



ORMEDIAN RESEARCH INSTITUTE

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

By:

ADEKOLA OLUWASEUN O.

February 25, 2023

TABLE OF CONTENT

- ❑ WHAT BITWISE OPERATOR IS & HOW IT WORKS
- ❑ BITWISE AND CONCEPT AND ITS IMPLEMENTATION
- ❑ BITWISE OR CONCEPT AND ITS IMPLEMENTATION
- ❑ BITWISE XOR CONCEPT AND ITS IMPLEMENTATION
- ❑ BITWISE NOT CONCEPT AND ITS IMPLEMENTATION
- ❑ MASKING IN OpenCV and HOW IT WORKS
- ❑ MASKING WITH DIFFERENT SHAPES

BITWISE OPERATORS IN OpenCV

Bitwise operators are like several other mathematical operators but are used for image manipulation purposes especially where there is a need to extract some specific regions of interest from images by using masks. Bitwise operators essentially help in image masking and in the enhancement of the properties of an image. When implementing bitwise operators, it is important to note that bitwise operators should be applied to images of the same dimensions. Bitwise operator in a binary manner by simply turning off a pixel if it has a binary value of 0, and then turn on a pixel if it has a binary value of 1.

There are basically 4 types of bitwise operators

- ❑ AND
- ❑ OR
- ❑ EXOR
- ❑ NOT

BITWISE OPERATORS IN OpenCV

Cont'd

BITWISE OPERATOR SYNTAX

Implementation of different bitwise operators requires a well-structured syntax with useful arguments.

The Syntax

```
cv2.bitwise_operator(src1, src2, dest, mask)
```

Where,

src1: The first input image

src2: The second input image

dest: Output array of similar dimensions as the input images

mask: The filter to perform operations directly on the input images.

BITWISE OPERATORS IN OpenCV

Cont'd

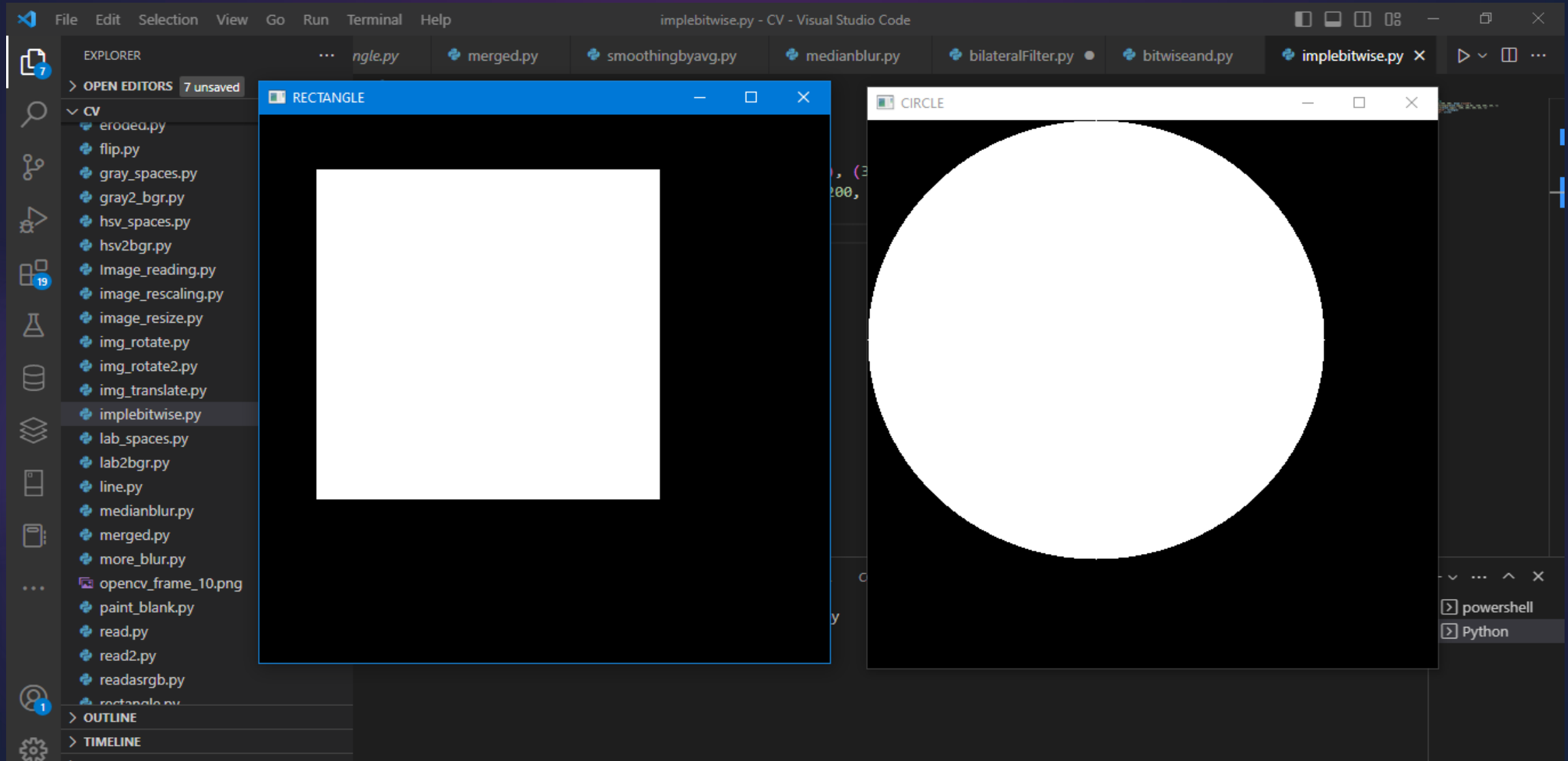
In this presentation, we will focus on two different shapes for the implementation of all the types of bitwise operators that exist.

I will be writing a simple algorithm to develop the shapes here.

```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
cv.imshow("RECTANGLE", rectangle)
cv.imshow("CIRCLE", circle)
cv.waitKey(0)
```

BITWISE OPERATORS IN OpenCV

Cont'd



BITWISE OPERATORS IN OpenCV

Cont'd

Bitwise AND

This is used to perform logical conjunction on images in OpenCV. It returns only the intersecting regions.

Bitwise AND is only true if and only if both pixels are greater than zero.

```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_and = cv.bitwise_and(rectangle, circle)
cv.imshow("Bitwise AND", bitwise_and)
cv.waitKey(0)
```

OUTPUT

The image shows a Visual Studio Code editor window with the file `bitwiseand.py` open. The code defines a 500x500 pixel blank image, draws a white rectangle (50, 50 to 350, 350) and a white circle (200, 200 to 255, 255) on a black background, and then performs a bitwise AND operation between the blank image and the combined shapes. The result is displayed in a window titled "Bitwise AND".

```
1 import cv2 as cv
2 import numpy as np
3 blank = np.zeros((500, 500), dtype="uint8")
4 rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
5 circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
6 bitwise_and = cv.bitwise_and(rectangle, circle)
7 cv.imshow("Bitwise AND", bitwise_and)
8 cv.waitKey(0)
```

The terminal output shows the command `python bitwiseand.py` being executed in a PowerShell shell at the path `C:\Users\SHOPINVERSE\Desktop\CV>`.

The "Bitwise AND" window displays a black image with a white rectangle and a white circle, representing the result of the bitwise AND operation between the blank image and the combined shapes.

BITWISE OPERATORS IN OpenCV

Cont'd

Bitwise OR

It returns both the intersecting and non-intersecting regions. Bitwise OR is only true if either of the two pixels is greater than zero.

```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_and = cv.bitwise_and(rectangle, circle)
cv.imshow("Bitwise AND", bitwise_and)
cv.waitKey(0)
```

OUTPUT

The image shows a Visual Studio Code interface with a Python script named `bitwise_or.py` open. The script performs a bitwise OR operation on a blank image and a circle. The output is displayed in a separate window titled "Bitwise OR".

EXPLORER

- OPEN EDITORS 7 unsaved
- CV
 - Images
 - friends.jpg
 - seun.jpg
 - videos
 - drunken_master.3gp
 - movement.mp4
 - bitlateralFilter.py
 - bitwise_or.py
 - bitwiseand.py
 - blur.py
 - channeldisplay.py
 - circle.py
 - circle1.py
 - contourdetect.py
 - contreduced.py
 - conversion2gray.py
 - convrgb2bgr.py
 - crop_image.py
 - cropped_image.py
 - dilated.py
 - draw_blank.py
 - drawContours.py
 - edge_cascade.py
 - ellipse.py
- OUTLINE
- TIMELINE
- MYSQL

bitwise_or.py

```
1 import cv2 as cv
2 import numpy as np
3 blank = np.zeros((500, 500), dtype="uint8")
4 rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
5 circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
6 bitwise_or = cv.bitwise_or(rectangle, circle)
7 cv.imshow("Bitwise OR", bitwise_or)
8 cv.waitKey(0)
```

TERMINAL

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

Bitwise OR

The output window shows a black image with a white circle and a white rectangle, representing the result of the bitwise OR operation.

BITWISE OPERATORS IN OpenCV

Cont'd

Bitwise XOR

It returns non-intersecting regions. Bitwise XOR is only true if and only if one of the two pixels is greater than zero, but not both.

```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_xor = cv.bitwise_xor(rectangle, circle)
cv.imshow("Bitwise XOR", bitwise_xor)
cv.waitKey(0)
```

OUTPUT

The image shows a Visual Studio Code editor window with the file `bitwise_xor.py` open. The Explorer sidebar on the left shows a project structure with folders `CV` and `Images`, and various Python files. The main editor area displays the code for `bitwise_xor.py`, which uses OpenCV to create a white circle and a black square, then performs a bitwise XOR operation on them. The output is shown in a separate window titled "Bitwise XOR", displaying a white circle with a black square inside it. The bottom of the editor shows the Terminal with the command `python bitwise_xor.py` executed successfully.

```
1 import cv2 as cv
2 import numpy as np
3 blank = np.zeros((500, 500), dtype="uint8")
4 rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
5 circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
6 bitwise_xor = cv.bitwise_xor(rectangle, circle)
7 cv.imshow("Bitwise XOR", bitwise_xor)
8 cv.waitKey(0)
```

PROBLEMS 36 OUTPUT TERMINAL DEBUG CONSOLE JUPYTER COMMENTS

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

BITWISE OPERATORS IN OpenCV

Cont'd

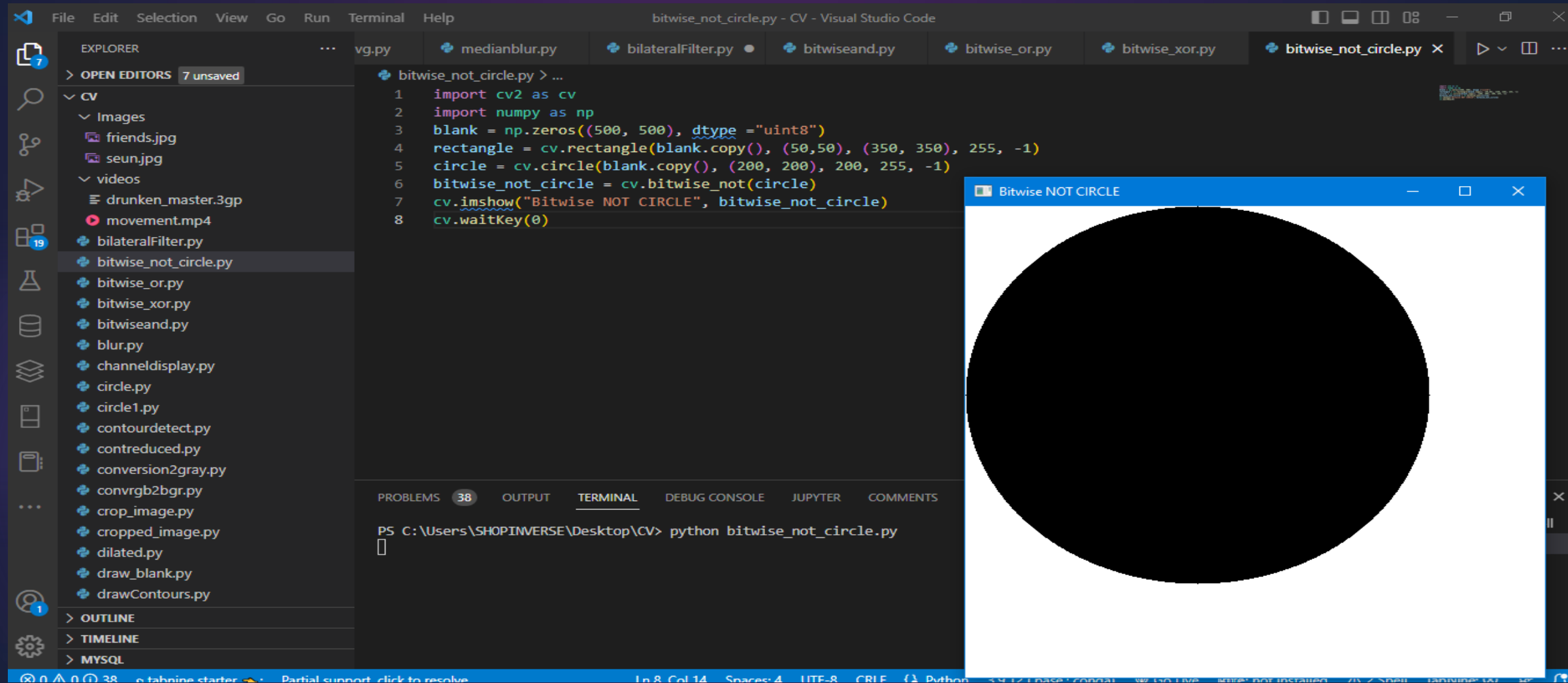
Bitwise NOT

Unlike other bitwise operators, bitwise NOT operator takes in only one image and then inverts the binary colour of such an input image. It takes in only one input image. In the examples shown below, let us consider finding the bitwise NOT operator of the circle and rectangle, respectively.

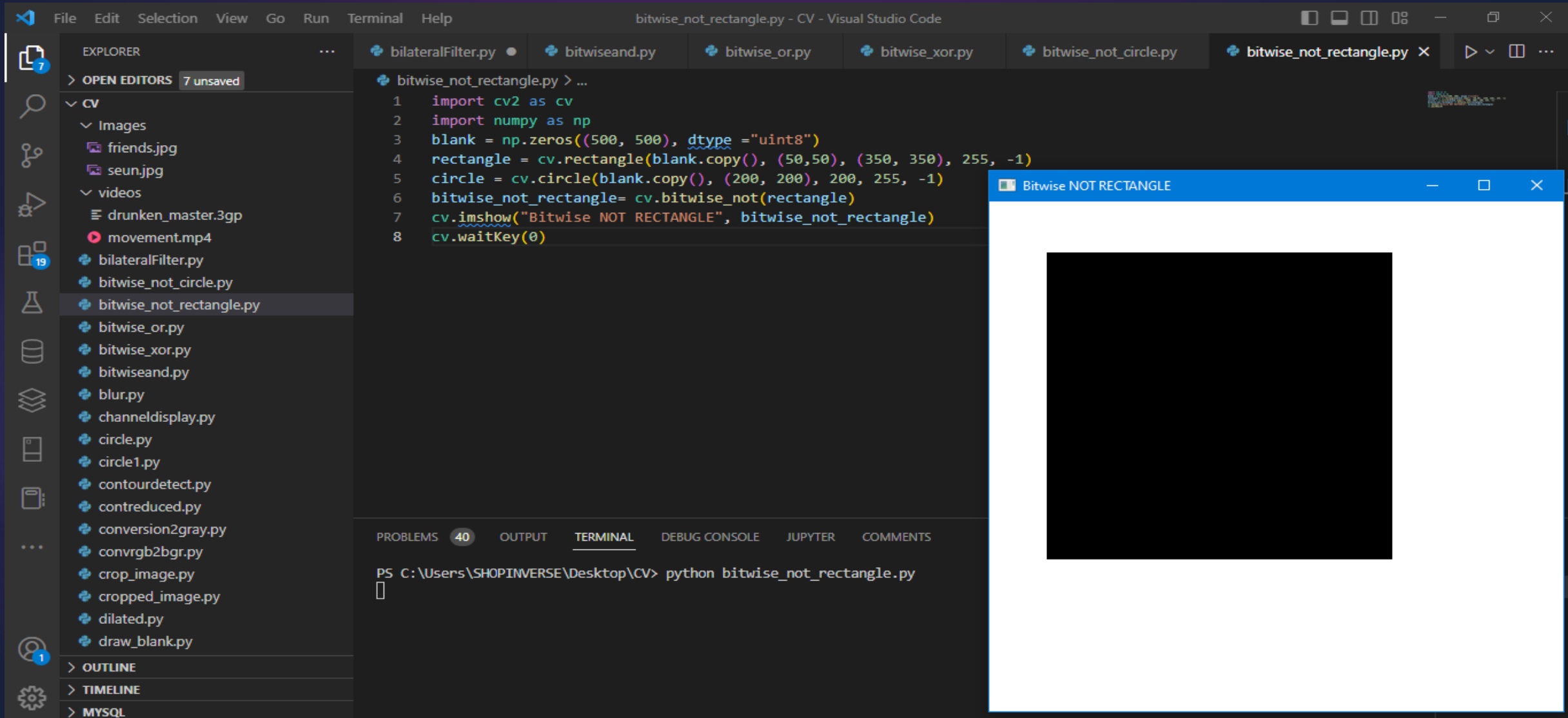
```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_not_circle = cv.bitwise_not(circle)
cv.imshow("Bitwise NOT CIRCLE", bitwise_not_circle)
cv.waitKey(0)
```

```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_not_rectangle= cv.bitwise_not(rectangle)
cv.imshow("Bitwise NOT RECTANGLE", bitwise_not_rectangle)
cv.waitKey(0)
```

OUTPUT (bitwise NOT circle)



OUTPUT (bitwise NOT Rectangle)



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

- EXPLORER:** Shows a project named 'CV' containing images (friends.jpg, seun.jpg), videos (drunken_master.3gp, movement.mp4), and Python scripts (bilateralFilter.py, bitwise_not_circle.py, bitwise_not_rectangle.py, bitwise_or.py, bitwise_xor.py, bitwiseand.py, blur.py, channeldisplay.py, circle.py, circle1.py, contourdetect.py, contreduced.py, conversion2gray.py, convrgb2bgr.py, crop_image.py, cropped_image.py, dilated.py, draw_blank.py).
- EDITOR:** Displays the code for `bitwise_not_rectangle.py`:

```
1 import cv2 as cv
2 import numpy as np
3 blank = np.zeros((500, 500), dtype="uint8")
4 rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
5 circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
6 bitwise_not_rectangle= cv.bitwise_not(rectangle)
7 cv.imshow("Bitwise NOT RECTANGLE", bitwise_not_rectangle)
8 cv.waitKey(0)
```
- TERMINAL:** Shows the command `PS C:\Users\SHOPINVERSE\Desktop\CV> python bitwise_not_rectangle.py` being executed.
- Bitwise NOT RECTANGLE Window:** A separate window displaying the output image, which is a black rectangle on a white background.

MASKING IN OpenCV

MASKING

Masking in OpenCV is defined as an image processing technique used to output the Region of Interest (RoI).

Masking allows us to focus on certain parts of an image. i.e., we can choose a particular portion of an image by simply removing the unwanted parts. And before masking could be successfully applied, the dimensions of the mask must be the same as of the input image.

SYNTAX

```
masked = cv2.bitwise_and(img, img, mask=mask)
```


MASKING IN OpenCV

Cont'd

Example 1: Mask over any part of the image “friends.jpg” by using:

- (i) Circular mask
- (ii) Rectangular mask
- (iii) Elliptical mask
- (iv) Any other weird mask

MASKING IN OpenCV

Cont'd

SOLUTION

(i)

```
import cv2 as cv
import numpy as np
img = cv.imread("Images/friends.jpg")
cv.imshow("Original Image", img)
blank = np.zeros(img.shape[:2], dtype = "uint8")
circular_mask = cv.circle(blank.copy(), (img.shape[1]//2, img.shape[0]//2), 70, 255, -1)
cv.imshow("CIRCULAR MASK", circular_mask)
masked = cv.bitwise_and(img, img, mask=circular_mask)
cv.imshow("MASKED IMAGE", masked)
cv.waitKey(0)
```

(ii)

```
import cv2 as cv
import numpy as np
img = cv.imread("Images/friends.jpg")
cv.imshow("Original Image", img)
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)
cv.imshow("MASKED IMAGE", masked)
cv.waitKey(0)
```

(iii)

```
import cv2 as cv
import numpy as np
img = cv.imread("Images/friends.jpg")
cv.imshow("Original Image", img)
blank = np.zeros(img.shape[:2], dtype = "uint8")
elliptical_mask = cv.ellipse(blank.copy(), (100, 100), (60, 120), 120, 0, 360, 255, -1)
cv.imshow("ELLIPTICAL MASK", elliptical_mask)
masked = cv.bitwise_and(img, img, mask=elliptical_mask)
cv.imshow("MASKED IMAGE", masked)
cv.waitKey(0)
```

(iv)

```
import cv2 as cv
import numpy as np
img = cv.imread("Images/friends.jpg")
cv.imshow("Original Image", img)
blank = np.zeros(img.shape[:2], dtype = "uint8")
elliptical_mask = cv.ellipse(blank.copy(), (60, 60), (60, 120), 120, 0, 360, 255, -1)
rectangular_mask = cv.rectangle(blank.copy(), (60, 60), (250, 250), 255, -1)
combined_mask = cv.bitwise_and(elliptical_mask, rectangular_mask)
cv.imshow("COMBINED MASK", combined_mask)
masked = cv.bitwise_and(img, img, mask=combined_mask)
cv.imshow("MASKED IMAGE", masked)
cv.waitKey(0)
```

OUTPUT

Visual Studio Code interface showing the execution of a Python script named `Masking.py` using OpenCV.

EXPLORER (Left Panel):

- CV
 - nsf_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
 - line.py
 - Masking.py**
 - medianblur.py
 - merged.py
 - more_blur.py
 - opencv_frame_10.png
 - paint_blank.py
 - read.py
 - read2.py
 - readasrgb.py
 - rectangle.py
 - rectangle1.py
 - smoothingbyavg.py
 - splitcolor.py
- OUTLINE
- TIMELINE

Masking.py (Editor):

```
1 import cv2 as cv
2 import numpy as np
3 img = cv.imread("Images/friends.jpg")
4 cv.imshow("Original Image", img)
5 blank = np.zeros(img.shape[:2], dtype="uint8")
6 circular_mask = cv.circle(blank.copy(), (img.shape[1]//2, img.shape[0]//2), 70, 255, -1)
7 cv.imshow("CIRCULAR MASK", circular_mask)
8 masked = cv.bitwise_or(img, img, mask=circular_mask)
9 cv.imshow("MASKED IMAGE", masked)
10 cv.waitKey(0)
11
```

Terminal (Bottom Panel):

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

Output Windows:

- Original...**: Displays the original image of three people standing in a hallway.
- CIRCULAR...**: Displays a white circular mask on a black background.
- MASKE...**: Displays the masked image, showing only the portion of the original image covered by the circular mask.

OUTPUT

Visual Studio Code interface showing the execution of a Python script for image masking.

EXPLORER (6 unsaved):

- CV
 - nsf_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
 - line.py
 - Masking.py
 - medianblur.py
 - merged.py
 - more_blur.py
 - opencv_frame_10.png
 - paint_blank.py
 - read.py
 - read2.py
 - readasrgb.py
 - rectangle.py
 - rectangle1.py
 - smoothingbyavg.py
 - colitecolor.py
- OUTLINE
- TIMELINE
- MYSQL

Masking.py (selected):

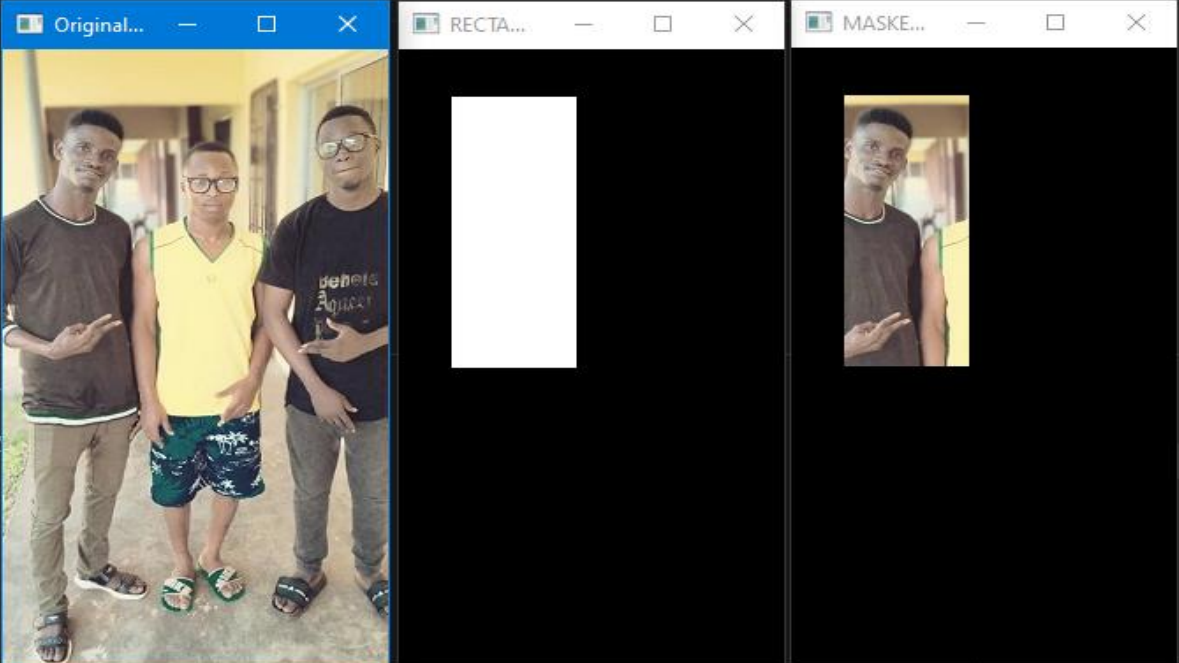
```
1 import cv2 as cv
2 import numpy as np
3 img = cv.imread("Images/friends.jpg")
4 cv.imshow("Original Image", img)
5 blank = np.zeros(img.shape[:2], dtype = "uint8")
6 rectangular_mask = cv.rectangle(blank.copy(), (30, 30), (100, 200), 255, -1)
7 cv.imshow("RECTANGULAR MASK", rectangular_mask)
8 masked = cv.bitwise_and(img, img, mask=rectangular_mask)
9 cv.imshow("MASKED IMAGE", masked)
10 cv.waitKey(0)
11
```

Terminal (24 problems):

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

Output Windows:

- Original... (Original Image)
- RECTA... (RECTANGULAR MASK)
- MASKE... (MASKED IMAGE)



OUTPUT

Visual Studio Code interface showing the execution of a Python script for image masking.

EXPLORER

- CV
 - nsf_spaces.py
 - hsv2bgr.py
 - Image_reading.py
 - image_rescaling.py
 - image_resize.py
 - img_rotate.py
 - img_rotate2.py
 - img_translate.py
 - imblebitwise.py
 - lab_spaces.py
 - lab2bgr.py
 - line.py
 - Masking.py**
 - medianblur.py
 - merged.py
 - more_blur.py
 - opencv_frame_10.png
 - paint_blank.py
 - read.py
 - read2.py
 - readasrgb.py
 - rectangle.py
 - rectangle1.py
 - smoothingbyavg.py
 - splitcolor.py
- OUTLINE
- TIMELINE
- MYSQL

Masking.py

```
1 import cv2 as cv
2 import numpy as np
3 img = cv.imread("Images/friends.jpg")
4 cv.imshow("Original Image", img)
5 blank = np.zeros(img.shape[:2], dtype = "uint8")
6 elliptical_mask = cv.ellipse(blank.copy(), (100, 100), (60, 120), 120, 0, 360, 255, -1)
7 cv.imshow("ELLIPTICAL MASK", elliptical_mask)
8 masked = cv.bitwise_and(img, img, mask=elliptical_mask)
9 cv.imshow("MASKED IMAGE", masked)
10 cv.waitKey(0)
11
```

Terminal

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

Output Windows:

- Original...: Original image showing three people.
- ELLIPTI...: Elliptical mask (white ellipse on black background).
- MASKE...: Masked image (original image with the elliptical mask applied).

OUTPUT

Visual Studio Code interface showing the execution of a Python script for image masking.

EXPLORER

- CV
 - nsv_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
 - line.py
 - Masking.py**
 - medianblur.py
 - merged.py
 - more_blur.py
 - opencv_frame_10.png
 - paint_blank.py
 - read.py
 - read2.py
 - readasrgb.py
 - rectangle.py
 - rectangle1.py
 - smoothingbyavg.py
 - colitoler.py
- OUTLINE
- TIMELINE
- MYSQL

Masking.py

```
1 import cv2 as cv
2 import numpy as np
3 img = cv.imread("Images/friends.jpg")
4 cv.imshow("Original Image", img)
5 blank = np.zeros(img.shape[:2], dtype = "uint8")
6 elliptical_mask = cv.ellipse(blank.copy(), (60, 60), (60, 120), 120, 0, 360, 255, -1)
7 rectangular_mask = cv.rectangle(blank.copy(), (60, 60), (250, 250), 255, -1)
8 combined_mask = cv.bitwise_and(elliptical_mask, rectangular_mask)
9 cv.imshow("COMBINED MASK", combined_mask)
10 masked = cv.bitwise_and(img, img, mask=combined_mask)
11 cv.imshow("MASKED IMAGE", masked)
12 cv.waitKey(0)
13
```

Terminal

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

Output Windows:

- Original...: Original image showing three people.
- COMBI...: Combined mask (white shape on black background).
- MASKE...: Masked image (original image with the combined mask applied).



THANKS FOR VIEWING

More tutorials will be covered in part eight