In this case, you are applying Canny to the image gray and your output will be another image called edges. low\_threshold and high\_threshold are your thresholds for edge detection.

The algorithm will first detect strong edge (strong gradient) pixels above the high\_threshold, and reject pixels below the low\_threshold. Next, pixels with values between the low\_threshold and high\_threshold will be included as long as they are connected to strong edges. The output edges is a binary image with white pixels tracing out the detected edges and black everywhere else. See the [OpenCV Canny Docs](http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html) for more details.

What would make sense as a reasonable range for these parameters? In our case, converting to grayscale has left us with an [8-bit](https://en.wikipedia.org/wiki/8-bit) image, so each pixel can take 2^8 = 256 possible values. Hence, the pixel values range from 0 to 255.

This range implies that derivatives (essentially, the value differences from pixel to pixel) will be on the scale of tens or hundreds. So, **a reasonable range for your threshold parameters would also be in the tens to hundreds**.

As far as a ratio of low\_threshold to high\_threshold, [John Canny himself recommended](http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html#steps) a low to high ratio of 1:2 or 1:3.

We'll also include Gaussian smoothing, before running Canny, which is essentially a way of suppressing noise and spurious gradients by averaging (check out the [OpenCV docs for GaussianBlur](http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#gaussianblur)). cv2.Canny() actually applies Gaussian smoothing internally, but we include it here because you can get a different result by applying further smoothing (and it's not a changeable parameter within cv2.Canny()!).

You can choose the kernel\_size for Gaussian smoothing to be any odd number. A larger kernel\_size implies averaging, or smoothing, over a larger area. The example in the previous lesson was kernel\_size = 3.

Note: If this is all sounding complicated and new to you, don't worry! We're moving pretty fast through the material here, because for now we just want you to be able to use these tools. If you would like to dive into the math underpinning these functions, please check out the free Udacity course, [Intro to Computer Vision](https://www.udacity.com/course/introduction-to-computer-vision--ud810), where the third lesson covers Gaussian filters and the sixth and seventh lessons cover edge detection.