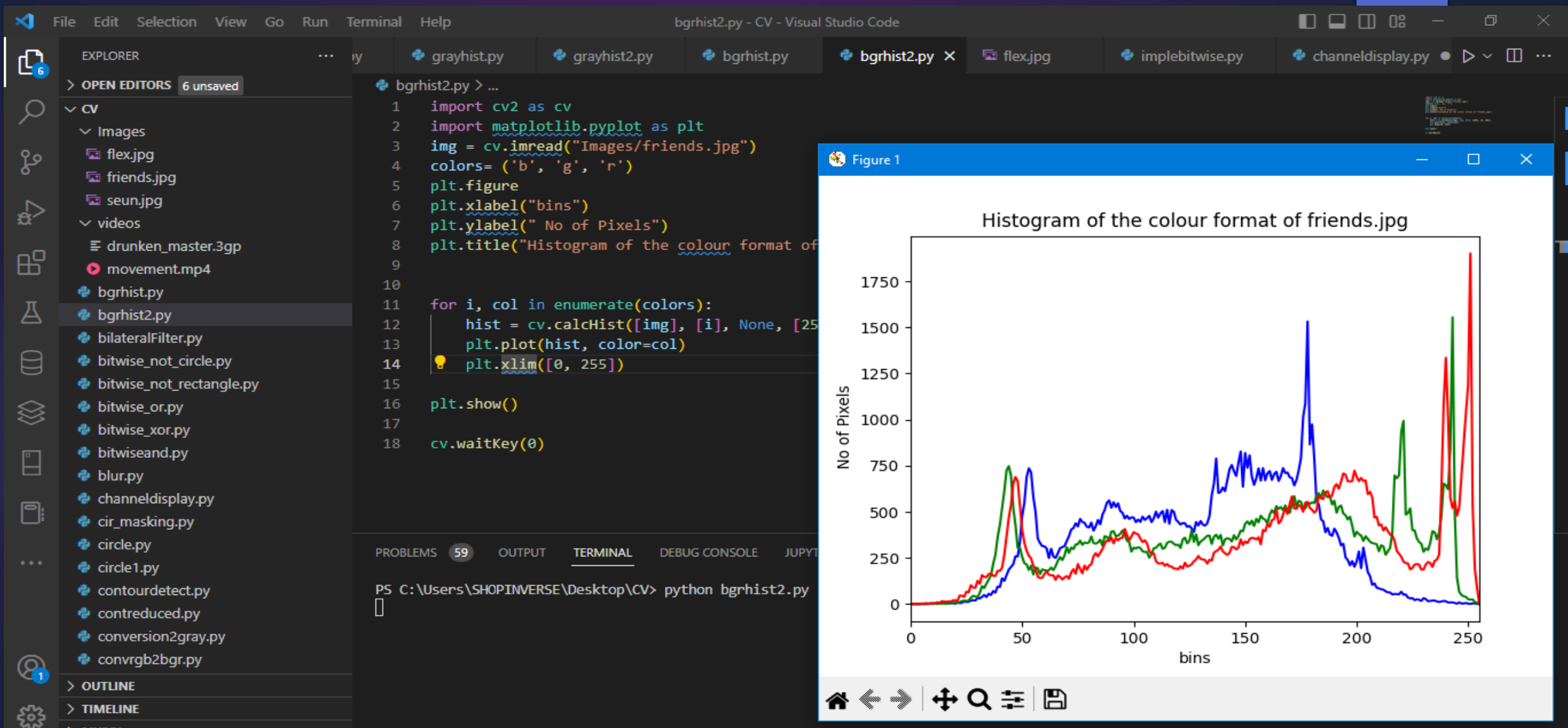
```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("Images/friends.jpg")
colors= ('b', 'g', 'r')
plt.figure
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the colour format of friends.jpg")

for i, col in enumerate(colors):
    hist = cv.calcHist([img], [i], None, [256], [0, 255])
    plt.plot(hist, color=col)
    plt.xlim([0, 255])

plt.show()

cv.waitKey(0)
```

# OUTPUT



# Computing Histograms in OpenCV

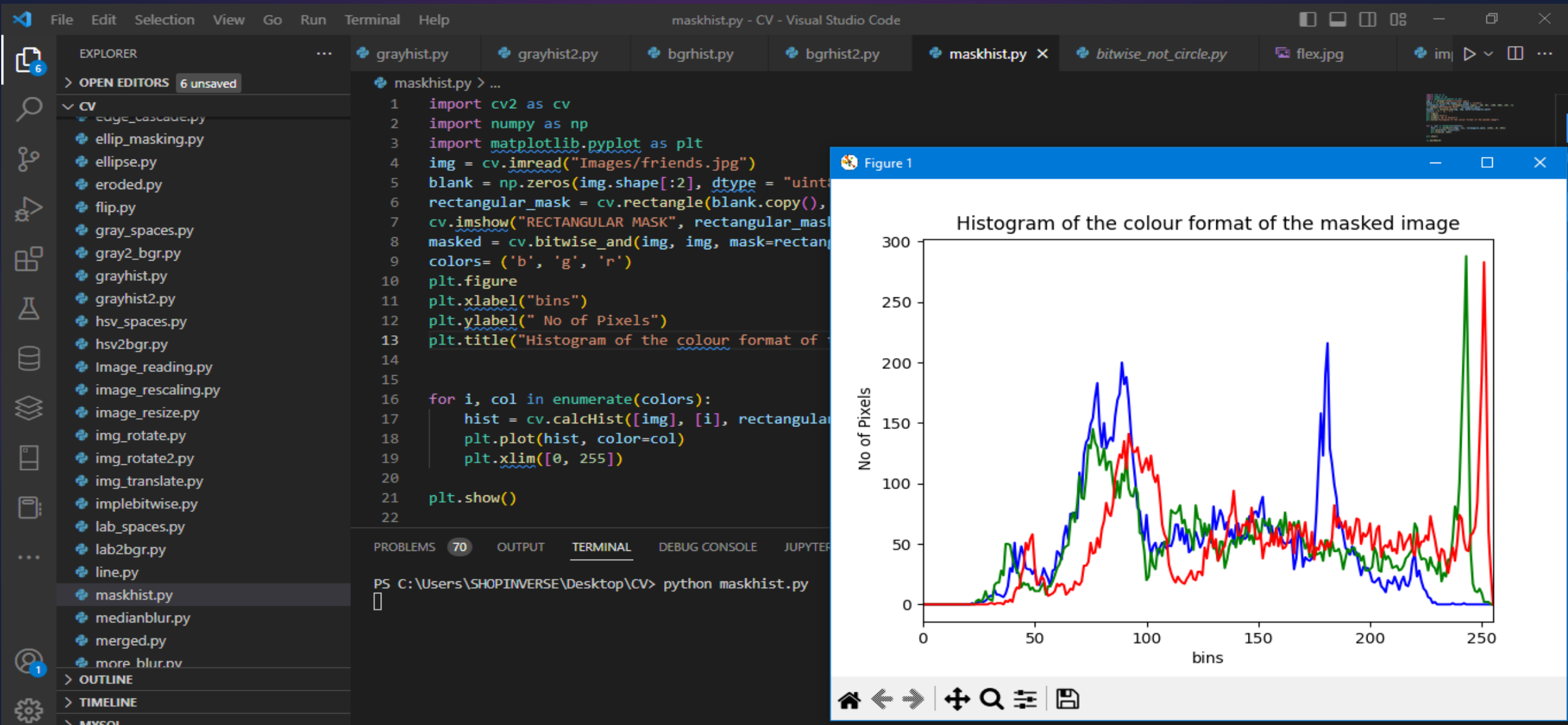
## Cont'd

**Example 3:** Compute the histogram for the BGR image version of “friend.jpg” which has been masked with a rectangular mask.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("Images/friends.jpg")
blank = np.zeros(img.shape[:2], dtype = "uint8")
rectangular_mask = cv.rectangle(blank.copy(), (30, 30), (100, 200), 255, -1)
cv.imshow("RECTANGULAR MASK", rectangular_mask)
masked = cv.bitwise_and(img, img, mask=rectangular_mask)
colors= ('b', 'g', 'r')
plt.figure
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the colour format of the masked image")

for i, col in enumerate(colors):
    hist = cv.calcHist([img], [i], rectangular_mask, [256], [0, 255])
    plt.plot(hist, color=col)
    plt.xlim([0, 255])
plt.show()
```

# OUTPUT





# THRESHOLDING IN OpenCV

Thresholding in OpenCV involves the binarization of an image. i.e. setting the image pixel to either dark or white. Threshold values are usually required to set intensity values for pixels. For example, if the intensities set are 125 and 255, then any pixel with intensities less than 125 will output dark colour while pixels with intensities greater than 125 or equal to 255 will output white.

**There are two basic thresholding techniques**

- ❖ Simple Thresholding
- ❖ Adaptive Thresholding

# THRESHOLDING IN OpenCV

Cont'd

## SIMPLE THRESHOLDING

This is a thresholding technique in which threshold values are not automatically chosen by the computer but requires human intervention to specify the threshold value to use.

## SYNTAX

```
threshold, thresh = cv2.threshold(src, threshold value, , maxvalue, threshold_type)
```

Upon the implementation of simple thresholding, two things are returned i.e. threshold and thresh. Thresh returns the binarized image while threshold returns the threshold values passed into the threshold function.



# THRESHOLDING IN OpenCV

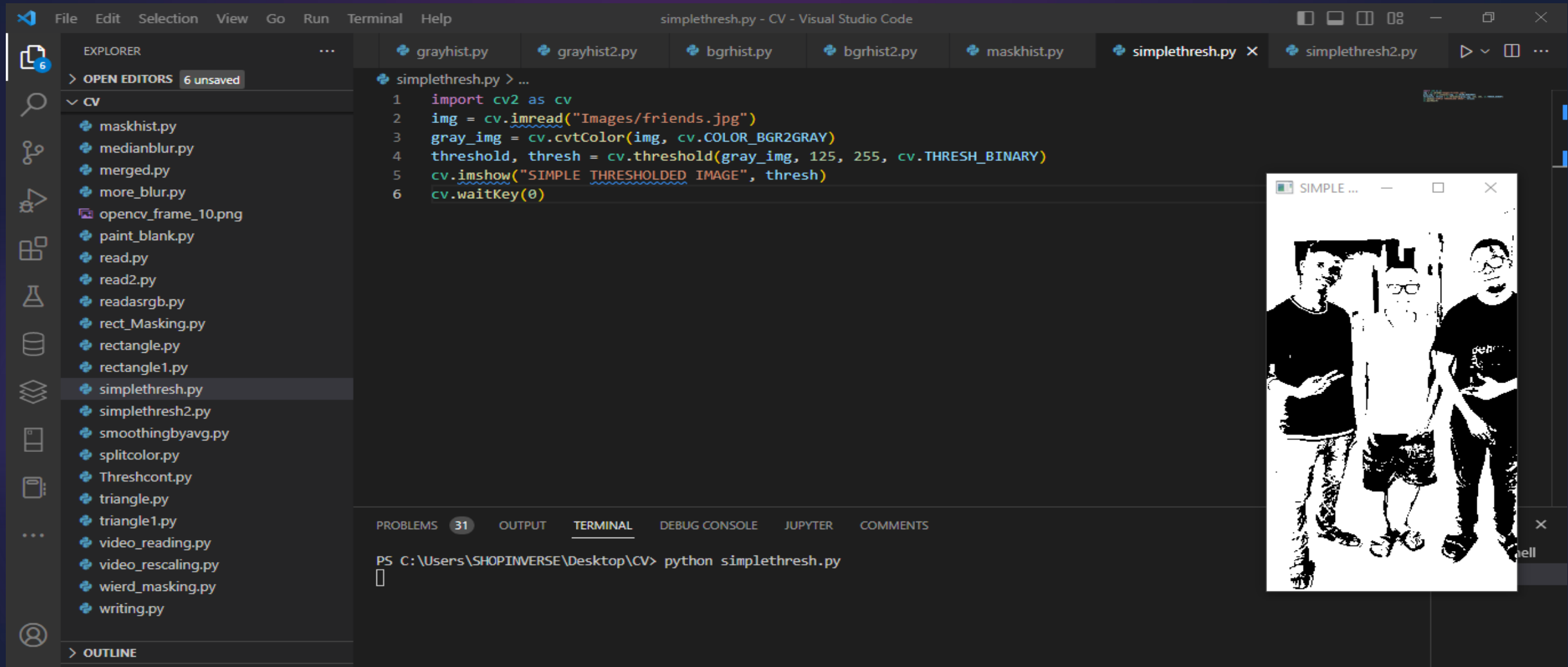
Cont'd

**Example 1:** Apply the simple thresholding function to “friends.jpg” provided that the threshold value should be 125 and the maxvalue should be 255.

## SOLUTION

```
import cv2 as cv
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
threshold, thresh = cv.threshold(gray_img, 125, 255, cv.THRESH_BINARY)
cv.imshow("SIMPLE THRESHOLDED IMAGE", thresh)
cv.waitKey(0)
```

# OUTPUT



# THRESHOLDING IN OpenCV

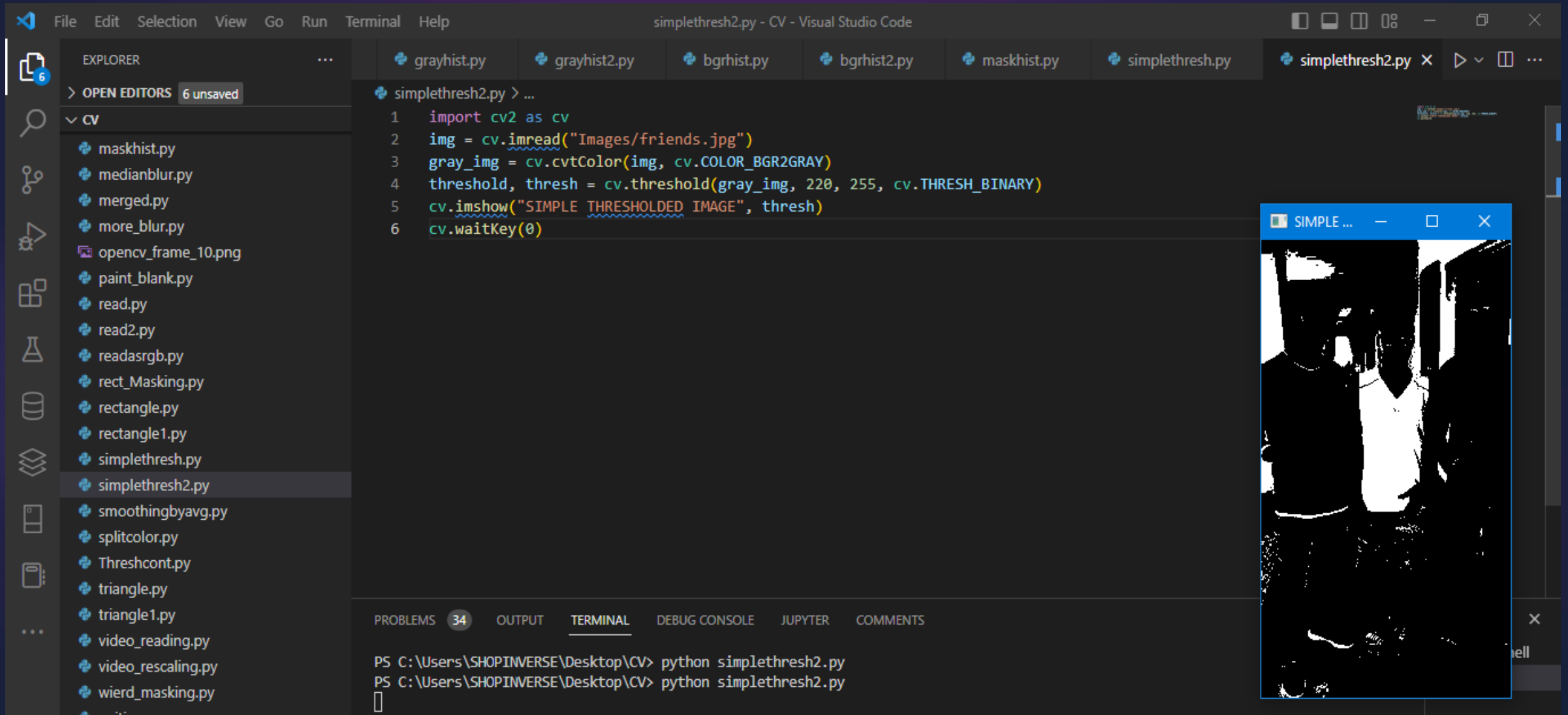
Cont'd

**Example 2:** Apply the simple thresholding function to “friends.jpg” provided that the threshold value should be 225 and the max Value should be 255.

## SOLUTION

```
import cv2 as cv
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
threshold, thresh = cv.threshold(gray_img, 225, 255, cv.THRESH_BINARY)
cv.imshow("SIMPLE THRESHOLDED IMAGE", thresh)
cv.waitKey(0)
```

# OUTPUT



# THRESHOLDING IN OpenCV

## Cont'd

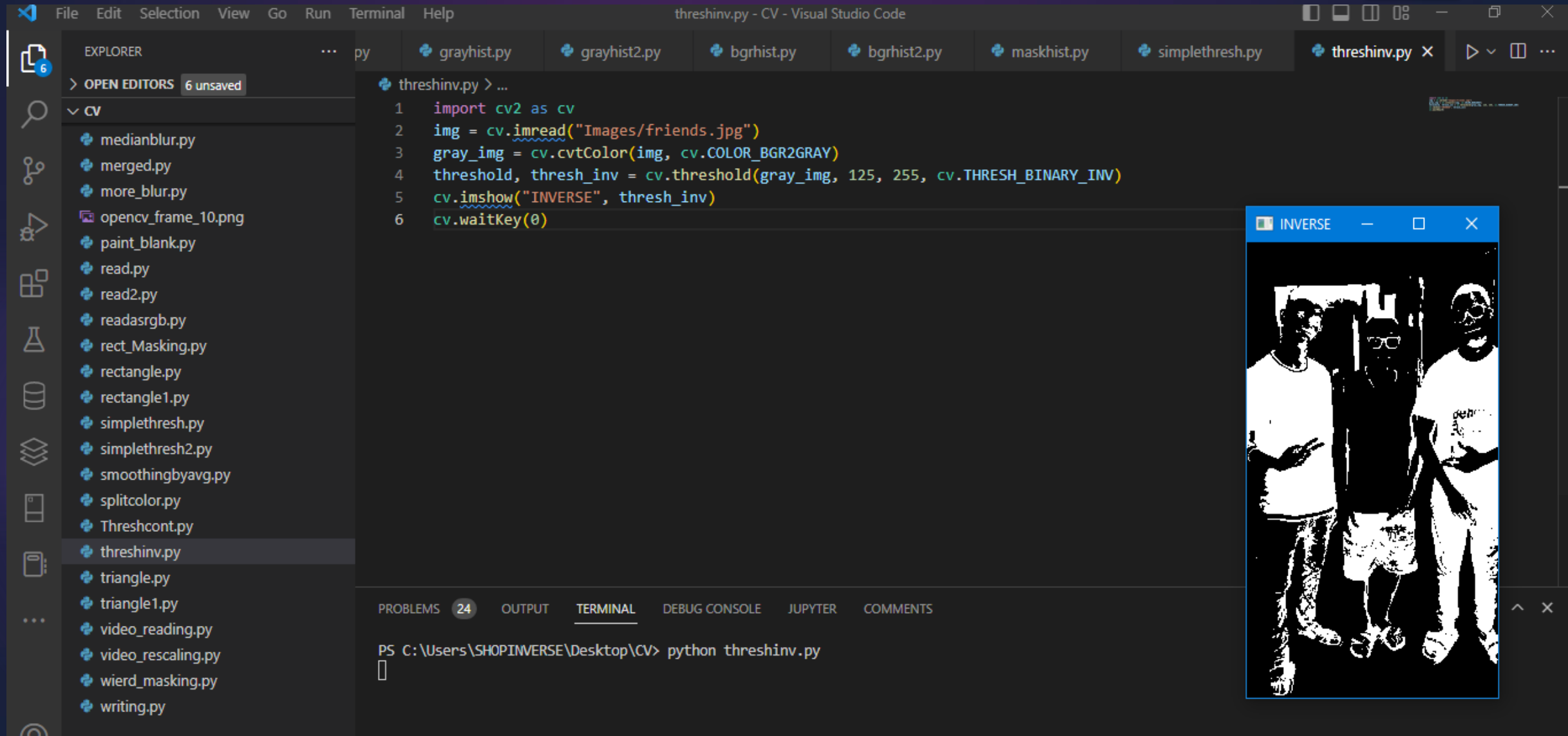
In Simple thresholding, inversion refers to the changing white colour of the thresholded image to black and changing the black colour to white.

The inversion process of the thresholded image involves the use of the `cv.THRESH_BINARY_INV` method in OpenCV to achieve that.

**Example:** obtain the inverse of the thresholded image in example 1

```
import cv2 as cv
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
threshold, thresh_inv = cv.threshold(gray_img, 125, 255, cv.THRESH_BINARY_INV)
cv.imshow("INVERSE", thresh_inv)
cv.waitKey(0)
```

# OUTPUT (Inverse of Simple Thresholded Image)



# THRESHOLDING IN OpenCV

Cont'd

## ADAPTIVE THRESHOLDING

Adaptive thresholding is a technique of dynamically calculating the threshold values for smaller regions of the image. Unlike Simple thresholding, adaptive thresholding does not need to manually specify the threshold values, the computer chooses the optimum threshold value automatically based on image pixel values and then binarizes the image.

## SYNTAX

```
adaptive = cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, cValue)
```



# THRESHOLDING IN OpenCV

Cont'd

- ▶ **src:** Source image we want to use
- ▶ **maxValue:** This represents the value that is to be given if the pixel value is more than the threshold value.
- ▶ **adaptiveMethod:** This represents the adaptive method to use. It could be `ADAPTIVE_THRESH_MEAN_C` which implies that the threshold value is the mean of the neighbourhood area. It could also be `ADAPTIVE_THRESH_GAUSSIAN_C` which implies the weighted sum of neighbourhood values where weights are Gaussian window.
- ▶ **blockSize:** This is the neighbourhood size of the kernel size which the OpenCV will use to compute the mean of the optimum threshold value.
- ▶ **cValue:** This is the value that helps us in fine-tuning the threshold. The cValue is usually subtracted from the mean value.



# THRESHOLDING IN OpenCV

## Cont'd

**Example:** Apply the adaptive threshold method to the image “friends.jpg”. Use the cv.ADAPTIVE\_THRESH\_MEAN\_C.

```
import cv2 as cv
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
adaptive = cv.adaptiveThreshold(gray_img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 3)
cv.imshow("ADAPTIVE THRESHOLDED IMAGE", adaptive)
cv.waitKey(0)
```

# OUTPUT

The screenshot displays the Visual Studio Code interface with the file `adaptive1.py` open. The Explorer sidebar on the left lists various image processing scripts, with `adaptive1.py` selected. The main editor window shows the following Python code:

```
1 import cv2 as cv
2 img = cv.imread("Images/friends.jpg")
3 gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
4 adaptive = cv.adaptiveThreshold(gray_img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 3)
5 cv.imshow("ADAPTIVE THRESHOLDED IMAGE", adaptive)
6 cv.waitKey(0)
7
```

Below the code editor, the TERMINAL panel shows the command `python adaptive1.py` being executed in a PowerShell prompt. To the right of the terminal, a small window titled `ADAPT...` displays the resulting binary image, which is a high-contrast, black and white version of the original image, showing the silhouettes of three people standing in a hallway.

# THRESHOLDING IN OpenCV

Cont'd

## Fine Tuning Adaptive Thresholded Images

Adaptive thresholding can be fine-tuned by subtracting a higher cValue from the mean value. Inversion of the image during this process will enable more features to be captured. The more the mean is subtracted, the more accurate the result becomes.

**Example:** Invert the thresholded image and use cValue = 7

```
import cv2 as cv
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
adaptive = cv.adaptiveThreshold(gray_img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY_INV, 11, 7)
cv.imshow("INVERSE THRESHOLDED IMAGE", adaptive)
cv.waitKey(0)
```

# OUTPUT

The screenshot displays the Visual Studio Code interface with the following components:

- EXPLORER:** A file tree on the left showing a project structure with folders 'cv' and 'videos'. The 'cv' folder contains image files (flex.jpg, friends.jpg, seun.jpg) and video files (drunken\_master.3gp, movement.mp4). The 'videos' folder contains several Python scripts for image processing, including `adaptive_inv.py`, which is currently selected.
- EDITOR:** The main workspace shows the code for `adaptive_inv.py`. The code imports `cv2` as `cv`, reads an image `friends.jpg`, converts it to grayscale, and applies adaptive thresholding using `cv.adaptiveThreshold` with parameters `255`, `cv.ADAPTIVE_THRESH_MEAN_C`, `cv.THRESH_BINARY_INV`, `11`, and `7`. The resulting image is displayed using `cv.imshow` with the title `"INVERSE THRESHOLDED IMAGE"`, and the program waits for a key press with `cv.waitKey(0)`.
- TERMINAL:** The bottom panel shows the command prompt output: `PS C:\Users\SHOPINVERSE\Desktop\CV> python adaptive_inv.py`, followed by a cursor on a new line.
- OUTPUT WINDOW:** A separate window titled `INVERS...` displays the result of the thresholding operation. It shows a high-contrast, black and white image of three people standing in a doorway, where the background is mostly black and the subjects are white.

# THRESHOLDING IN OpenCV

Cont'd

**Example:** Apply the adaptive threshold method to the image “friends.jpg”. Use the cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C.

```
import cv2 as cv
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
adaptivegauss = cv.adaptiveThreshold(gray_img, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 11, 3)
cv.imshow("ADAPTIVE THRESHOLDED IMAGE", adaptivegauss)
cv.waitKey(0)
```

# OUTPUT

The image shows a Visual Studio Code interface with the following components:

- EXPLORER:** A file tree on the left showing a project structure with folders 'cv' and 'videos'. The 'cv' folder contains images like 'flex.jpg', 'friends.jpg', and 'seun.jpg'. The 'videos' folder contains 'drunken\_master.3gp' and 'movement.mp4'. Below these are several Python files, including 'adaptivegaus.py' which is currently selected.
- EDITOR:** The main workspace shows the code for 'adaptivegaus.py':

```
1 import cv2 as cv
2 img = cv.imread("Images/friends.jpg")
3 gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
4 adaptivegauss = cv.adaptiveThreshold(gray_img, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 11, 3)
5 cv.imshow("ADAPTIVE THRESHOLDED IMAGE", adaptivegauss)
6 cv.waitKey(0)
```
- TERMINAL:** At the bottom, the terminal shows the command to run the script: `PS C:\Users\SHOPINVERSE\Desktop\CV> python adaptivegaus.py`, followed by a blank line indicating successful execution.
- OUTPUT:** A separate window titled 'ADAPT...' displays the result of the script: a high-contrast, black and white thresholded image of three people standing in a hallway.



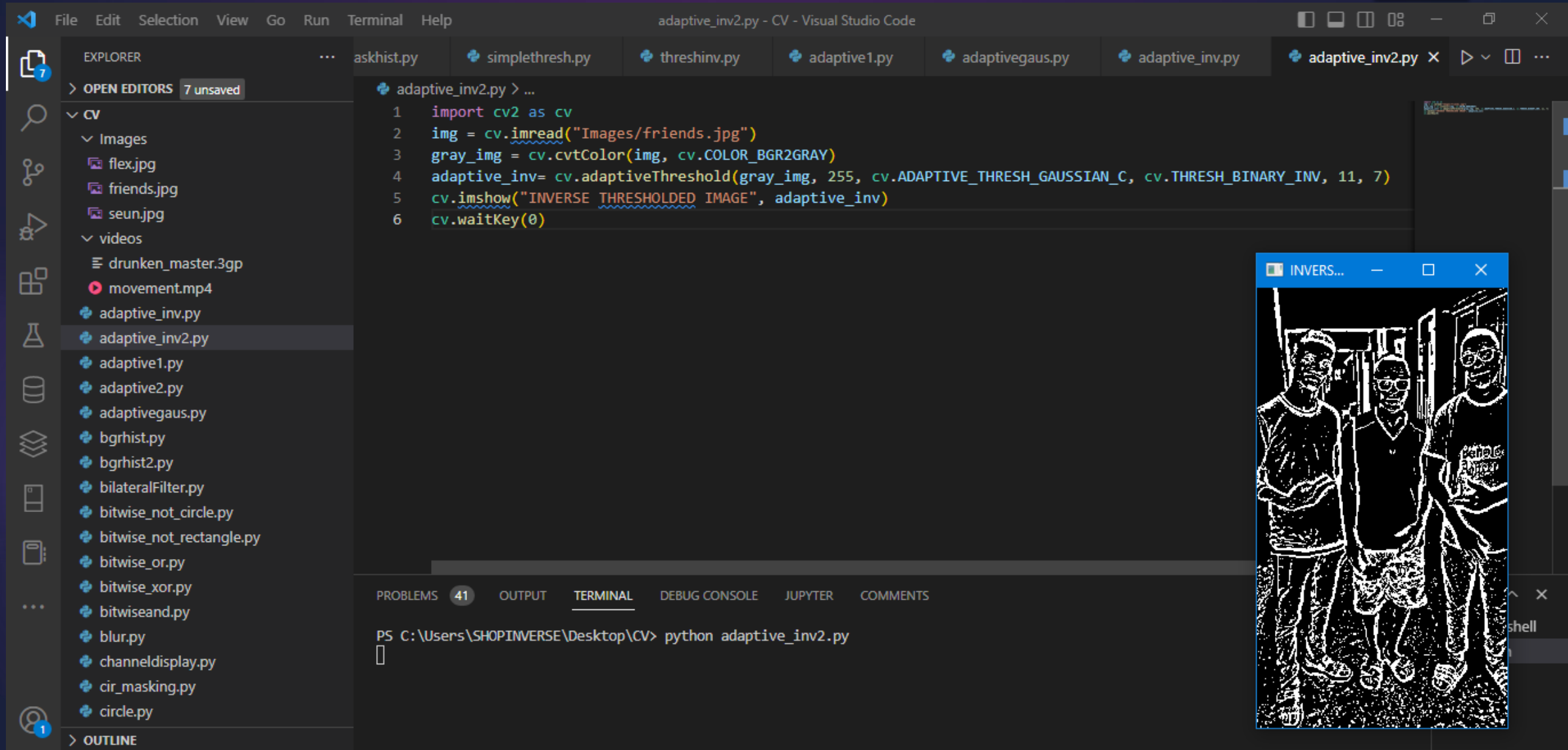
# THRESHOLDING IN OpenCV Cont'd

**Example:** Fine-tune the image by setting  $cValue = 7$  and invert the image but still retain the adaptive method used.

## SOLUTION

```
import cv2 as cv
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
adaptive_inv= cv.adaptiveThreshold(gray_img, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY_INV, 11, 7)
cv.imshow("INVERSE THRESHOLDED IMAGE", adaptive_inv)
cv.waitKey(0)
```

# OUTPUT





# EDGE DETECTION

There are several ways to compute edges in OpenCV. Some of these techniques are:

- ▶ Canny Edge Detector
- ▶ Laplacian Edge Detector
- ▶ Sobel Edge detector

**NB :** Canny edge detector was explained in one of the previous lectures.

# EDGE DETECTION

Cont'd

## LAPLACIAN EDGE DETECTOR

Laplacian edge detection helps lower the number of (pixels) to process by still maintaining the “structural” aspect of the image. By definition, Laplacian edge detectors are simply used to compute the second derivatives of an image. It uses just applies one symmetrical kernel over the image whose second derivative is to be computed. While implementing the laplacian edge detector, images are first converted to grayscale images before being subjected to the laplacian function in OpenCV. Also, some pixels which are read as negative pixels need to be converted to positive pixels. For this reason, the absolute method has to be employed using the NumPy module to get these negative pixels changed to positive pixels of image-specific datatype.

## SYNTAX

```
Lap = cv2.Laplacian(src, ddept)
```

```
Lap = np.uint8(np.absolute(lap))
```

# EDGE DETECTION

## Cont'd

**Example:** Compute the edges in the image named “friends.jpg” using the Laplacian edge detector.

### SOLUTION

```
import cv2 as cv
import numpy as np
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
lap_edge = cv.Laplacian(gray_img, cv.CV_64F)
lap_edge = np.uint8(np.absolute(lap_edge))
cv.imshow("LAPLACIAN", lap_edge)
cv.waitKey(0)
```

# OUTPUT

Visual Studio Code interface showing the execution of a Laplacian edge detection script.

**EXPLORER**

- OPEN EDITORS (7 unsaved)
- CV
  - grayniz2.py
  - hsv\_spaces.py
  - hsv2bgr.py
  - Image\_reading.py
  - image\_rescaling.py
  - image\_resize.py
  - img\_rotate.py
  - img\_rotate2.py
  - img\_translate.py
  - implebitwise.py
  - lab\_spaces.py
  - lab2bgr.py
  - laplacian.py**
  - line.py
  - maskhist.py
  - medianblur.py
  - merged.py
  - more\_blur.py
  - opencv\_frame\_10.png
  - paint\_blank.py
  - read.py
  - read2.py
  - readasrgb.py
  - rect\_Masking.py
  - rectangle.py
- OUTLINE
- TIMELINE

**laplacian.py**


```
1 import cv2 as cv
2 import numpy as np
3 img = cv.imread("Images/friends.jpg")
4 gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
5 lap_edge = cv.Laplacian(gray_img, cv.CV_64F)
6 lap_edge = np.uint8(np.absolute(lap_edge))
7 cv.imshow("LAPLACIAN", lap_edge)
8 cv.waitKey(0)
9
10
```

**TERMINAL**

```
PS C:\Users\SHOPINVERSE\Desktop\CV> python laplacian.py
```

**OUTPUT**

LAPLACIAN



# EDGE DETECTION

## Cont'd

### SOBEL EDGE DETECTOR

Sobel edge detector works differently in the sense that it can be used to compute the gradients of an image in the x-direction and y-direction. Depending on which direction we are interested in, we can find edges in the x-direction and y-direction. Also, we can compute the edges for both the x-direction and y-direction as a combined edge by using a bitwise operator on the `sobel_x` and `sobel_y` results.

### SYNTAX

```
Sobel = cv.Sobel(src, ddepth, x-direction, y-direction)
```

# EDGE DETECTION

Cont'd

**EXAMPLE:** Compute the edges of the image “friends.jpg” using the sobel technique in:

- (i) x- direction
- (ii) y-direction
- (iii) Combined x- and y- direction

# EDGE DETECTION

## Cont'd

(i)

```
import cv2 as cv
import numpy as np
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
sobel_x_dir = cv.Sobel(gray_img, cv.CV_64F, 1, 0)
cv.imshow("SOBEL X DIRECTION", sobel_x_dir)
cv.waitKey(0)
```

(ii)

```
import cv2 as cv
import numpy as np
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
sobel_y_dir = cv.Sobel(gray_img, cv.CV_64F, 0, 1)
cv.imshow("SOBEL Y DIRECTION", sobel_y_dir)
cv.waitKey(0)
```

(iii)

```
import cv2 as cv
import numpy as np
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
sobel_x = cv.Sobel(gray_img, cv.CV_64F, 1, 0)
sobel_y = cv.Sobel(gray_img, cv.CV_64F, 0, 1)
sobelcombo = cv.bitwise_or(sobel_x, sobel_y)
cv.imshow("SOBEL COMBINED", sobelcombo)
cv.waitKey(0)
```



# OUTPUT (i)

Visual Studio Code interface showing the execution of a Sobel edge detection script.

**EXPLORER**

- CV
  - more\_blur.py
  - opencv\_frame\_10.png
  - paint\_blank.py
  - read.py
  - read2.py
  - readasrgb.py
  - rect\_Masking.py
  - rectangle.py
  - rectangle1.py
  - simplethresh.py
  - simplethresh2.py
  - smoothingbyavg.py
  - sobel\_x.py**
  - sobel\_y.py
  - splitcolor.py
  - Threshcont.py
  - threshinv.py
  - triangle.py
  - triangle1.py
  - video\_reading.py
  - video\_rescaling.py
  - wierd\_masking.py
  - writing.py

**Code Editor (sobel\_x.py):**

```
1 import cv2 as cv
2 import numpy as np
3 img = cv.imread("Images/friends.jpg")
4 gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
5 sobel_x_dir = cv.Sobel(gray_img, cv.CV_64F, 1, 0)
6 cv.imshow("SOBEL X DIRECTION", sobel_x_dir)
7 cv.waitKey(0)
```

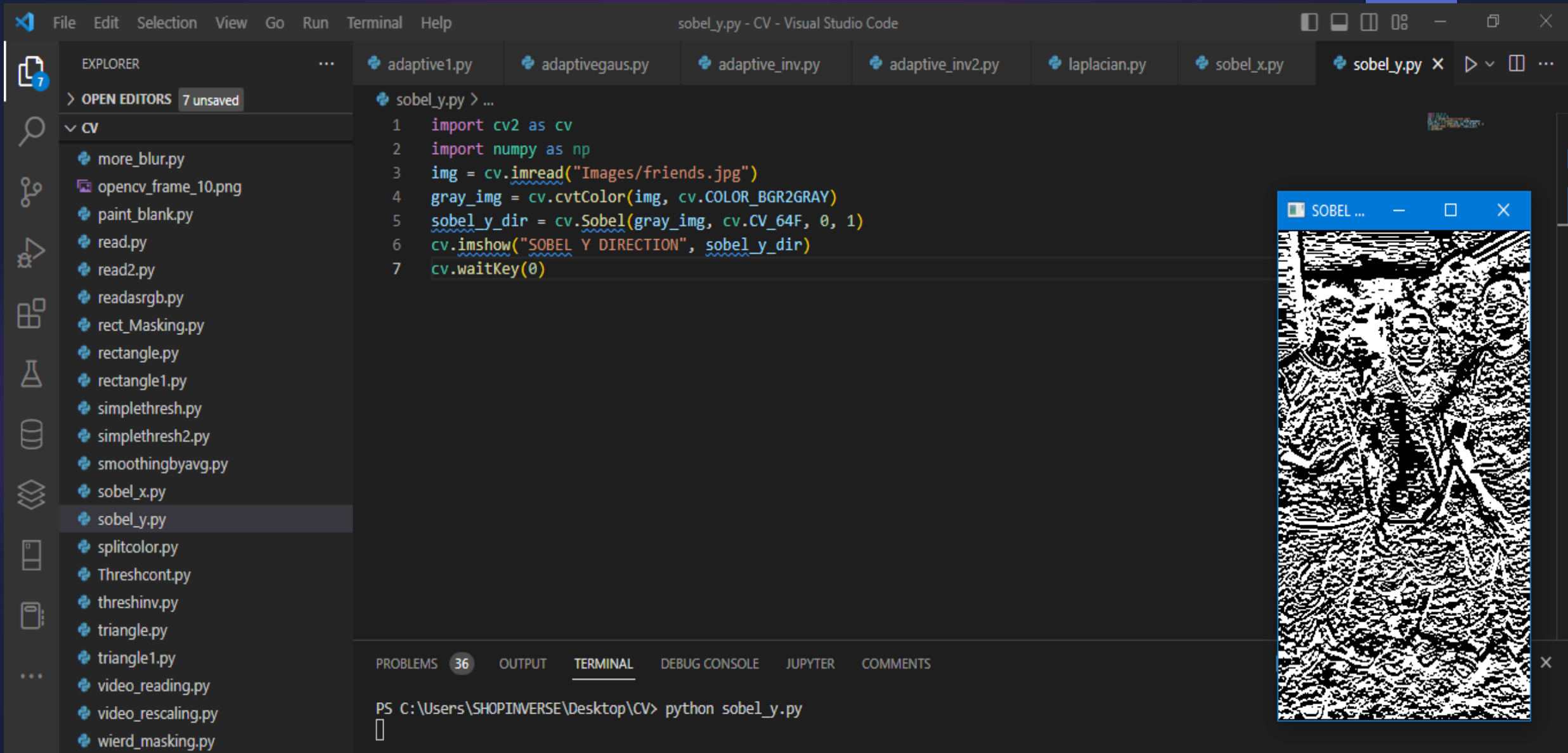
**Terminal:**

```
PS C:\Users\SHOPINVERSE\Desktop\CV> python sobel_x.py
```

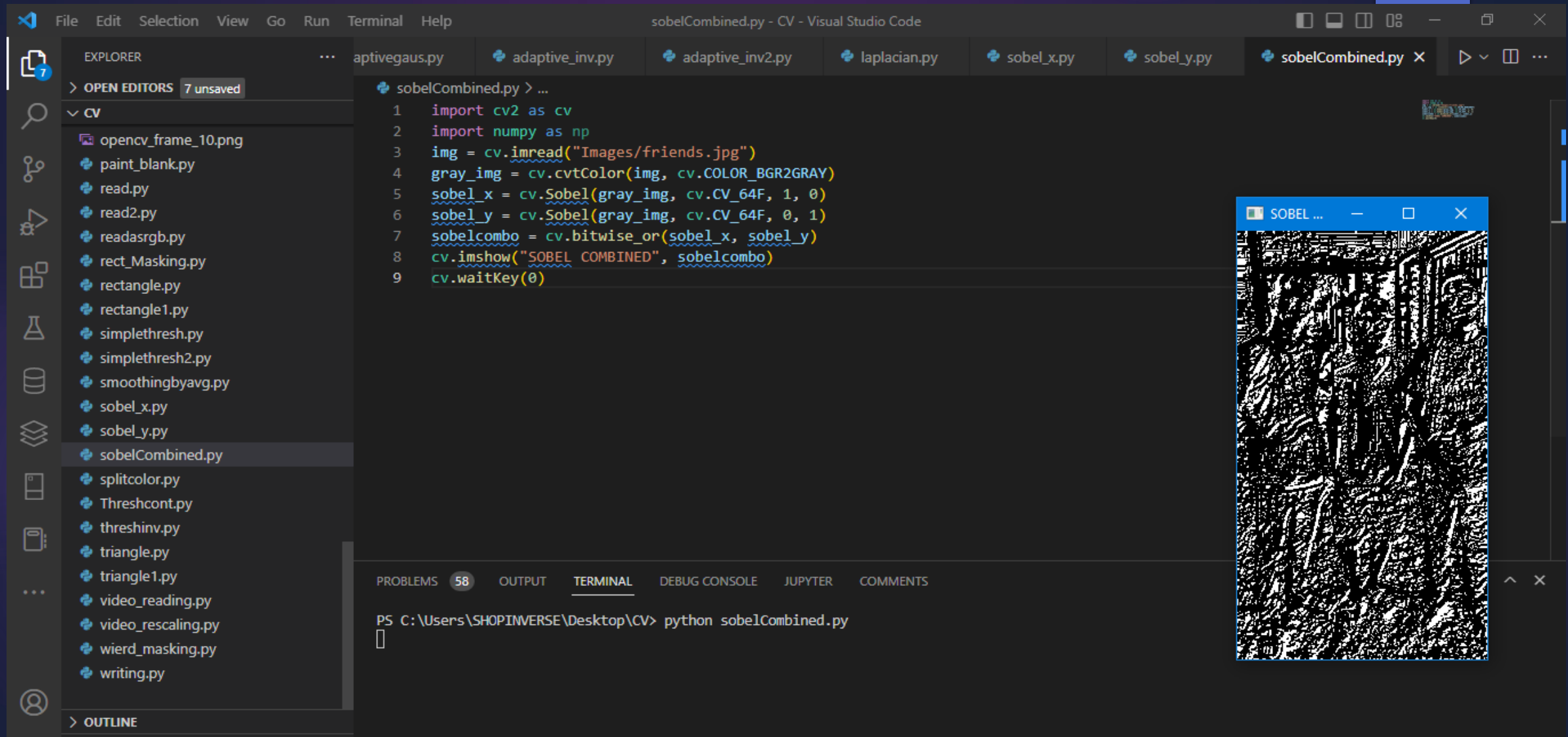
**Output:** A window titled "SOBEL ..." displays the resulting edge detection output, showing a high-contrast, black and white image of three people.



# OUTPUT (ii)



# OUTPUT (iii)



Visual Studio Code interface showing the execution of a Sobel edge detection script.

**EXPLORER** (7 unsaved)

- opencv\_frame\_10.png
- paint\_blank.py
- read.py
- read2.py
- readasrgb.py
- rect\_Masking.py
- rectangle.py
- rectangle1.py
- simplethresh.py
- simplethresh2.py
- smoothingbyavg.py
- sobel\_x.py
- sobel\_y.py
- sobelCombined.py**
- splitcolor.py
- Threshcont.py
- threshinv.py
- triangle.py
- triangle1.py
- video\_reading.py
- video\_rescaling.py
- wierd\_masking.py
- writing.py

**CODE EDITOR** (sobelCombined.py)

```
1 import cv2 as cv
2 import numpy as np
3 img = cv.imread("Images/friends.jpg")
4 gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
5 sobel_x = cv.Sobel(gray_img, cv.CV_64F, 1, 0)
6 sobel_y = cv.Sobel(gray_img, cv.CV_64F, 0, 1)
7 sobelcombo = cv.bitwise_or(sobel_x, sobel_y)
8 cv.imshow("SOBEL COMBINED", sobelcombo)
9 cv.waitKey(0)
```

**TERMINAL**

```
PS C:\Users\SHOPINVERSE\Desktop\CV> python sobelCombined.py
```

**SOBEL ...** (Output Image)

**THANKS FOR VIEWING**