ORMEDIAN RESEARCH INSTITUTE

TOPIC: : OpenCV Tutorials (part seven)

By:

ADEKOLA OLUWASEUN O.

February 25, 2023

TABLE OF CONTENT

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.

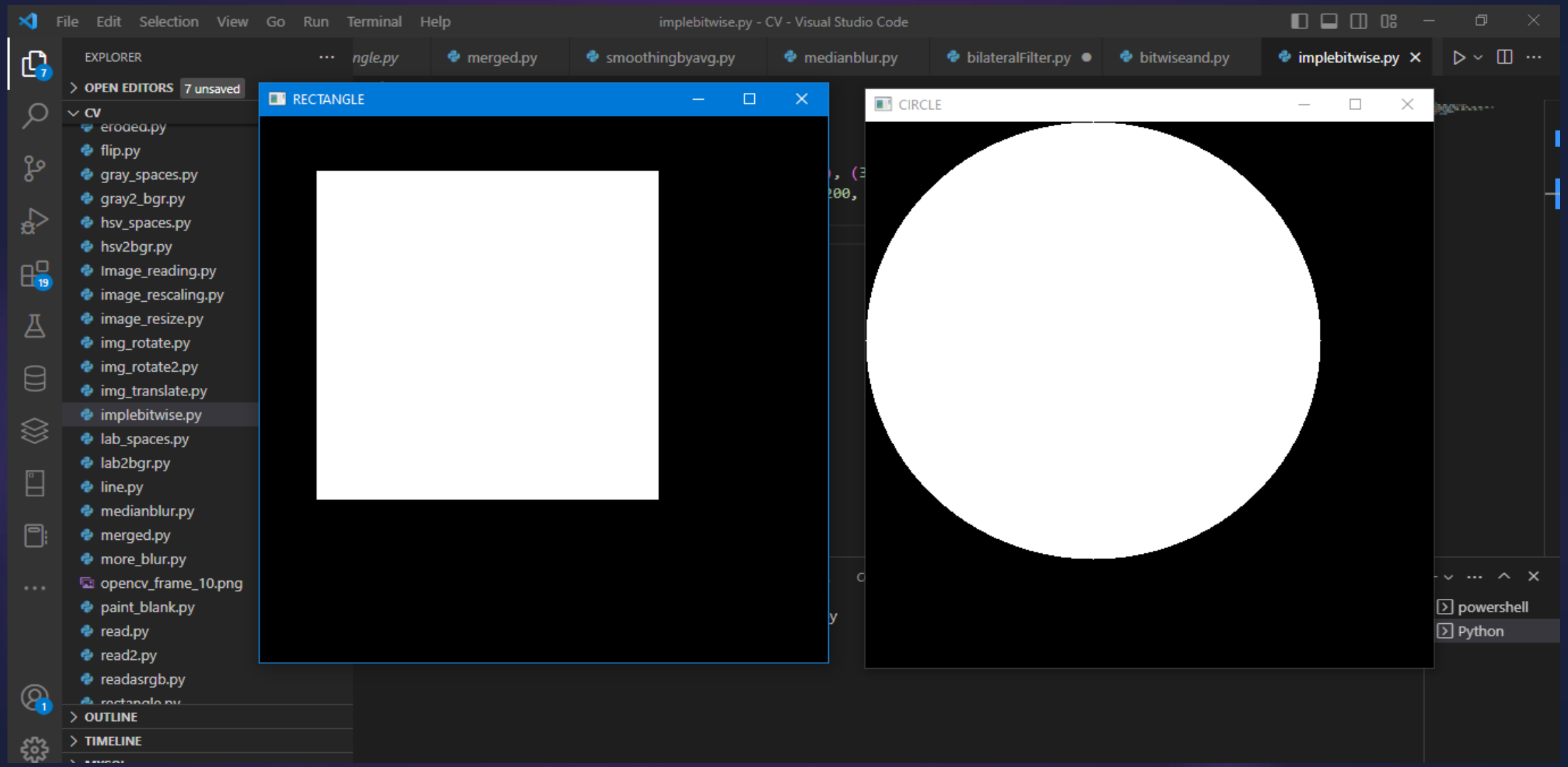
BITWISE OPERATORS IN OpenCV Cont'd

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.

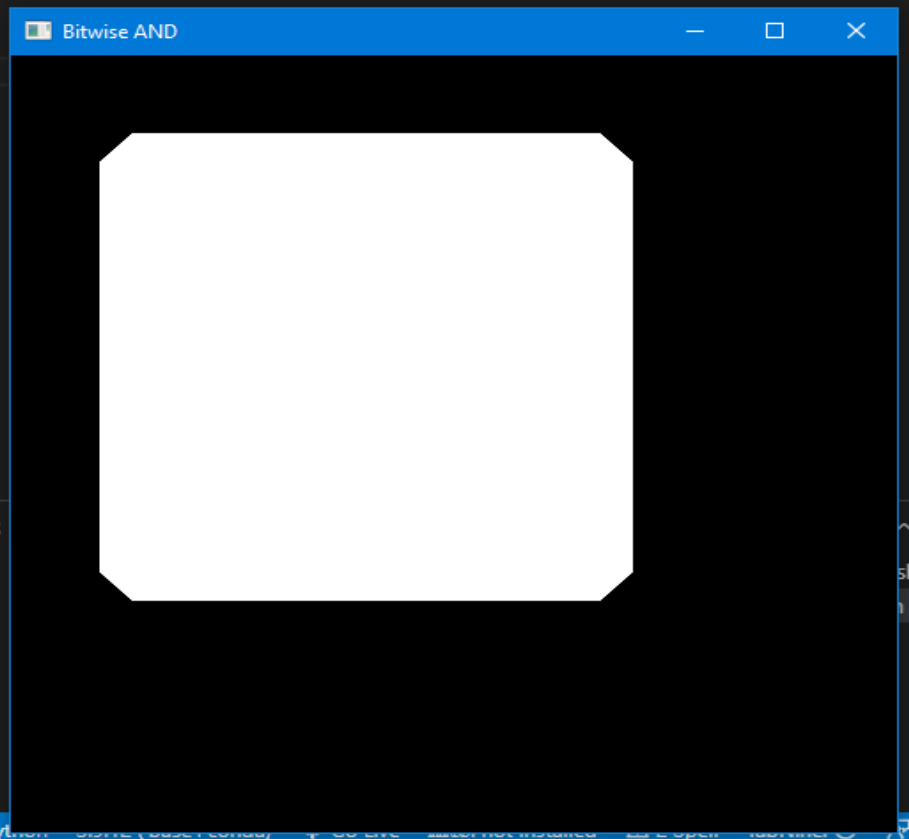
```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype ="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_and = cv.bitwise_and(rectangle, circle)
cv.imshow("Bitwise AND", bitwise_and)
cv.waitKey(0)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

OUTPUT

bitwiseand.py - CV - Visual Studio Code

```python
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)
```

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



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.

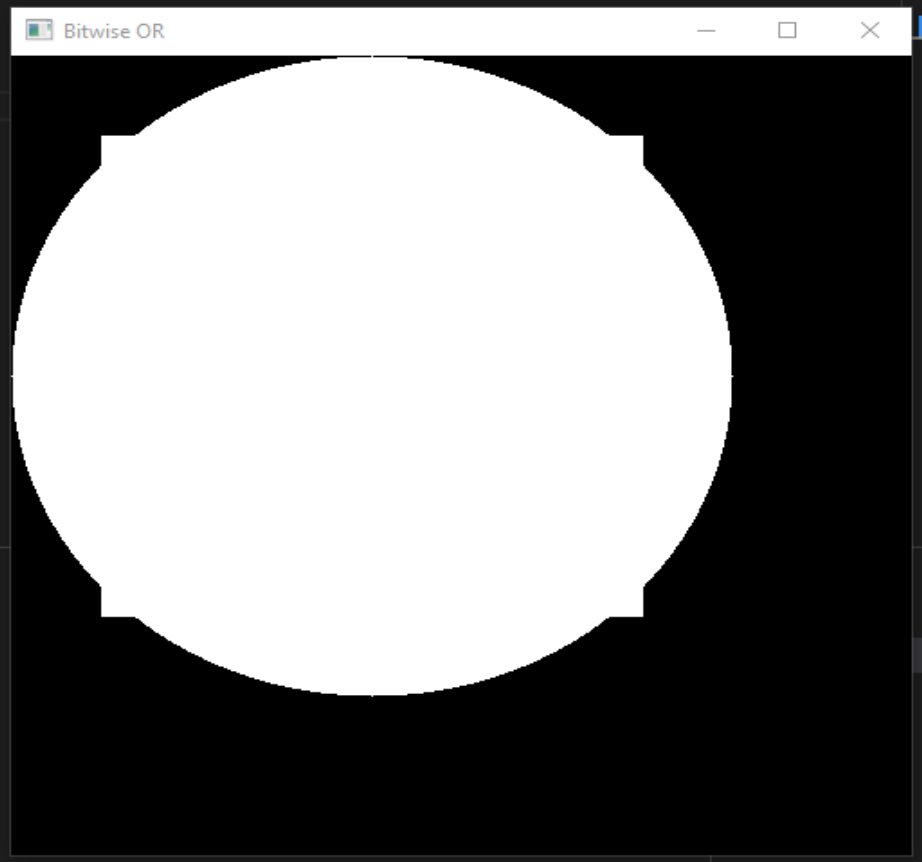
```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype ="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_and = cv.bitwise_and(rectangle, circle)
cv.imshow("Bitwise AND", bitwise_and)
cv.waitKey(0)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

OUTPUT

bitwise_or.py - CV - Visual Studio Code

```python
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_or = cv.bitwise_or(rectangle, circle)
cv.imshow("Bitwise OR", bitwise_or)
cv.waitKey(0)
```

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



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.

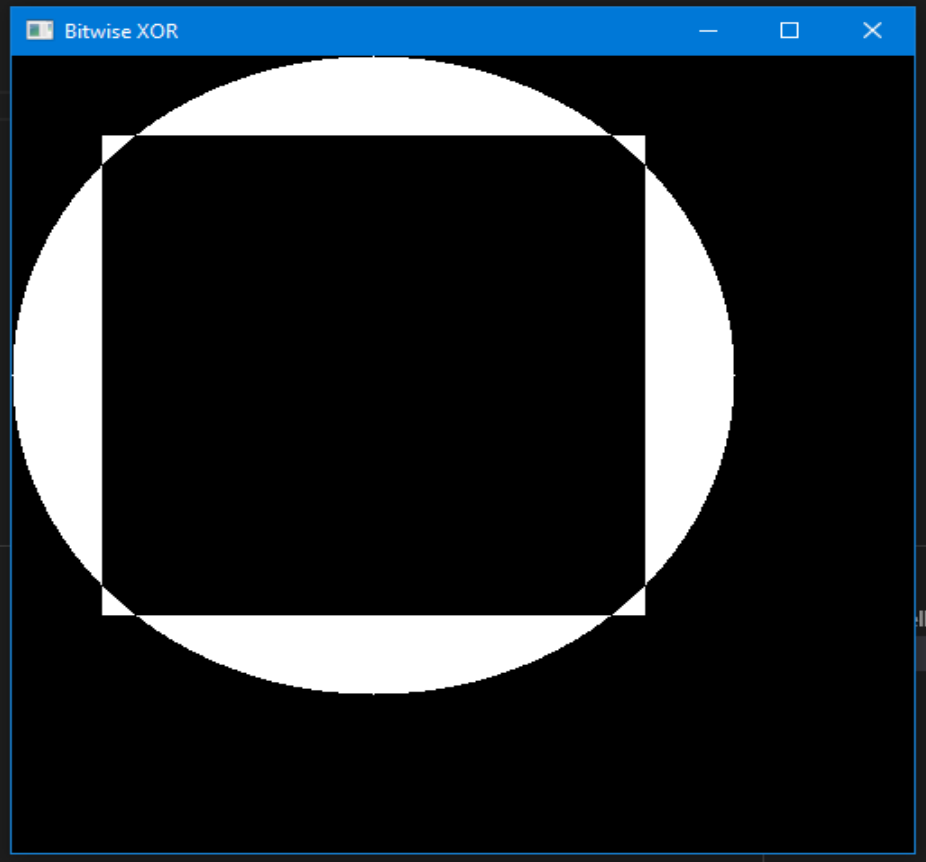
```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype ="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_xor = cv.bitwise_xor(rectangle, circle)
cv.imshow("Bitwise XOR", bitwise_xor)
cv.waitKey(0)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

OUTPUT

bitwise_xor.py - CV - Visual Studio Code

```python
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)
```

```
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)
```

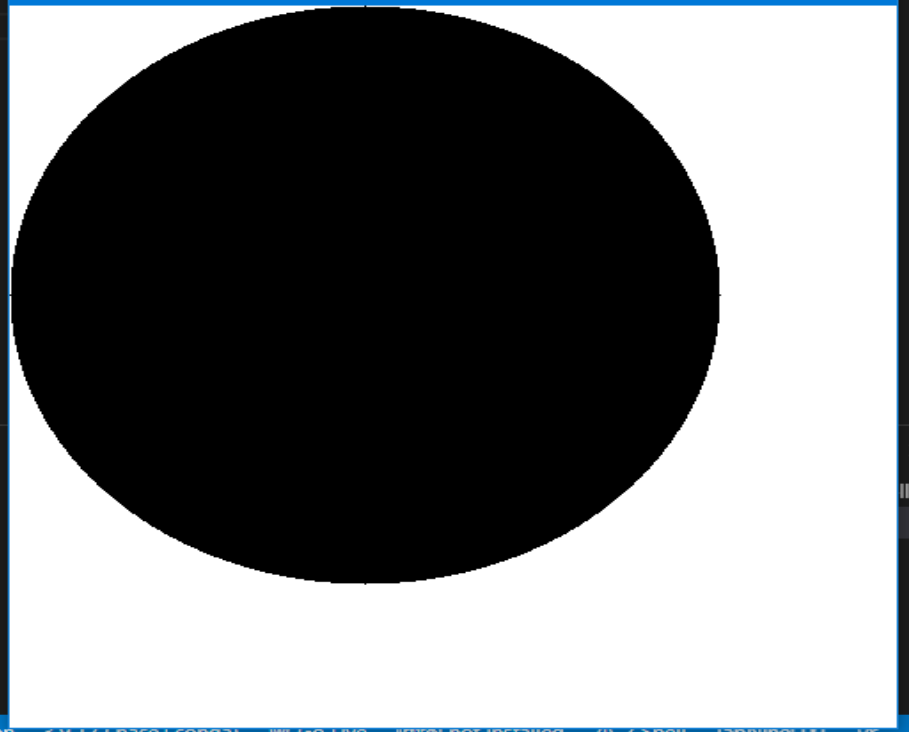
```
import cv2 as cv
import numpy as np
blank = np.zeros((500, 500), dtype ="uint8")
rectangle = cv.rectangle(blank.copy(), (50,50), (350, 350), 255, -1)
circle = cv.circle(blank.copy(), (200, 200), 200, 255, -1)
bitwise_not_rectangle= cv.bitwise_not(rectangle)
cv.imshow("Bitwise NOT RECTANGLE", bitwise_not_rectangle)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

OUTPUT (bitwise NOT circle)

bitwise_not_circle.py - CV - Visual Studio Code

```python
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)
```

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

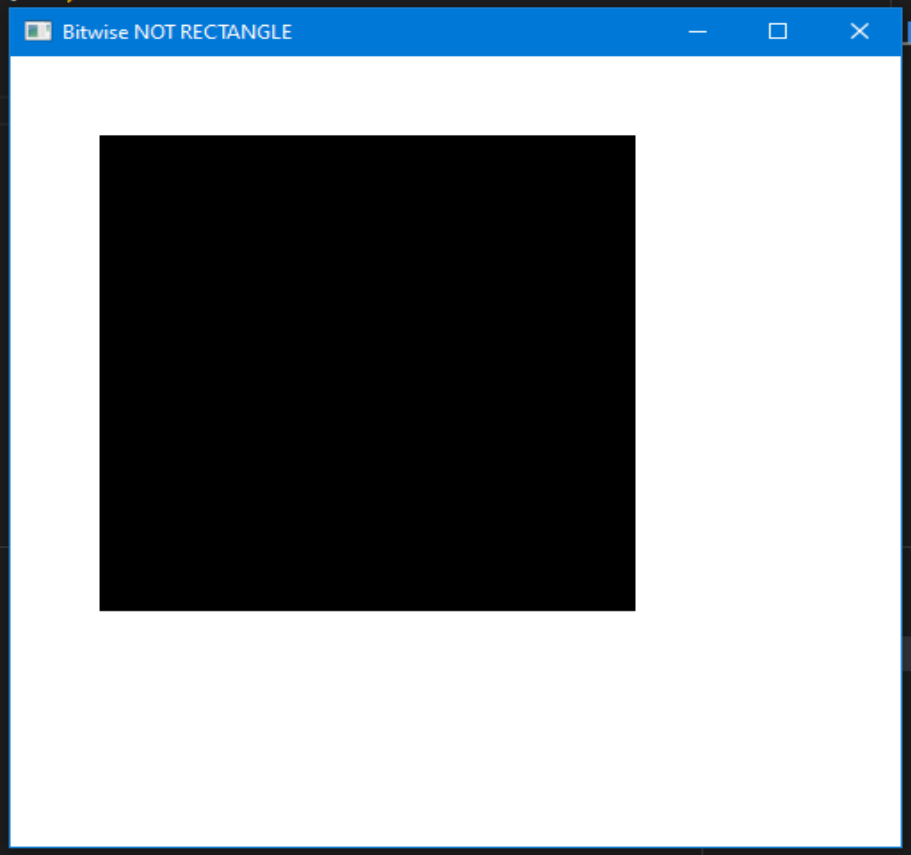


OUTPUT (bitwise NOT Rectangle)

bitwise_not_rectangle.py - CV - Visual Studio Code

```python
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)
```

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



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)
```

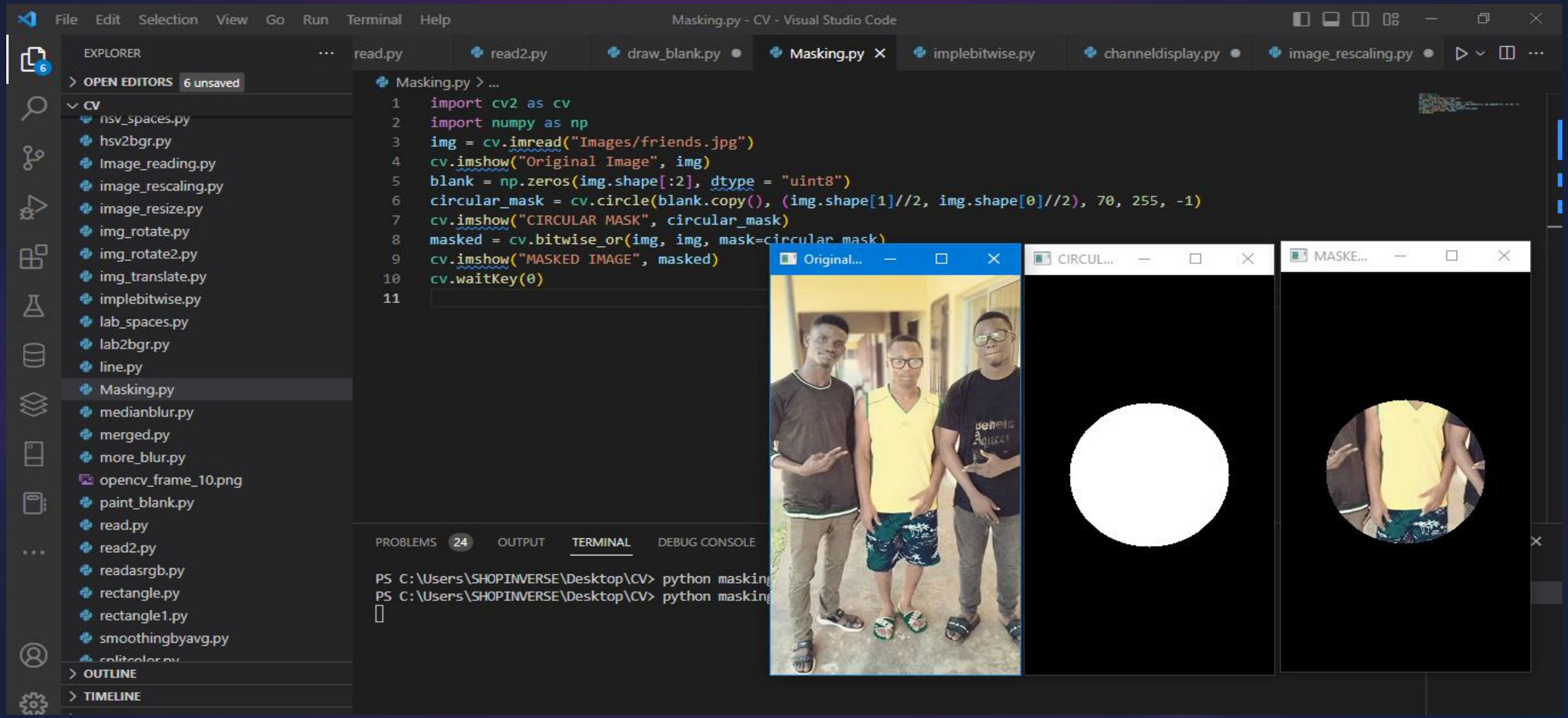
(iii)

```
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
iimg mport numpy as np
= 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

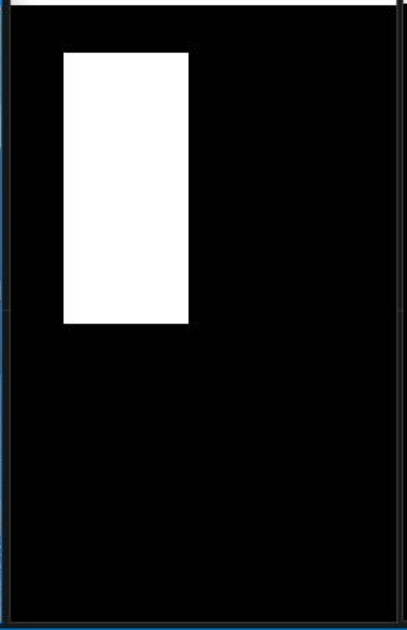

File Edit Selection View Go Run Terminal Help
Masking.py - CV - Visual Studio Code
EXPLORER
OPEN EDITORS 6 unsaved
CV
hsv_spaces.py
hsv2bgr.py
Image_reading.py
image_rescaling.py
image_resize.py
img_rotate.py
img_rotate2.py
img_translate.py
implebitwise.py
lab_spaces.py
lab2bgr.py
line.py
Masking.py
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
OUTLINE
TIMELINE
read.py
read2.py
draw_blank.py
Masking.py
implebitwise.py
channeldisplay.py
image_rescaling.py
Masking.py > ...
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_or(img, img, mask=circular_mask)
cv.imshow("MASKED IMAGE", masked)
cv.waitKey(0)
PROBLEMS 24
OUTPUT
TERMINAL
DEBUG CONSOLE
PS C:\Users\SHOPINVERSE\Desktop\CV> python maskin
PS C:\Users\SHOPINVERSE\Desktop\CV> python maskin
Original...
CIRCUL...
MASKE...

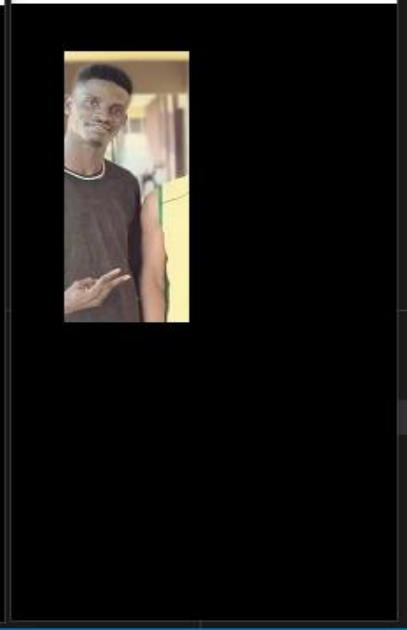
OUTPUT

Masking.py - CV - Visual Studio Code

```python
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)
```

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



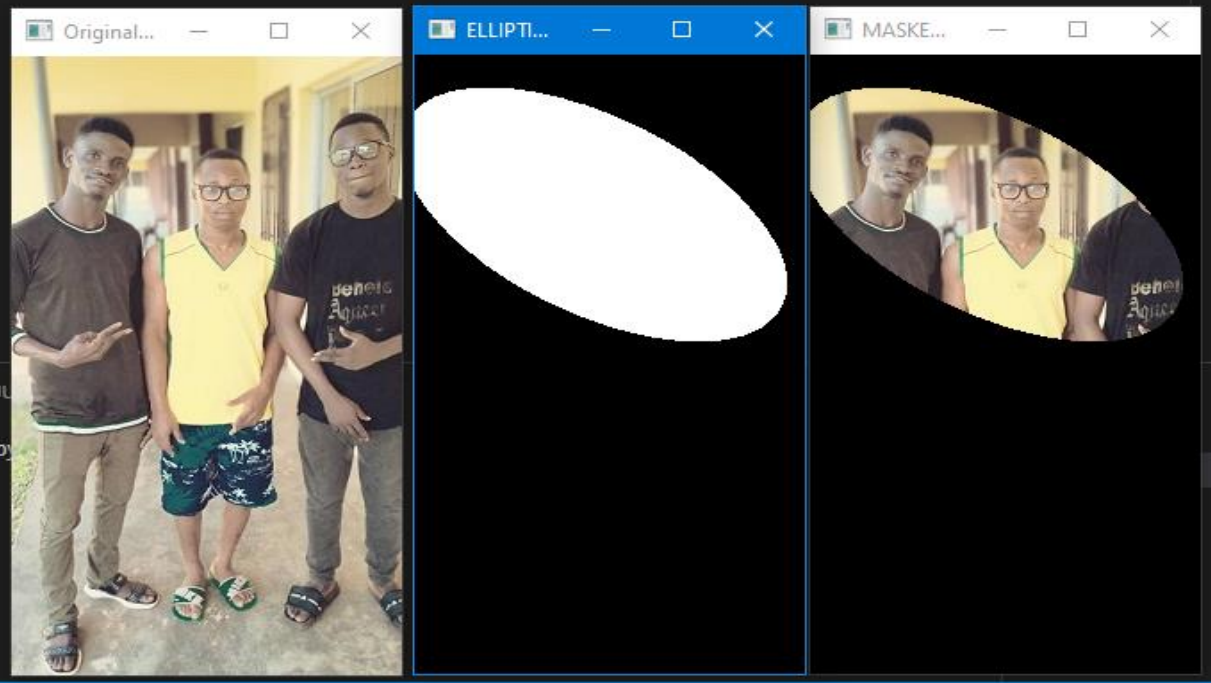


OUTPUT

Masking.py - CV - Visual Studio Code

```python
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)
```

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

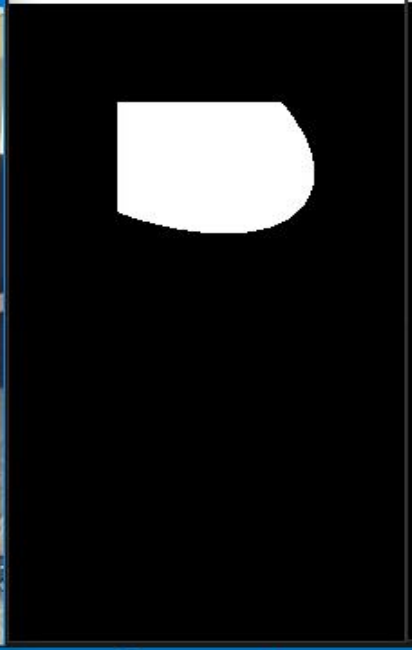


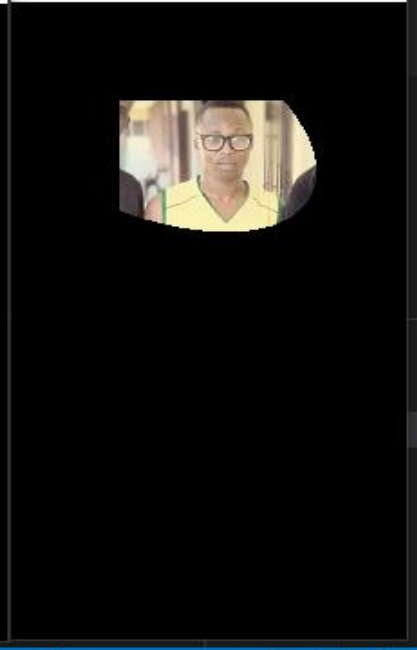
OUTPUT

Masking.py - CV - Visual Studio Code

```python
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)
```

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





THANKS FOR VIEWING

More tutorials will be covered in part eight