For questions please email nick@nimble.ai

Nimble AI Programming Challenge

Skills needed:

- Python 3.4+
- Python `multiprocessing` (https://docs.python.org/3.7/library/multiprocessing.html)
- Python `numpy` (http://www.numpy.org/)
- Python `opencv` (https://pypi.org/project/opencv-python/)

Requirements:

1. Make a main python program that runs from the command line.
2. This main program should spawn three different python `multiprocessing.Process`. Must use process, not thread.
3. The first process should:
    a. Use numpy to generate an RGB image of a solid color. The color generated should be randomly selected from the following list of RGB colors: `['black', 'white', 'red', 'yellow', 'lime', 'aqua', 'blue', 'fuschia']`
    b. Create a wrapper for `multiprocessing.Array`
    c. After the image is generated it should pass the image to the second process using a `multiprocessing.Queue` (`queue_b`).
4. The second process should:
    a. Wait for a new image from an item from `queue_a`.
    b. Inspect the image in order to determine its color. Use only the image. Do not pass additional strings with the image.
    c. After inspecting the image and determining its color, use `opencv` to watermark the image with the name of the color. The second process may assume that the color is one of the colors from the above list.
    d. Draw a filled circle of the complementary color (opposite on the color wheel) in the middle of the image using a radius of ¼ the image width. Avoid using for loops to do this.
    e. After watermarking and drawing a circle on the image, place the new image onto a different `multiprocessing.Queue` (`queue_b`).
5. In the main application thread:
    a. Ask user for the number of random images to generate.
        i. Store in `multiprocessing.Value` (`num_images`).
    b. Ask the user for the dimensions of the image.
        i. Store in `multiprocessing.Value` (`height` and `width`).
    c. Spawn the three processes.
    d. Wait for a new image from `queue_b`
    e. Store the image in a `multiprocessing.Array` (`array_a`).
    f. Allow the user to quit the program with '`q`'.
        i. Use `multiprocessing.Event` (`event_quit`) to signal program completion to the processes.
    g. Properly join all processes and cleanly exit.
6. The third process should:
    a. Continually read from `array_a`.
        i. `numpy.frombuffer` will be useful for reading and writing this array efficiently.
    b. Display the image using `opencv.imshow`.