ORMEDIAN RESEARCH INSTITUTE

TOPIC: : OpenCV Tutorials (part eight)

By:

ADEKOLA OLUWASEUN O.

February 26, 2023

TABLE OF CONTENT

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.

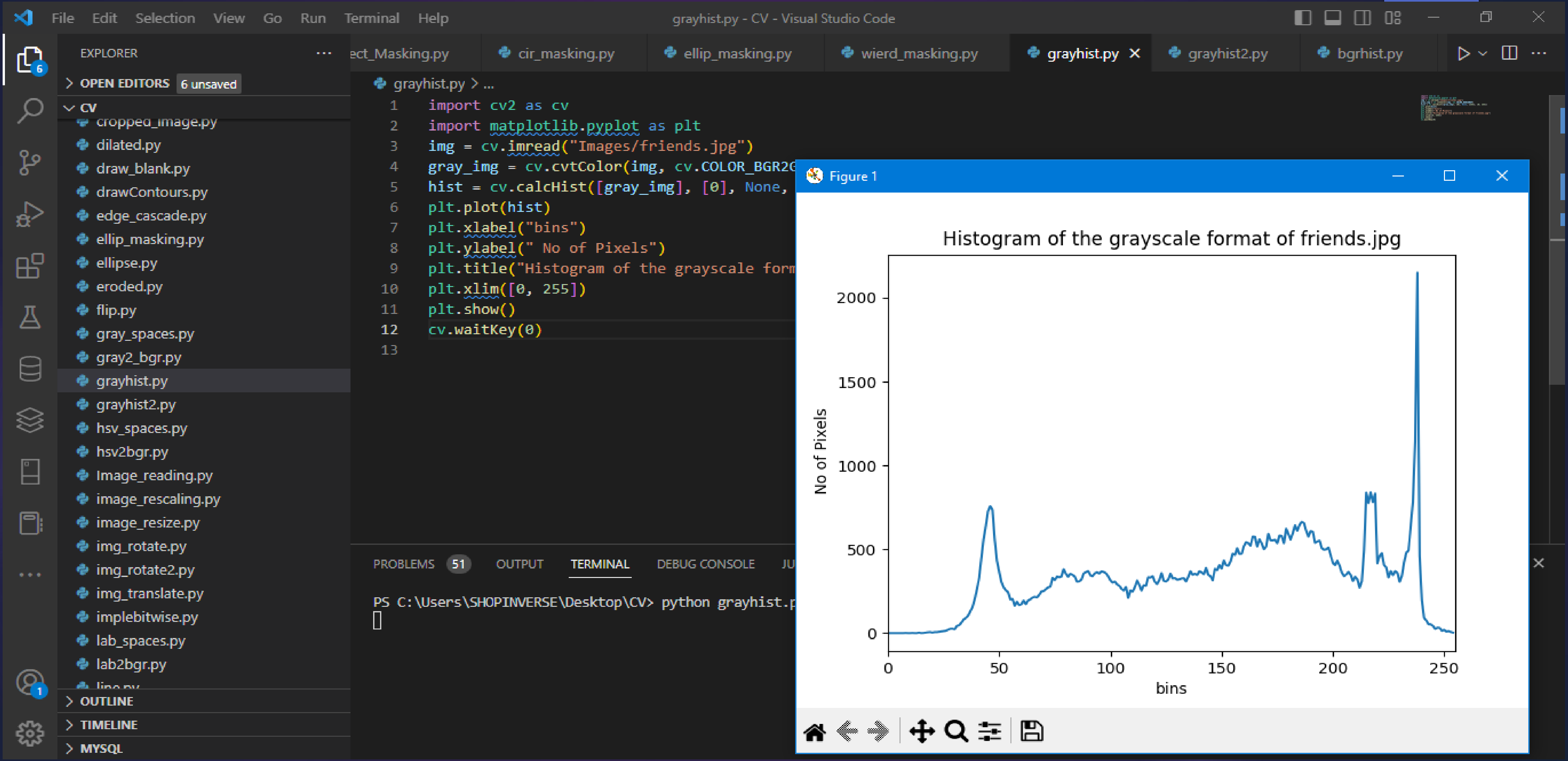
histSize: This is used to specify the number of bins to use during histogram computation.

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



```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2G
hist = cv.calcHist([gray_img], [0], None,
plt.plot(hist)
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the grayscale form
plt.xlim([0, 255])
plt.show()
cv.waitKey(0)
```

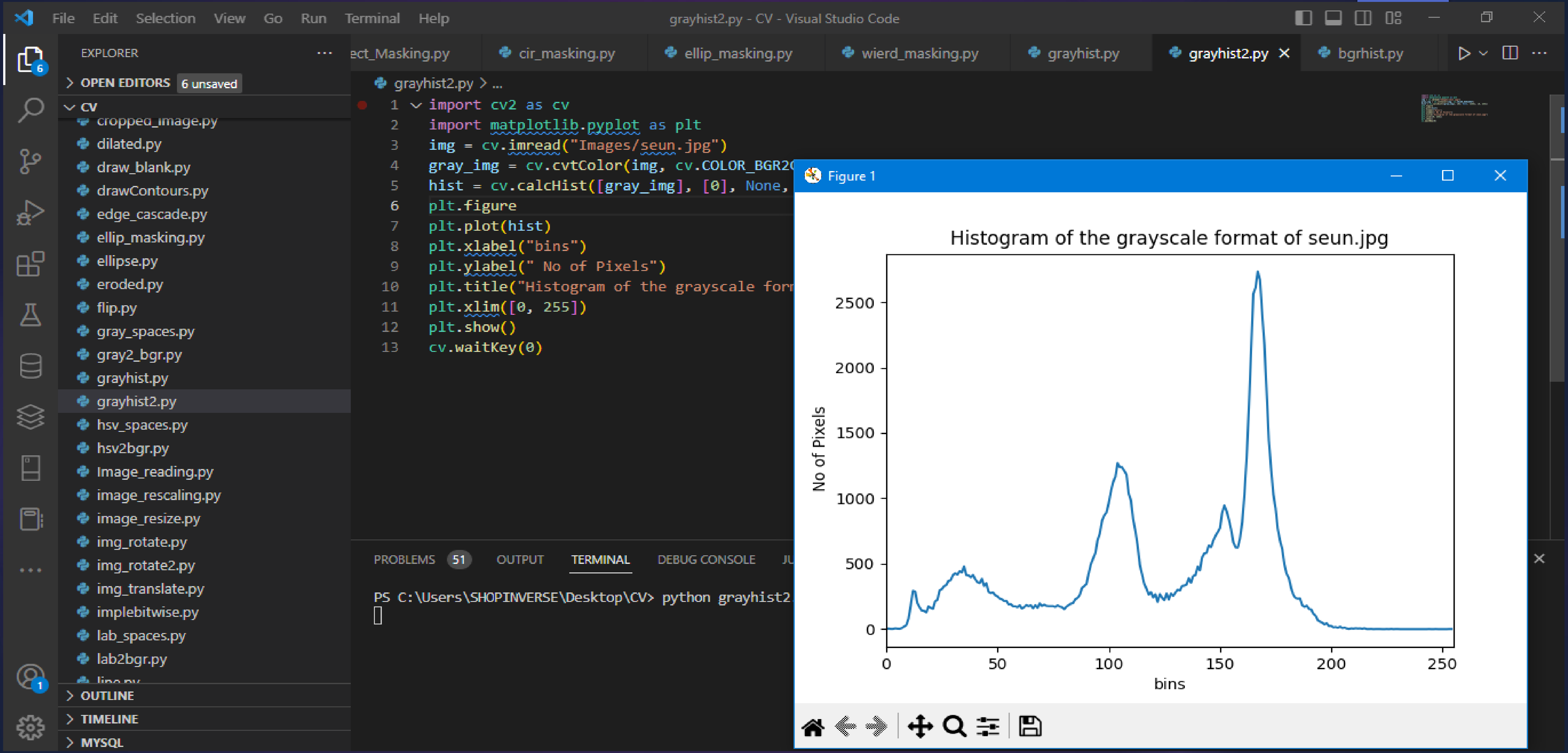
```
PS C:\Users\SHOPINVERSE\Desktop\CV> python grayhist.p
```

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



```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("Images/seun.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2
hist = cv.calcHist([gray_img], [0], None,
plt.figure
plt.plot(hist)
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the grayscale for
plt.xlim([0, 255])
plt.show()
cv.waitKey(0)
```

```
PS C:\Users\SHOPINVERSE\Desktop\CV> python grayhist2
```

Computing Histograms in OpenCV Cont’d

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

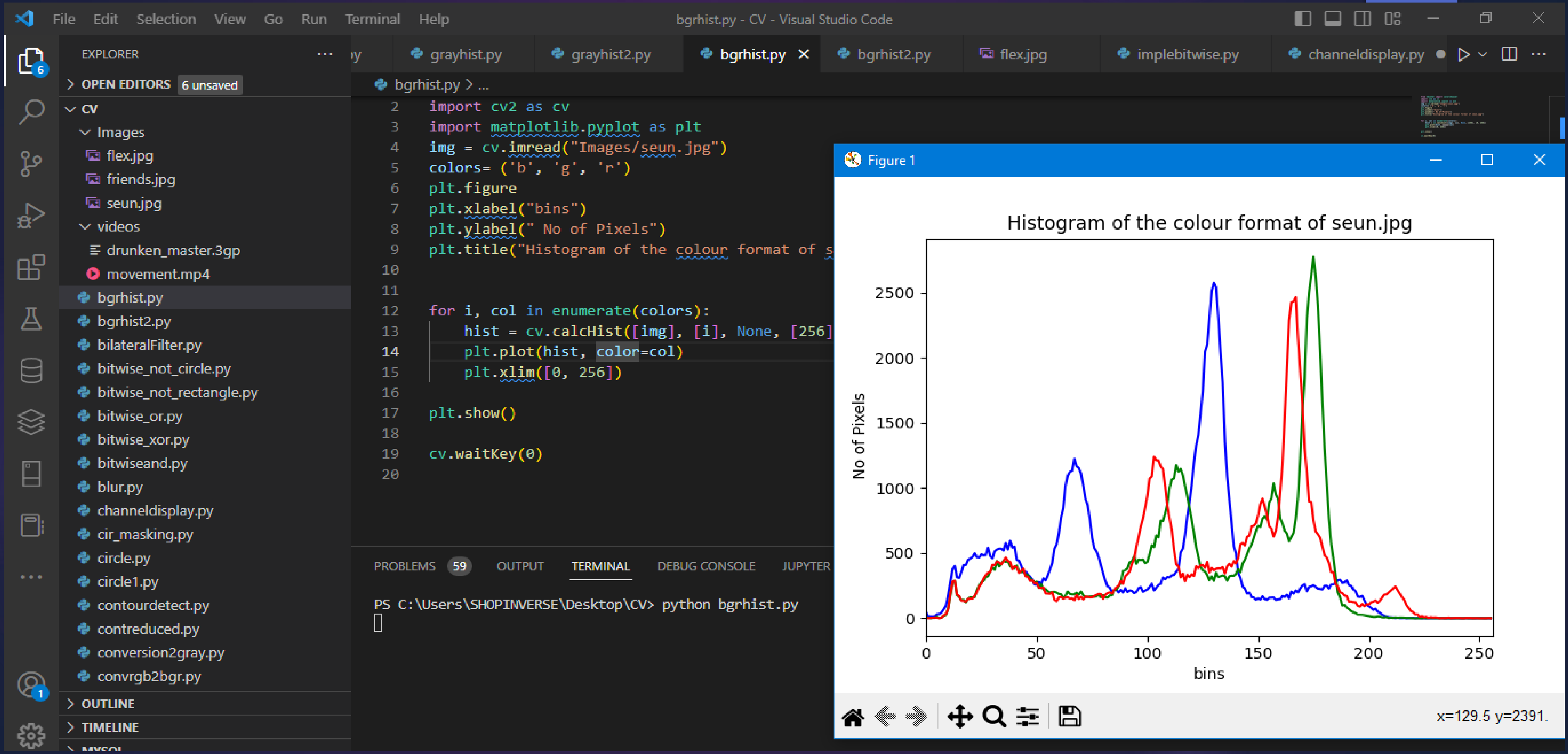
Example 2: Compute the histogram for the BGR image version of “seun.jpg”.

```
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



```python
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 s


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

plt.show()

cv.waitKey(0)
```

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

Computing Histograms in OpenCV Cont’d

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

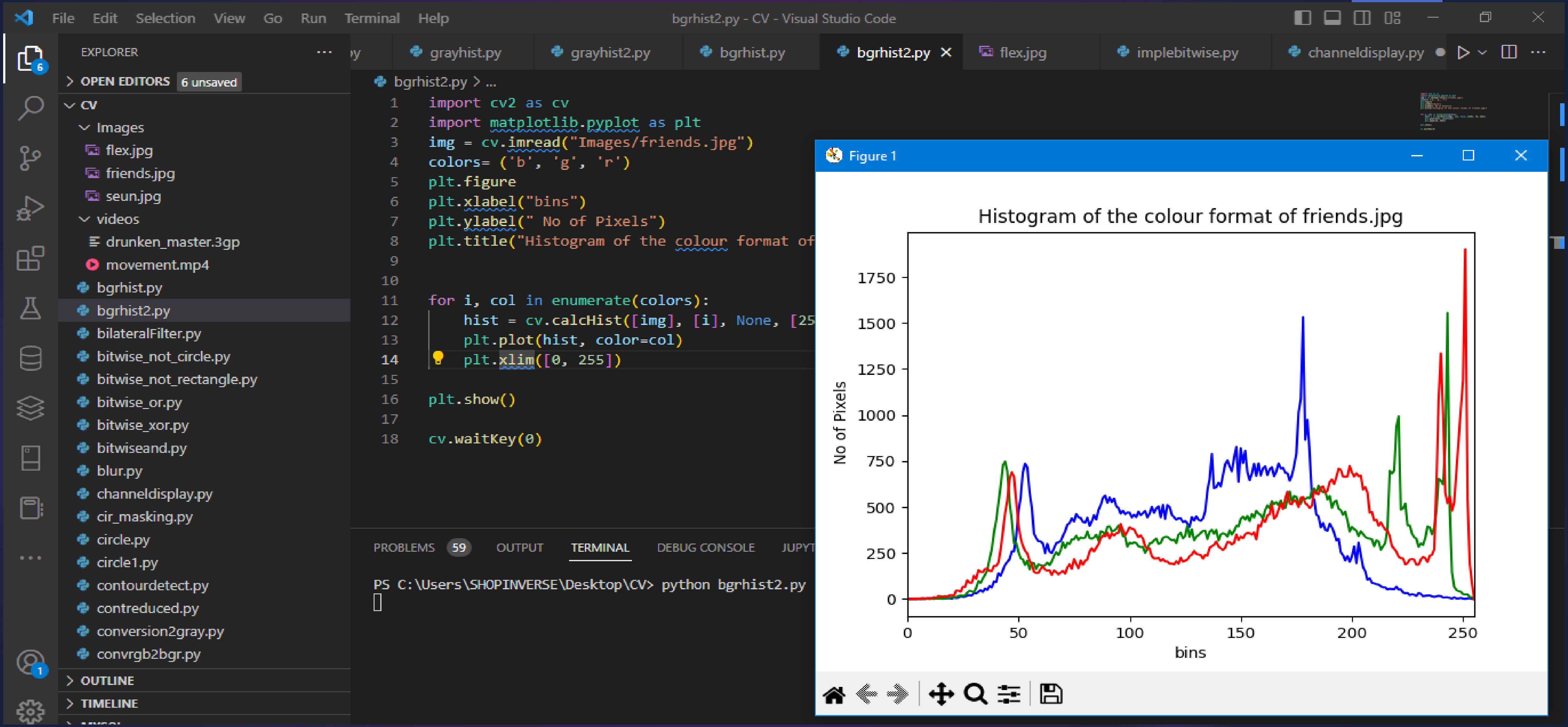
```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("Images/friends.jpg")
colors= ('b', 'g', 'r')
plt.figure
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the colour format of friends.jpg")

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



```
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


for i, col in enumerate(colors):
    hist = cv.calcHist([img], [i], None, [25
    plt.plot(hist, color=col)
    plt.xlim([0, 255])

plt.show()

cv.waitKey(0)
```

```
PS C:\Users\SHOPINVERSE\Desktop\CV> python bgrhist2.py
```

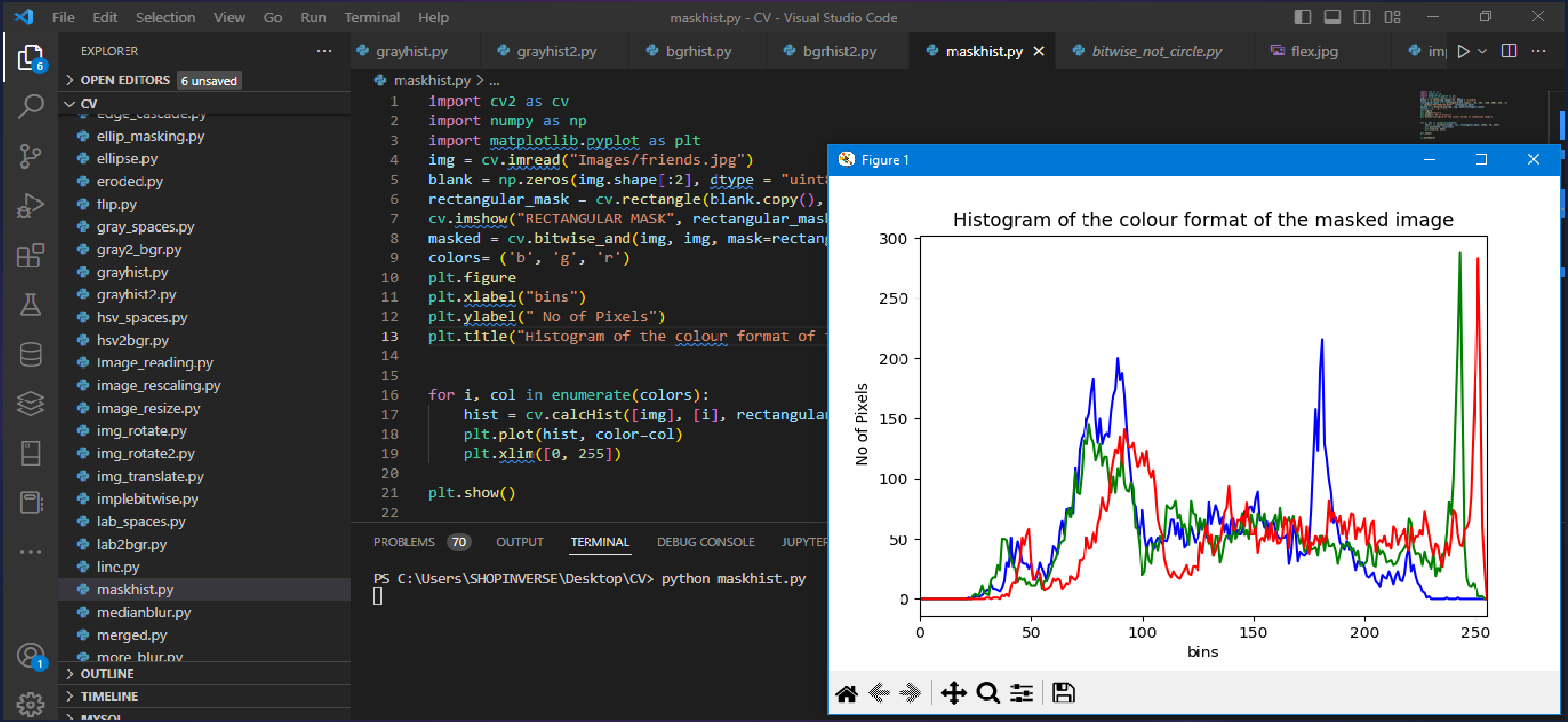
Computing Histograms in OpenCV Cont’d

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



```python
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 = "uint
rectangular_mask = cv.rectangle(blank.copy(),
cv.imshow("RECTANGULAR MASK", rectangular_mas
masked = cv.bitwise_and(img, img, mask=rectan
colors= ('b', 'g', 'r')
plt.figure
plt.xlabel("bins")
plt.ylabel(" No of Pixels")
plt.title("Histogram of the colour format of


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

plt.show()
```

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

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, , maxalue, 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.

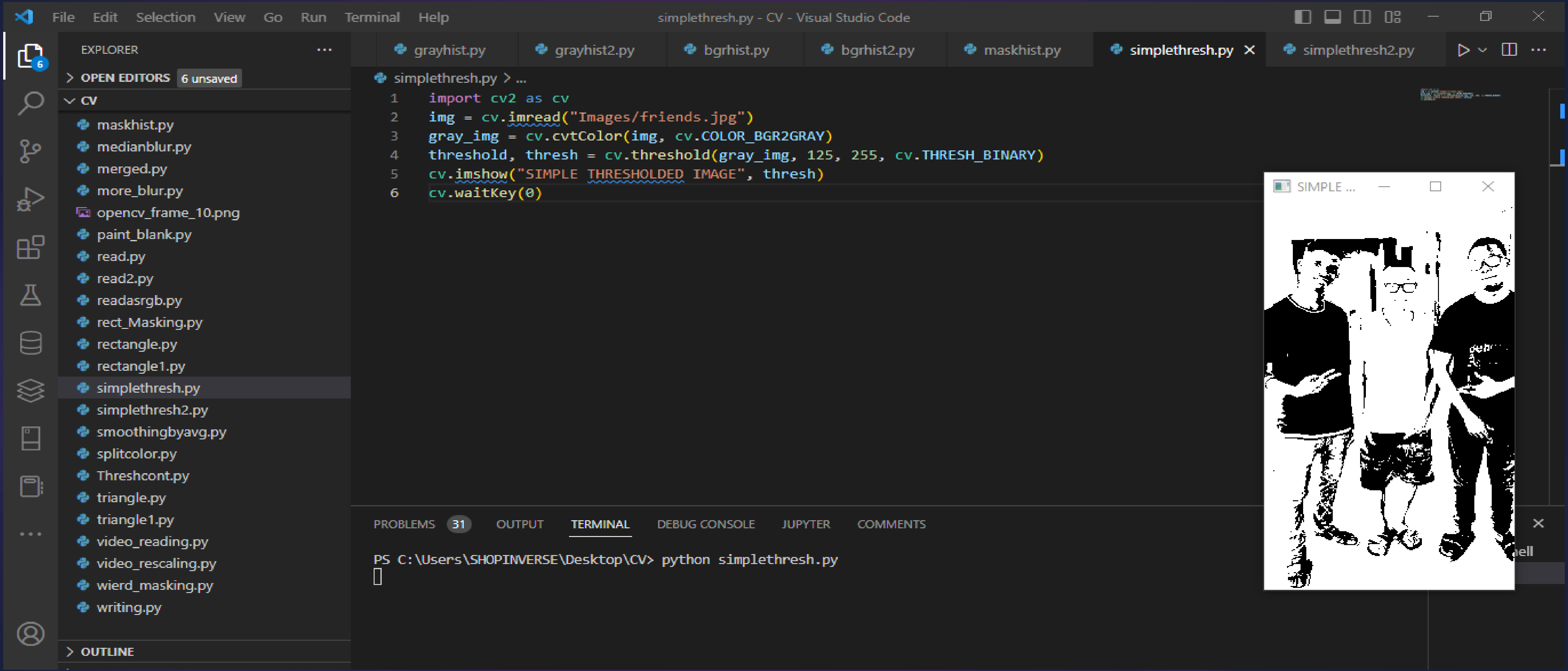
THRESHOLDING IN OpenCV Cont'd

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



```python
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)
```

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

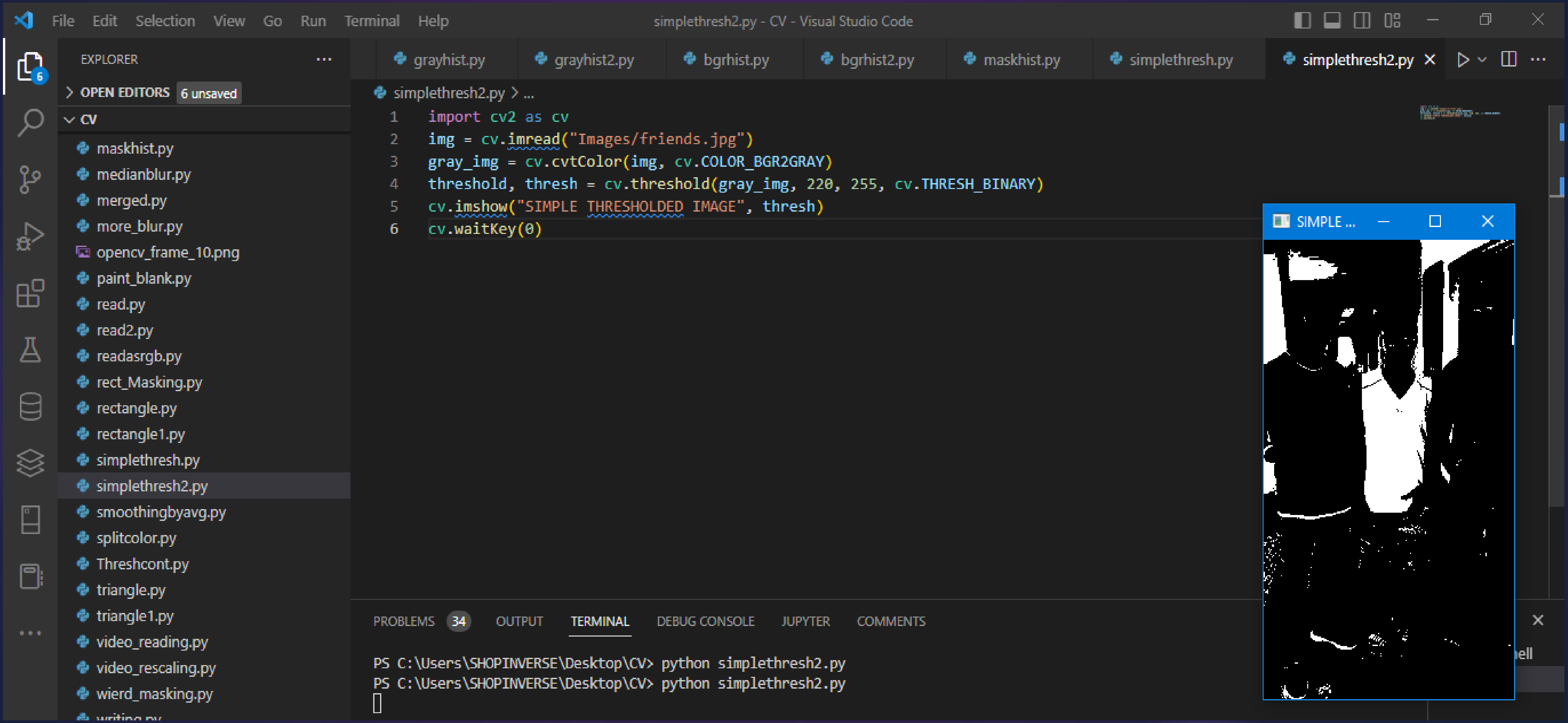
THRESHOLDING IN OpenCV Cont’d

Example 2: Apply the simple thresholding function to “friends.jpg” provided that the threshold value should be 225 and the 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, 225, 255, cv.THRESH_BINARY)
cv.imshow("SIMPLE THRESHOLDED IMAGE", thresh)
cv.waitKey(0)
```

OUTPUT



```python
import cv2 as cv
img = cv.imread("Images/friends.jpg")
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
threshold, thresh = cv.threshold(gray_img, 220, 255, cv.THRESH_BINARY)
cv.imshow("SIMPLE THRESHOLDED IMAGE", thresh)
cv.waitKey(0)
```

```
PS C:\Users\SHOPINVERSE\Desktop\CV> python simplethresh2.py
PS C:\Users\SHOPINVERSE\Desktop\CV> python simplethresh2.py
```

THRESHOLDING IN OpenCV Cont’d

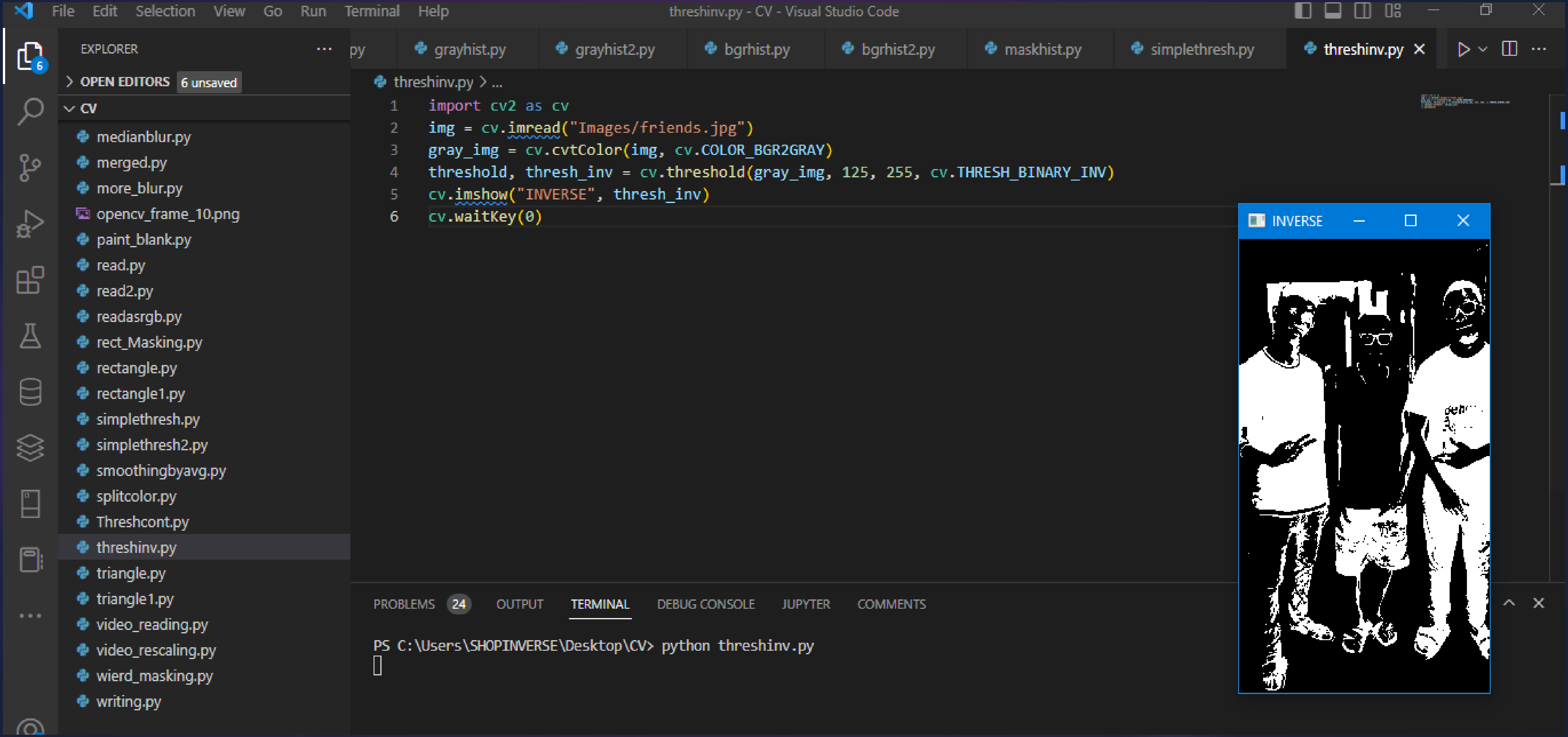
In Simple thresholding, inversion refers to the changing white colour of the thresholded image to black and changing the black colour to white.

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)



```python
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)
```

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

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

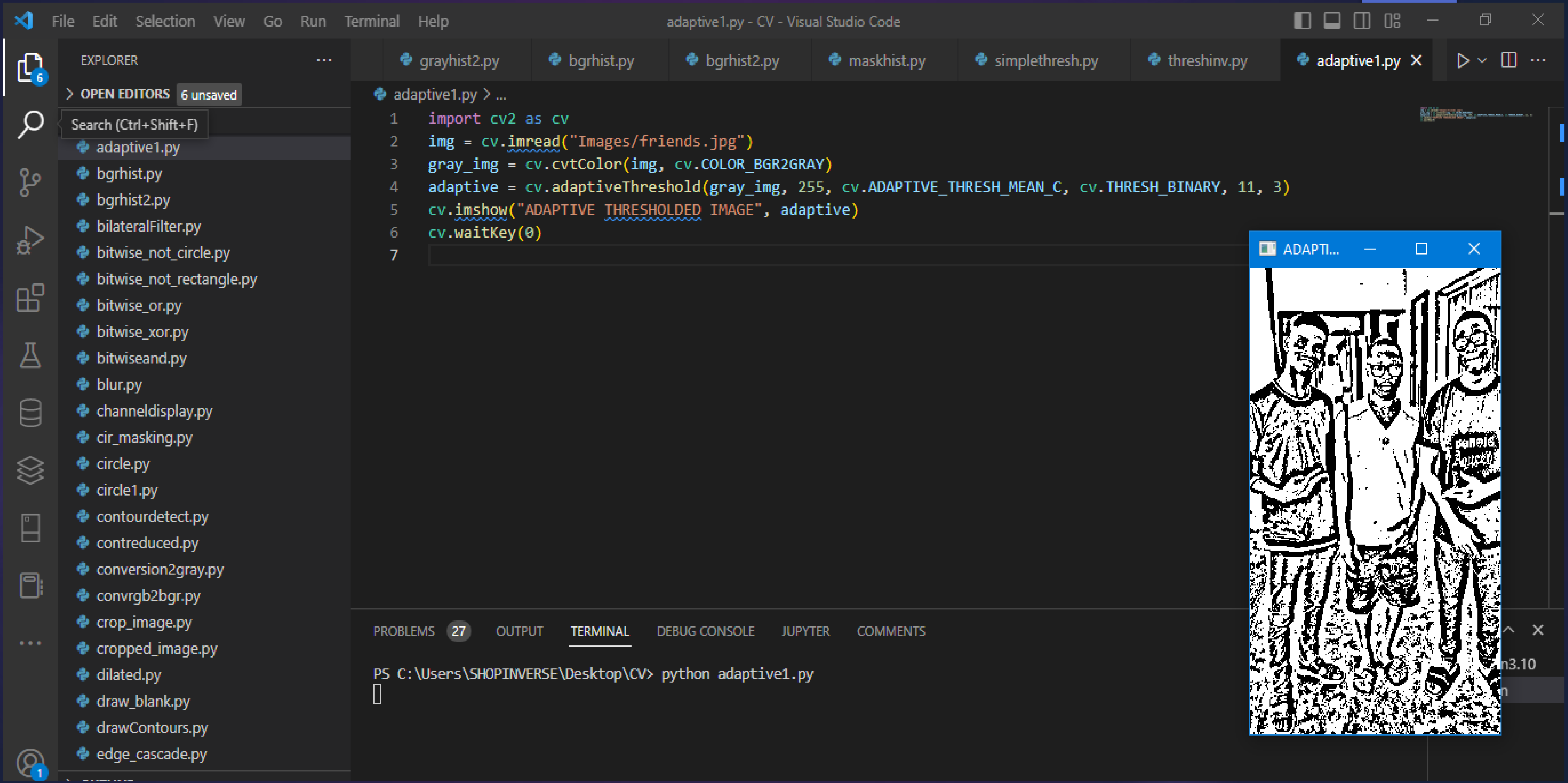
- **src:** Source image we want to use
- **maxValue:** This represents the value that is to be given if the pixel value is more than the threshold value.
- **adaptiveMethod:** This represents the adaptive method to use. It could be ADAPTIVE_THRESH_MEAN_C which implies that the threshold value is the 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



```python
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)
```

```
PS C:\Users\SHOPINVERSE\Desktop\CV> python adaptive1.py
```

THRESHOLDING IN OpenCV Cont'd

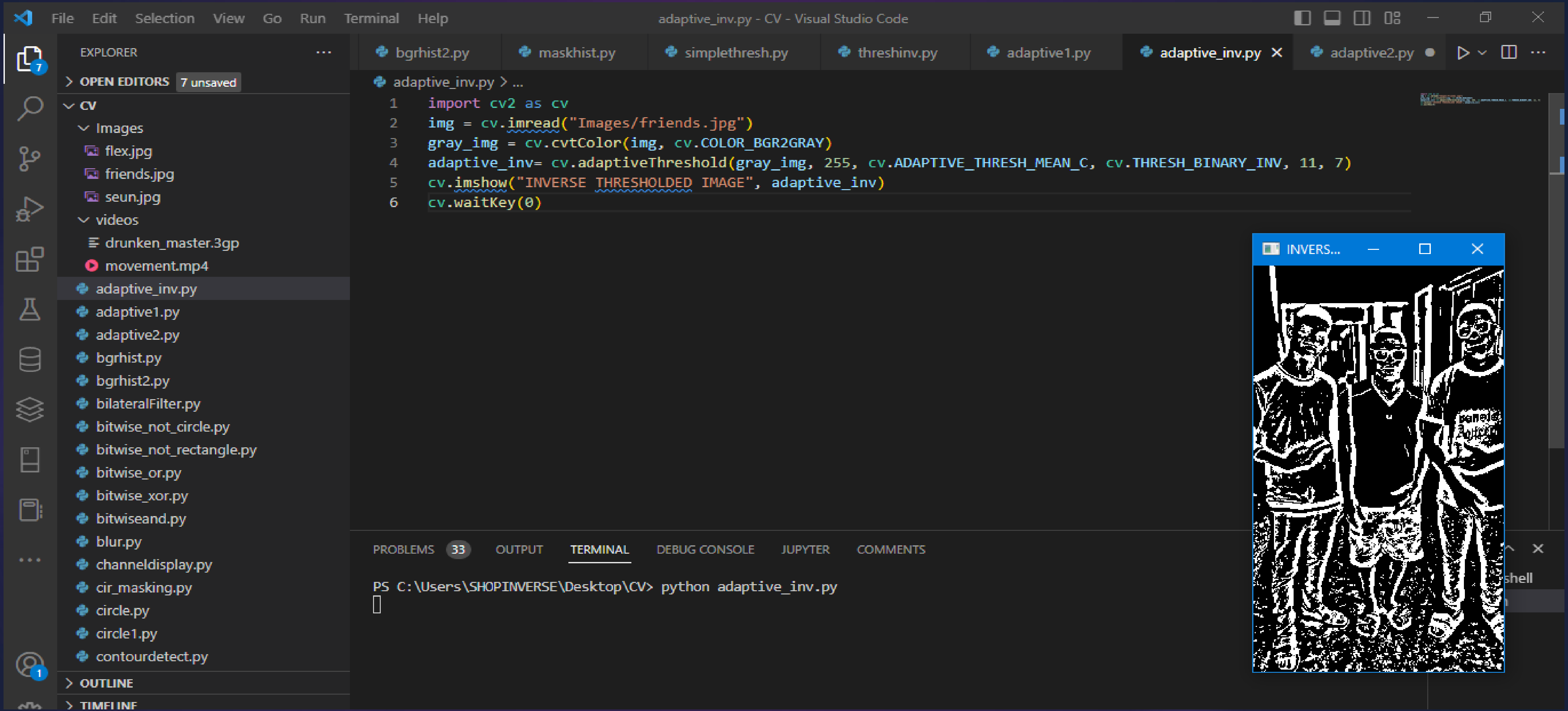
Fine Tuning Adaptive Thresholded Images

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



```python
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_MEAN_C, cv.THRESH_BINARY_INV, 11, 7)
cv.imshow("INVERSE THRESHOLDED IMAGE", adaptive_inv)
cv.waitKey(0)
```

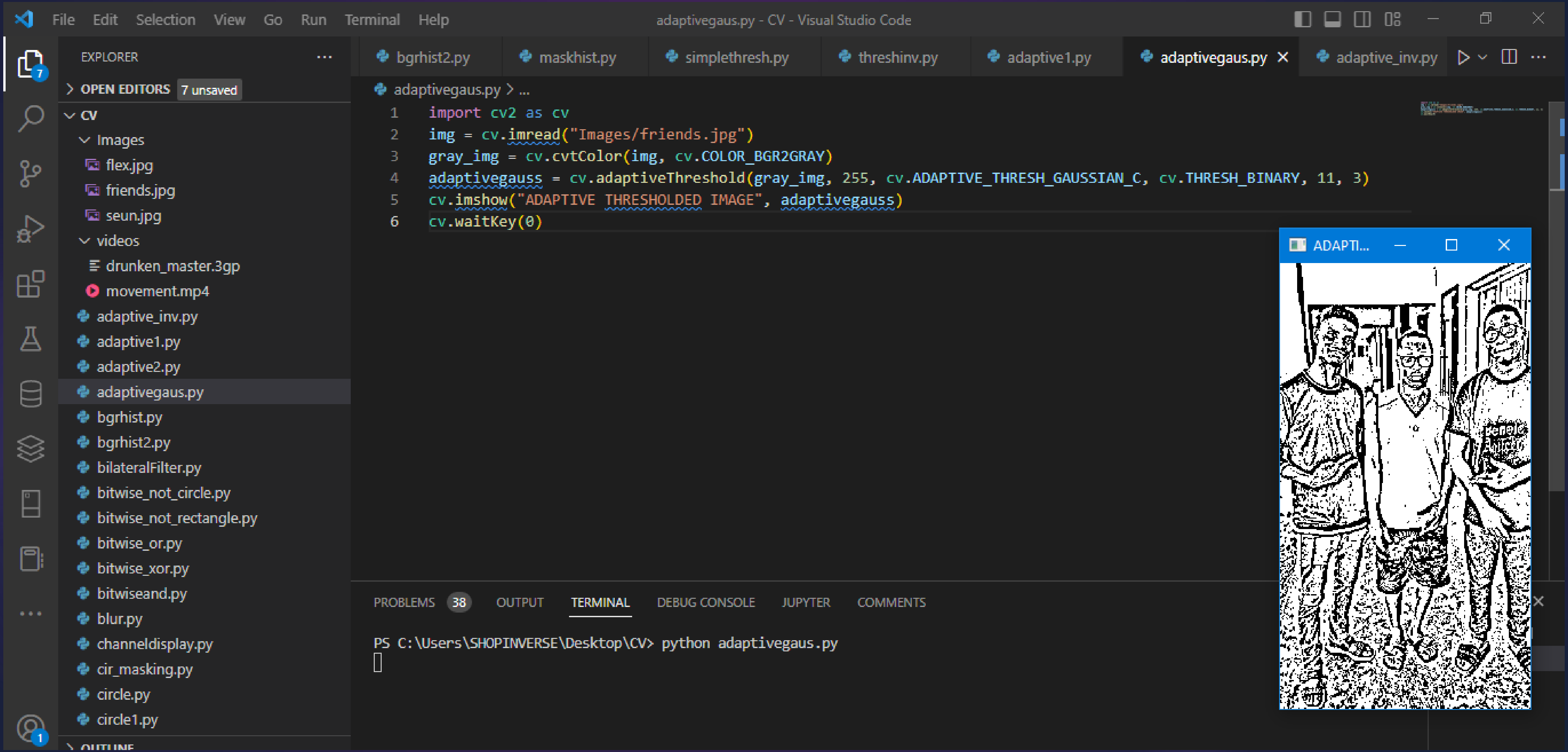
```
PS C:\Users\SHOPINVERSE\Desktop\CV> python adaptive_inv.py
```

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



```python
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)
```

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

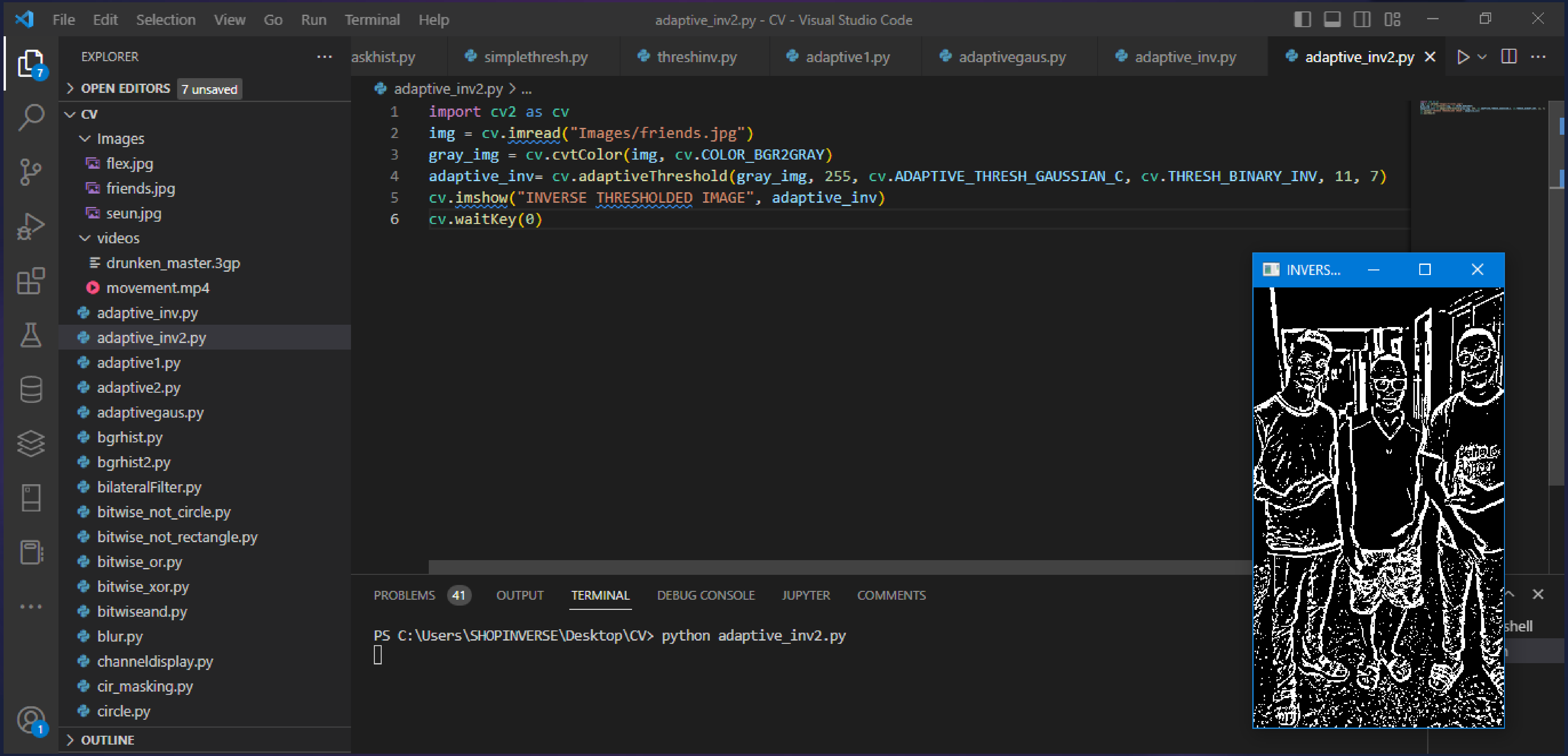
THRESHOLDING IN OpenCV Cont’d

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



```python
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)
```

```
PS C:\Users\SHOPINVERSE\Desktop\CV> python adaptive_inv2.py
```

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))
```

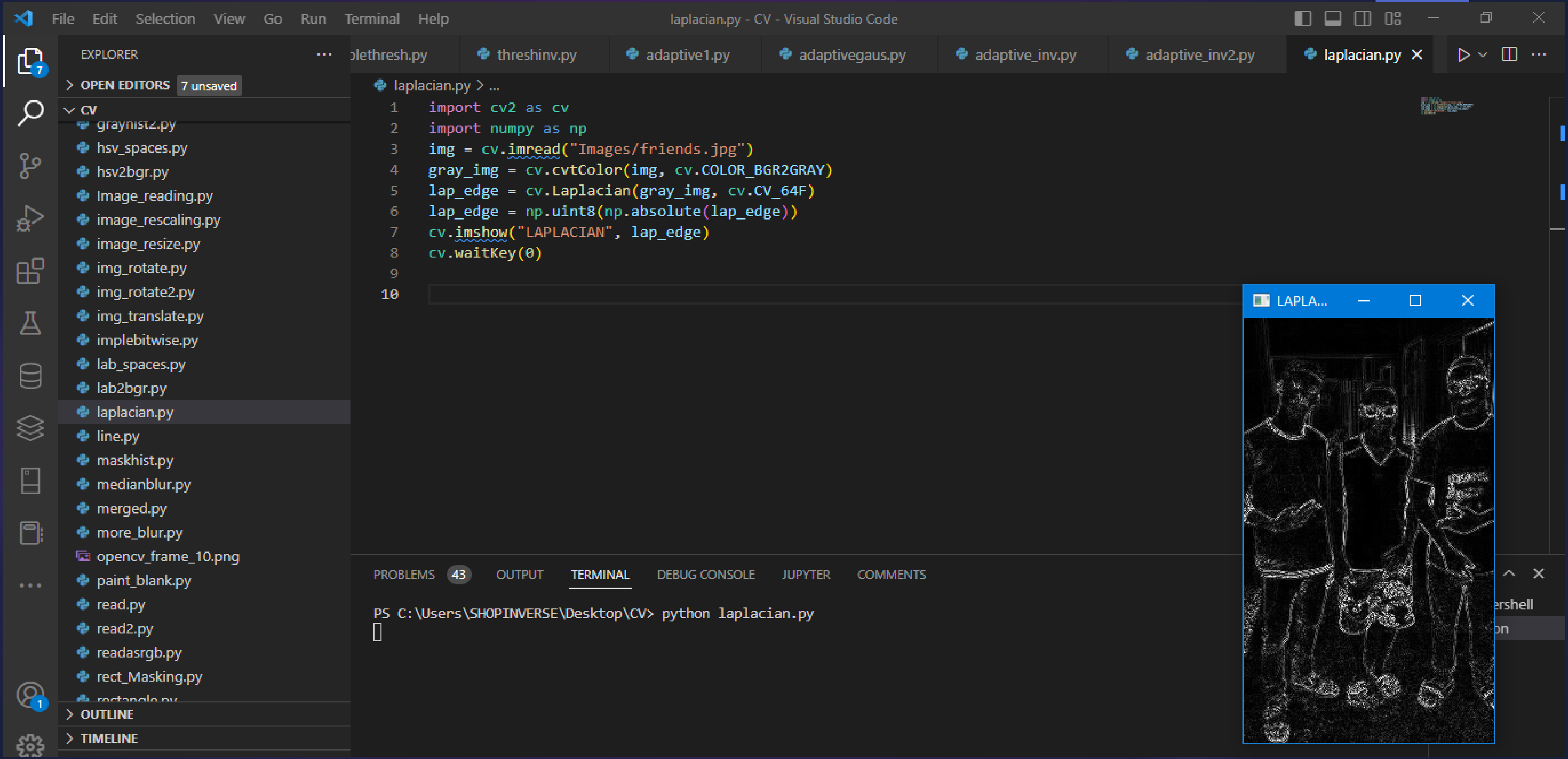
EDGE DETECTION Cont’d

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)
```

OUPUT



```python
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)
```

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

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, ddept, 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)
```

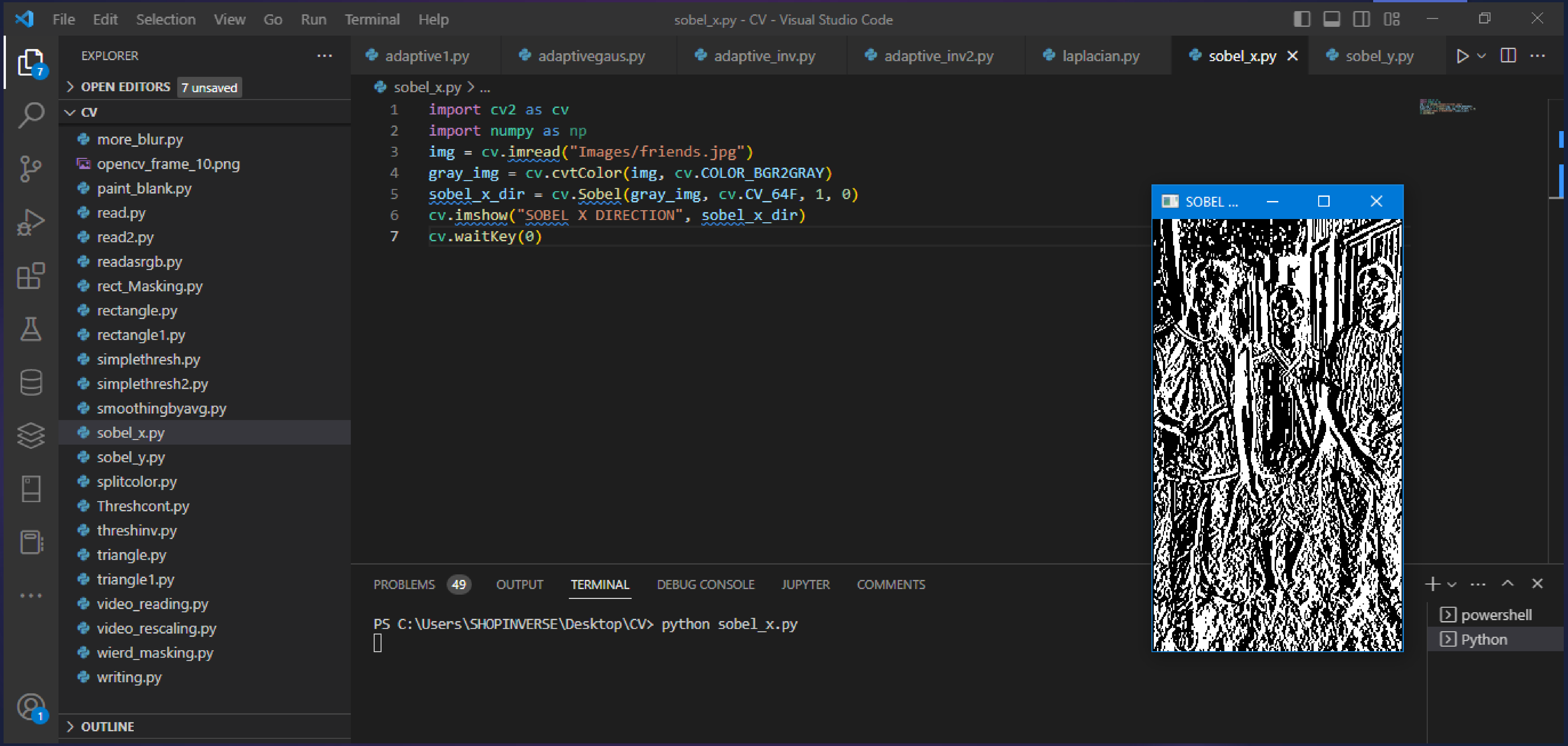
(ii)

```
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)
```

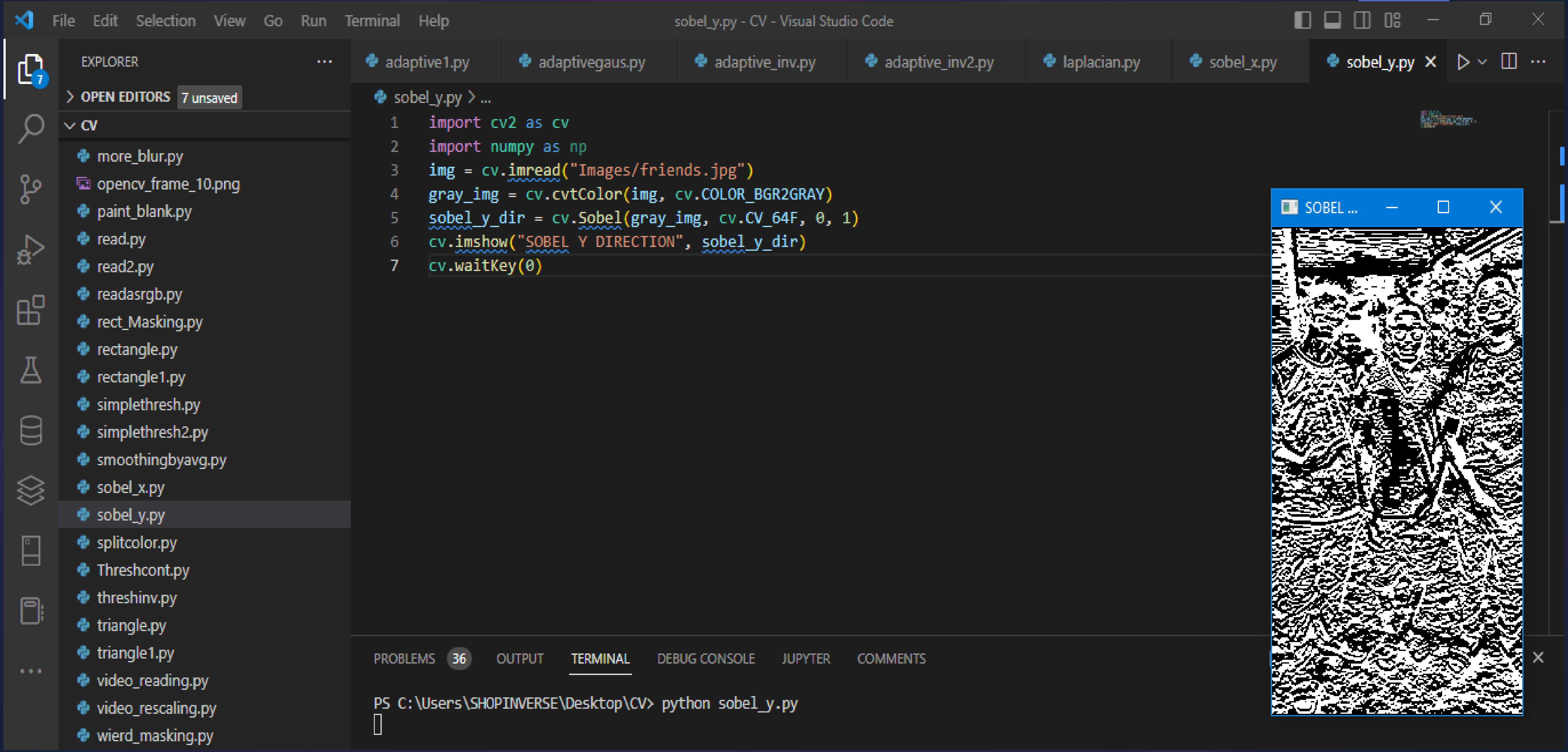
OUTPUT (i)



```python
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)
```

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

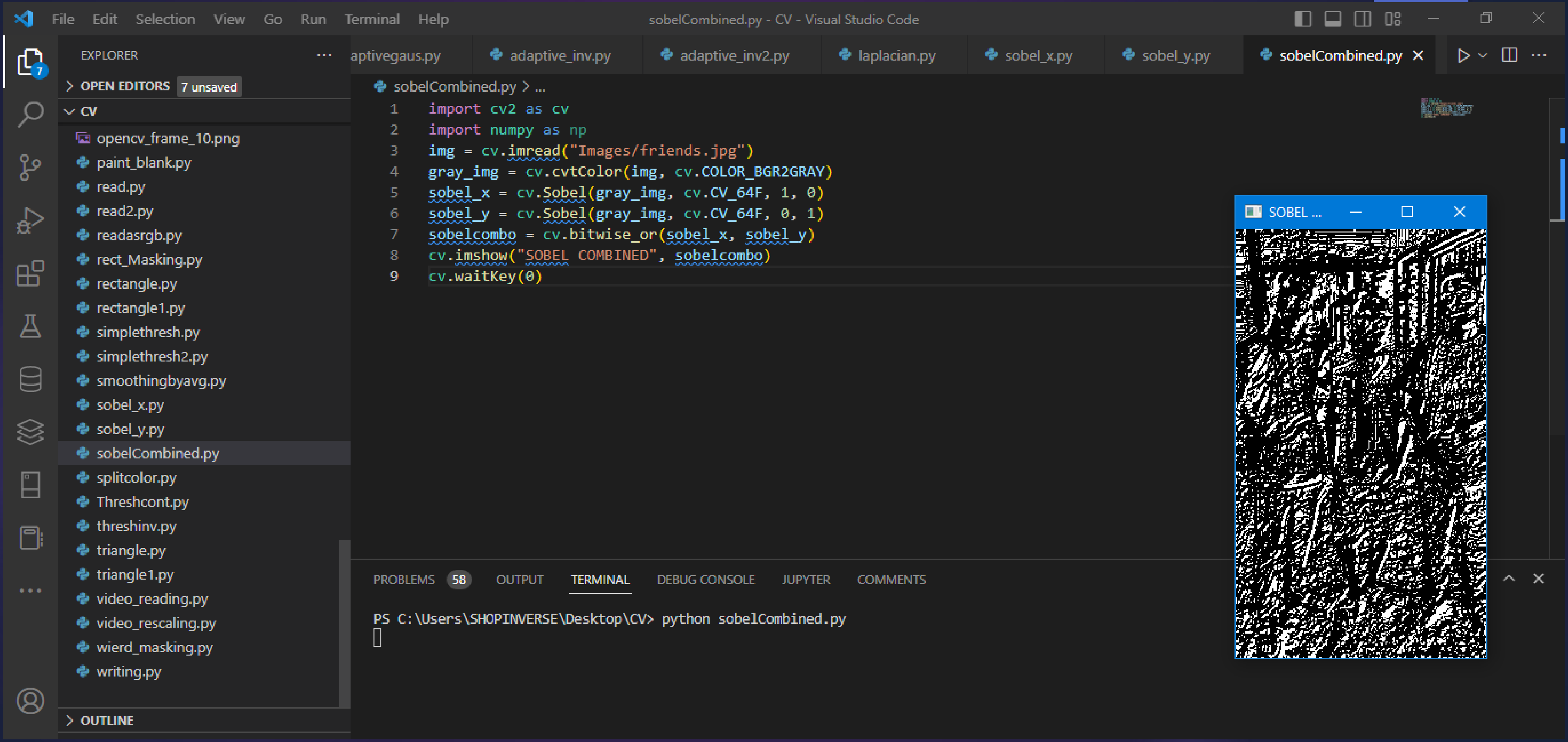
OUTPUT (ii)



```python
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)
```

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

OUTPUT (iii)



```python
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)
```

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

THANKS FOR VIEWING