Image Processing 2

You will program 2 more complex image manipulations. Just as before, each function will take the original image as a parameter and return a **new** image without modifying the original. Create a new file named photolab2.py which will contain your 2 functions and a `main()` function. `main()` should draw the original image and the 2 manipulated images such that all 3 can be seen at once. Each of the new functions requires a second parameter besides the original image; you should read all 3 necessary user inputs (original image, scale factor, blur radius) from the command line as explained in the first image processing project.

This assignment should be completed with your newly assigned partner. Make sure to include your names and use good coding standard!

Deliverables: zip folder named username1_username2 with your `photolab2.py`.

Applying advanced filters to an image

The 2 new manipulation functions are as follows:

- `scale(image, scale_factor)`

  This function returns a new image that is a scaled version of `image`. The `scale_factor` must be a positive float. If `scale_factor` is 2, for example, each side of your new image should be double the size of the original. If the original image was 100x100 pixels, the new image will be 200x200 pixels. Likewise, if `scale_factor` is 0.25, each side of your new image should be 1/4 the size of the original, or 25x25 if the original image was 100x100. Clearly you will only want to test relatively small scale factors.

  Question: What if the new image is bigger than the original? Do you need to interpolate between pixels to calculate appropriate pixel values and so on? Answer: No. Every pixel on the new image should just be a copy of some pixel on the old. The challenge is in figuring out which one. If the new image is smaller than the old, then some of the pixels on the original will be ignored. If the new image is larger than the old, than some of the pixels from the original will be duplicated.

  Hint, of sorts: You've got a choice to make. You can use the code

  ```
  blankIm = EmptyImage(width, height)
  ```

  to create a blank image of a given size, but do you loop over all the pixels in the old image, and figure out where they belong in the new image? Or do you loop over all the pixels in the new image (which are initially unassigned), and for each one figure out which is the right pixel in the old image to copy in? One of these approaches is *much* easier than the other. Definitely spend time in advance thinking about what kinds of calculations you would have to do for each of these approaches, and ask for help if you need it.

- `blur(image, radius)`
  This function returns a copy of the image that has been blurred. For each pixel, average all of the pixels within a square centered at that pixel and with a `radius` as given. In other words, if your radius is 3, your square should go 3 pixels left, 3 pixels right, 3 pixels up, and 3 pixels down from the center to form a 7 x 7 square. Make sure to put all of your results in a **new** image, instead of overwriting your original as you go; otherwise, your blurred pixels will cascade on top of each other. You'll want to work with a small image for this task, as blurring in Python can take some time. Even though blur will be somewhat slow in general, some approaches result in much slower algorithms than others. Any technique that works will receive most of the points for this problem (assuming it runs in under one minute for a radius of 3, I can't keep the grader testing these indefinitely), but efficiency will be considered as well.

  You can choose to ignore blurring around the edges of the image, that is in the range where the blurring square would run off the edge of the image. Alternatively, if you like, you can blur those pixels by using whatever pixels you do have in range.

Every line in your file (besides comments, import statements, and a single line calling main()) should be part of a function. Remember to use good coding standard and comment appropriately. Have fun!