1.4.7: Bitwise operations

Topic Progress: (https://gurus.pyimagesearch.com/topic/translation/) (https://gurus.pyimagesearch.com/topic/rotation/) (https://gurus.pyimagesearch.com/topic/resizing/) (https://gurus.pyimagesearch.com/topic/flipping/) (https://gurus.pyimagesearch.com/topic/cropping/) (https://gurus.pyimagesearch.com/topic/image-arithmetic/) (https://gurus.pyimagesearch.com/topic/bitwise-operations/) (https://gurus.pyimagesearch.com/topic/masking/) (https://gurus.pyimagesearch.com/topic/splitting-and-merging-channels/)

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/basic-image-processing/)

Feedback

Remember back in **Module 1.4.5 (https://gurus.pyimagesearch.com/topic/cropping/)** when we discussed cropping and extracting the Region of Interest (ROI) from an image?

Well, in that particular example our ROI had to be *rectangular*.

However, what happens if our ROI is *non-rectangular?*

What would you do then?

As we'll find out in this Module, along with **Module 1.4.8 (https://gurus.pyimagesearch.com/topic/masking/),** a combination of bitwise operations and masking can help us extract non-rectangular ROIs from image with ease.

For now, we'll cover the basic bitwise operations — and in the next module, we'll learn how to utilize these bitwise operations to construct masks.

Objectives:

By the end of this topic you'll understand the four primary bitwise operations:

- AND
- OR
- XOR
- NOT

Bitwise Operations

In this section we will review four bitwise operations: AND, OR, XOR, and NOT. These four operations, while very basic and low level, are paramount to image processing — especially when we start working with masks in **Section 1.4.8** (https://gurus.pyimagesearch.com/topic/masking/).

Bitwise operations operate in a binary manner and are represented as grayscale images. A given pixel is turned "off" if it has a value of zero and it is turned "on" if the pixel has a value greater than zero.

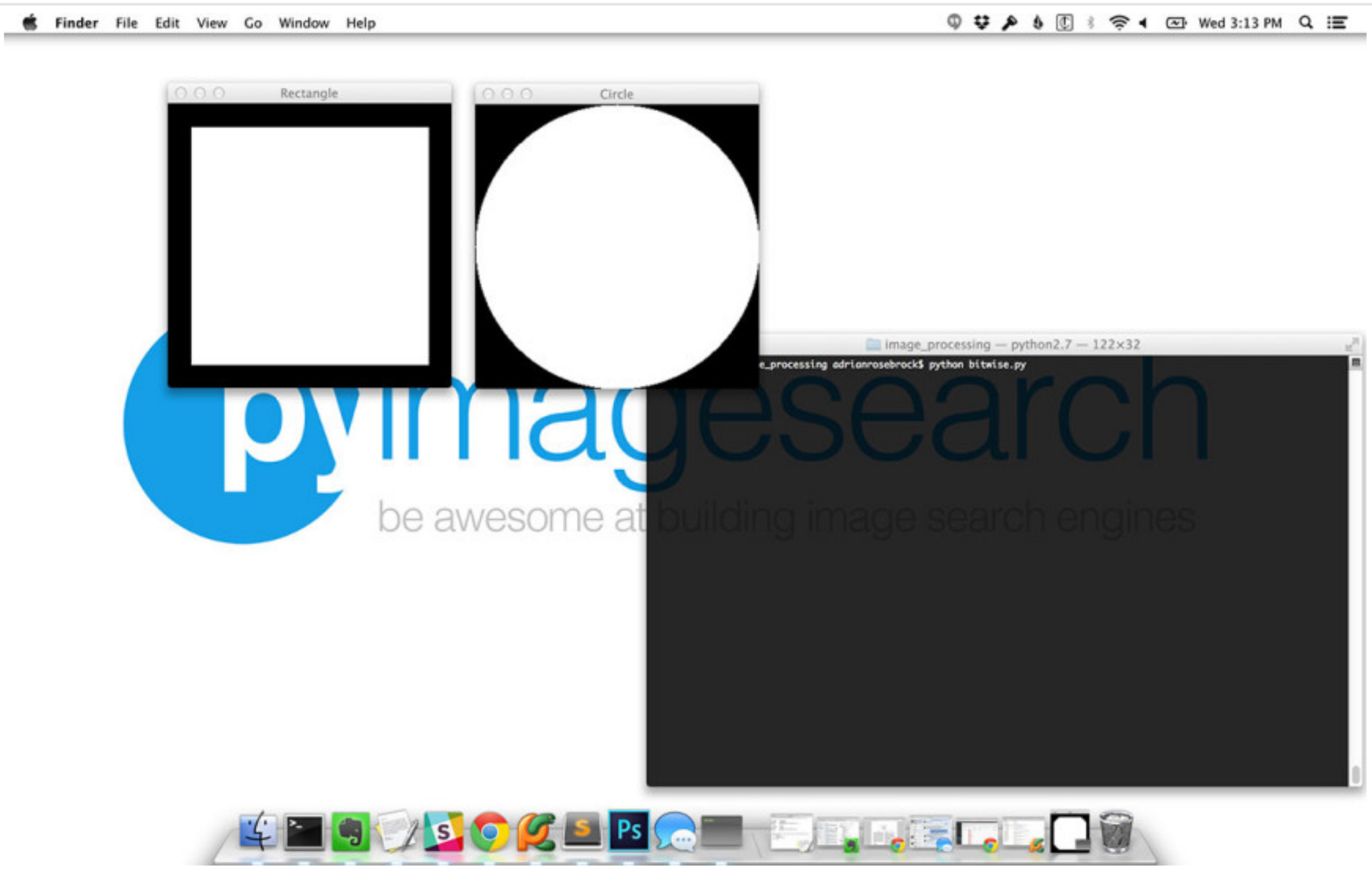
Let's go ahead and jump into some code:

```python
bitwise.py                                                          Python
1  # import the necessary packages
2  import numpy as np
3  import cv2
4
5  # first, let's draw a rectangle
6  rectangle = np.zeros((300, 300), dtype = "uint8")
7  cv2.rectangle(rectangle, (25, 25), (275, 275), 255, -1)
8  cv2.imshow("Rectangle", rectangle)
9
10 # secondly, let's draw a circle
11 circle = np.zeros((300, 300), dtype = "uint8")
12 cv2.circle(circle, (150, 150), 150, 255, -1)
13 cv2.imshow("Circle", circle)
```

The first few lines of code import the packages we will need: `numpy` and `cv2`. We initialize our rectangle image as a *300 x 300* NumPy array on **Line 6**. We then draw a *250 x 250* white rectangle at the center of the image.

Similarly, on **Line 11** we initialize another image to contain our circle, which we draw on **Line 12** again centered at the center of the image, with a radius of 150 pixels.

Figure 1 below displays our two shapes:


Finder File Edit View Go Window Help
Wed 3:13 PM
Rectangle
Circle
pyimagesearch
be awesome at building image search engines

```
image_processing — python2.7 — 122×32
e_processing adrianrosebrock$ python bitwise.py
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/bitwise_input_images.jpg)

FIGURE 1: OUR INITIAL INPUT IMAGES THAT WE WILL BE PERFORMING BITWISE OPERATIONS ON.

Feedback

If we consider these input images, we'll see that they only have two pixel intensity values — either the pixel is 0 (black) or the pixel is greater than zero (white). We call images that only have two pixel intensity values *binary* images.

Another way to think of binary images is like an on/off switch in our living room. Imagine each pixel in the *300 x 300* image is a light switch. If the switch is ***off*** then the pixel has a value of *0*. But if the pixel is ***on***, it has a value greater than zero.

In **Figure 1** above, we can see that the white pixels that comprise the rectangle and circle, respectively, all have pixels values that are *on*, whereas the surrounding pixels have a value of *off*.

Keep this notion of on/off as we demonstrate bitwise operations:

bitwise.py Python

```
15 # A bitwise 'AND' is only True when both rectangle and circle have
16 # a value that is 'ON.' Simply put, the bitwise AND function
17 # examines every pixel in rectangle and circle. If both pixels

18 # have a value greater than zero, that pixel is turned 'ON' (i.e
19 # set to 255 in the output image). If both pixels are not greater
20 # than zero, then the output pixel is left 'OFF' with a value of 0.
21 bitwiseAnd = cv2.bitwise_and(rectangle, circle)
22 cv2.imshow("AND", bitwiseAnd)
23 cv2.waitKey(0)
24
25 # A bitwise 'OR' examines every pixel in rectangle and circle. If
26 # EITHER pixel in rectangle or circle is greater than zero, then
27 # the output pixel has a value of 255, otherwise it is 0.
28 bitwiseOr = cv2.bitwise_or(rectangle, circle)
29 cv2.imshow("OR", bitwiseOr)
30 cv2.waitKey(0)
31
32 # The bitwise 'XOR' is identical to the 'OR' function, with one
33 # exception: both rectangle and circle are not allowed to BOTH
34 # have values greater than 0.
35 bitwiseXor = cv2.bitwise_xor(rectangle, circle)
36 cv2.imshow("XOR", bitwiseXor)
37 cv2.waitKey(0)
38
39 # Finally, the bitwise 'NOT' inverts the values of the pixels. Pixels
40 # with a value of 255 become 0, and pixels with a value of 0 become
41 # 255.
42 bitwiseNot = cv2.bitwise_not(circle)
43 cv2.imshow("NOT", bitwiseNot)
44 cv2.waitKey(0)
```

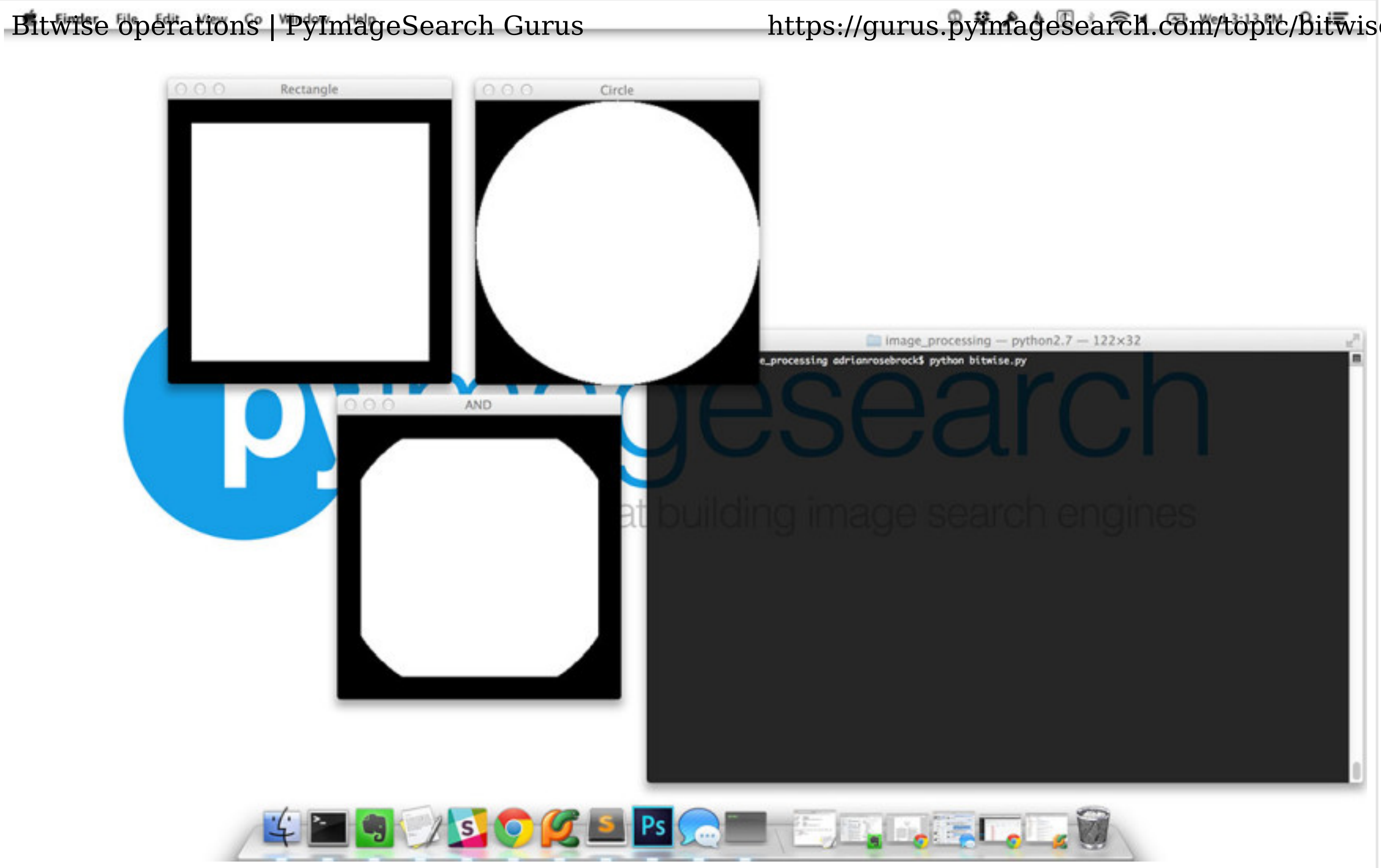
As I mentioned above, a given pixel is turned "on" if it has a value greater than zero and it is turned "off" if it has a value of zero. Bitwise functions operate on these binary conditions.

In order to utilize bitwise functions we assume (in most cases) that we are comparing two pixels (the only exception is the NOT function). We'll compare each of the pixels and then construct our bitwise representation.

Let's quickly review our binary operations:

- **AND:** A bitwise AND is true *if and only if* both pixels are greater than zero.
- **OR:** A bitwise OR is true if *either* of the two pixels are greater than zero.
- **XOR:** A bitwise XOR is true *if and only if* one of the two pixels is greater than zero, *but not both*.
- **NOT:** A bitwise NOT inverts the "on" and "off" pixels in an image.

On **Line 21** we apply a bitwise AND to our rectangle and circle images using the `cv2.bitwise_and` function. As the list above mentions, a bitwise AND is true if and only if both pixels are greater than zero. The output of our bitwise AND can be seen below:



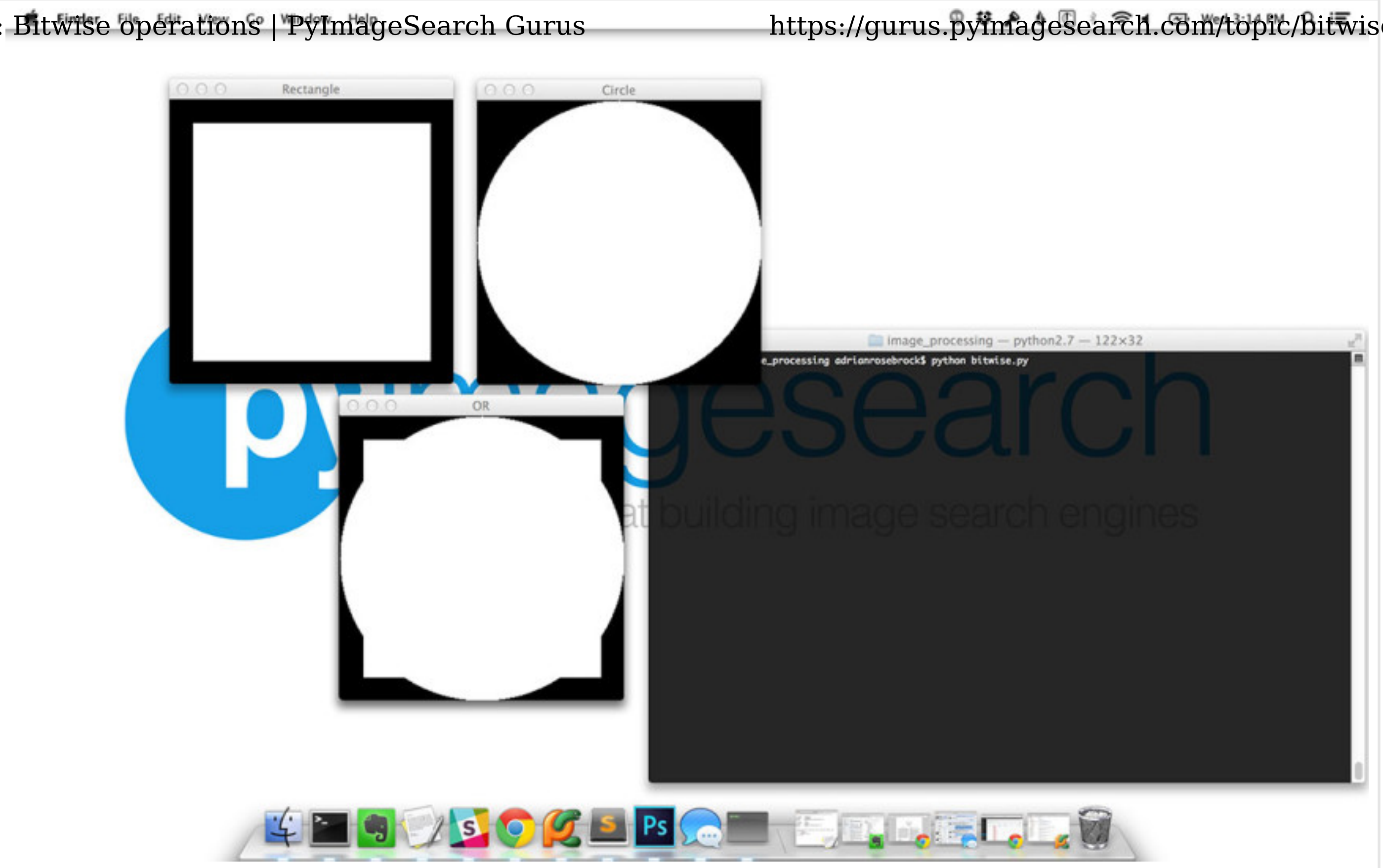
```
e_processing adrianrosebrock$ python bitwise.py
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/bitwise_and.jpg)

FIGURE 2: APPLYING A BITWISE AND.

We can see that edges of our square are lost — this makes sense because our rectangle does not cover as large of an area as the circle, and thus both pixels are not "on".

We then apply a bitwise OR on **Line 29** using the `cv2.bitwise_or` function. A bitwise OR is true if either of the two pixels are greater than zero. Take a look at the output of the bitwise OR below:



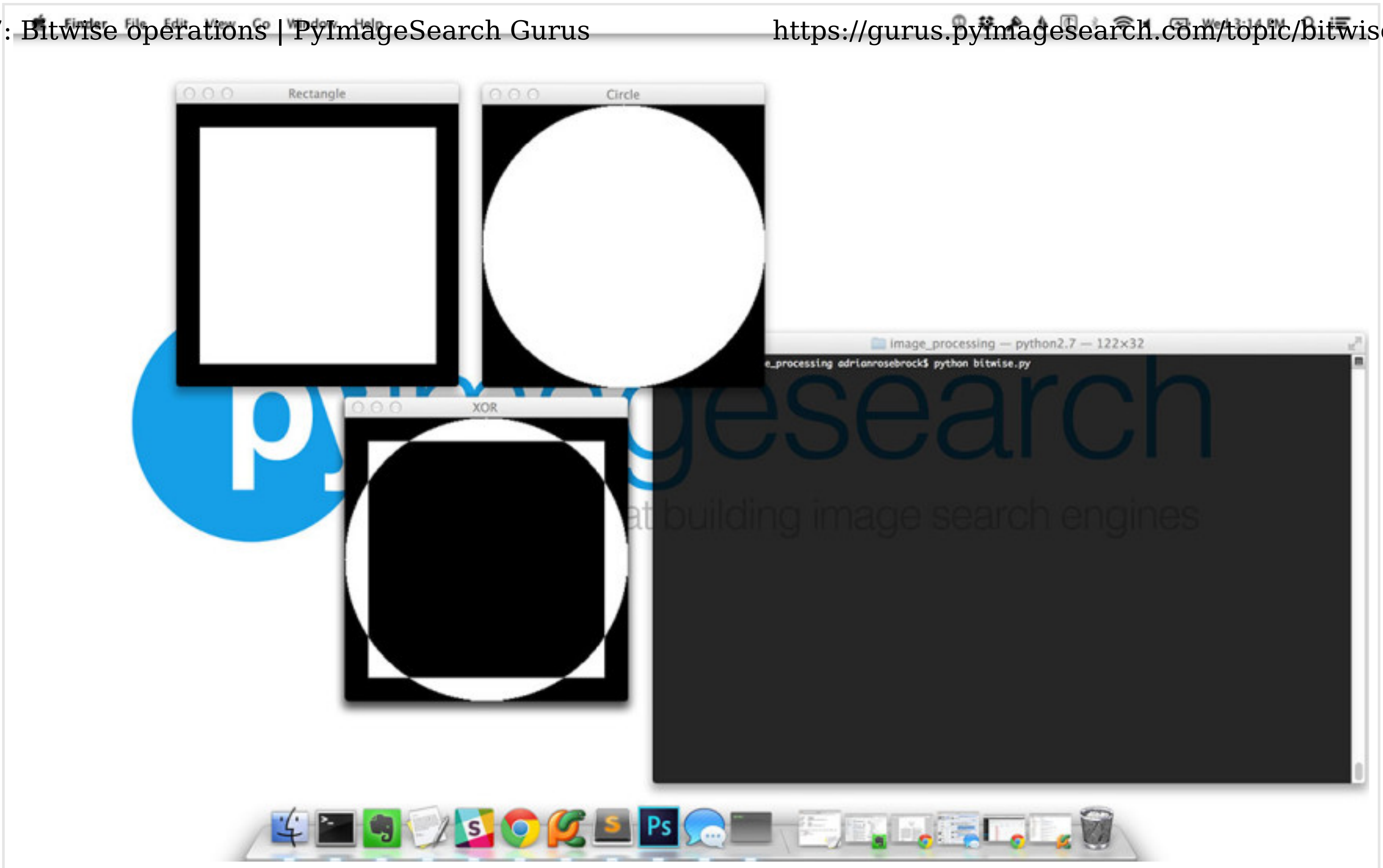
```
e_processing adrianrosebrock$ python bitwise.py
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/bitwise_or.jpg)

FIGURE 3: APPLYING A BITWISE OR.

In this case, our square and rectangle have been combined together.

Next up is the bitwise XOR function, applied on **Line 36** using the `cv2.bitwise_xor` function. A XOR operation is true if and only if one of the two pixels is greater than zero, *but*, both pixels cannot be greater than zero. The output of the XOR operation is displayed in **Figure 4:**



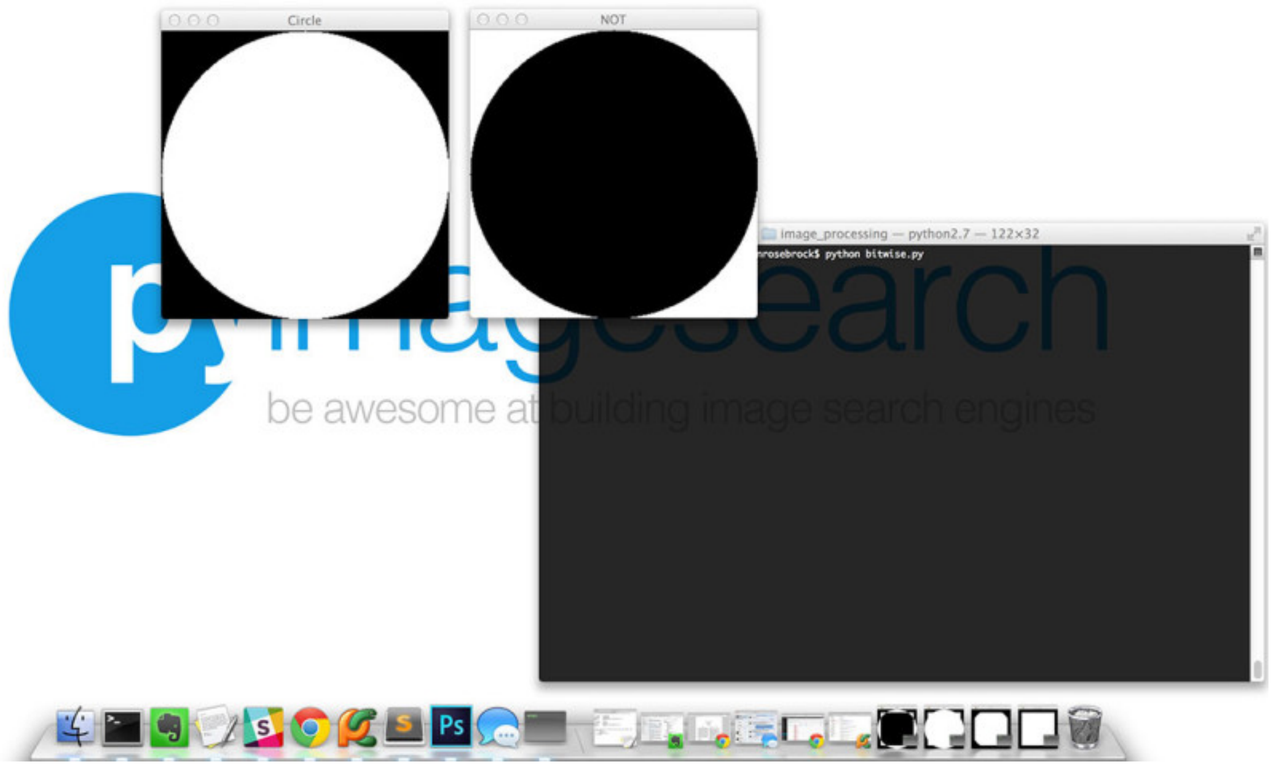
```
e_processing adrianrosebrock$ python bitwise.py
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/bitwise_xor.jpg)

FIGURE 4: APPLYING A BITWISE OR.

Here we see that the center of the square has been removed. Again, this makes sense because an XOR operation cannot have both pixels greater than zero.

Finally, we apply the NOT function on **Line 42** using the `cv2.bitwise_not` function. Essentially, the bitwise NOT function flips pixel values. All pixels that are greater than zero are set to zero, and all pixels that are set to zero are set to 255:



```
rosebrock$ python bitwise.py
```

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/bitwise_not.jpg)

FIGURE 5: APPLYING A BITWISE NOT.

Feedback

Notice how our circle has been *inverted* — originally the circle was white on a black background, now the circle is black on a white background.

Summary

Overall, bitwise functions are extremely simple, yet very powerful. And they are absolutely essential when we start to discuss *masking* in **Section 1.4.8** (https://gurus.pyimagesearch.com/topic/masking/). Take the time to practice and become familiar with bitwise operations now before proceeding with the lessons. We will be making heavy use of these operations throughout the rest of this course.

Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/computer_vision_basics/bitwise.zip)

Quizzes Status

1 Bitwise Operations Quiz (https://gurus.pyimagesearch.com/quizzes/bitwise-operations-quiz/)

← Previous Topic (https://gurus.pyimagesearch.com/topic/image-arithmetic/) Next Topic → (https://gurus.pyimagesearch.com/topic/masking/)

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Feedback

Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)

Search

Feedback