Computer Science 135 (Spring 2017)
Duane A. Bailey
Laboratory 5
Building Image Filters (due 5pm, Friday, March 17)

Objective. To learn how to use classes to reduce code complexity.

In this lab, we'll investigate how we might write code to transform images. For example, we'll think about how to convert color images to monochrome images using a grayscale filter. Our work here actually duplicates efforts in the Python Image Library (PIL) and will be a bit less efficient. Still, our approach to building special-purpose image filters parallels techniques we'll commonly use to build up toolkits to solve other big data problems.

Before we get started, we're going to need to make use of the PIL we installed in our own personal environments two weeks ago. We need to make sure that we activate our virtual environment to get access to that image-handling software:

```
$ cd ~/cs135
$ source bin/activate
(cs135)$
```

Make sure you clone your lab5 repository. Recall that if you are 18xyz, you type:

```
(cs135)$ git clone ssh://18xyz@gala.cs.williams.edu/~cs135/18xyz/lab5.git lab5
```

The RGB Colorspace Model.
This lab depends heavily on the red-green-blue (RGB) colorspace model. This models colors that occur as the result of mixing three colored lights with varying intensity. For technical reasons, we'll encode the intensities as integer values between 0 and 255. When the intensity is 0, that light does not contribute to the final mixture while an intensity of 255 indicates that light is on full. We'll think of a colors, then, as being represented by 3-tuples (viz. \((r,g,b)\)) of integers between 0 and 255. The color black is represented by \((0,0,0)\) and white is \((255,255,255)\). Red would be represented by \((255,0,0)\), and dark red might be \((100,0,0)\). All monochrome values (shades of gray) have equal amounts of red, green, and blue.

The Filter Classes.
In the file `filter.py` you'll find a number of classes that I have written that describe filters. These are objects that, when they're constructed can be used to perform a transformation on the pixels of an image. For example, you can use the GrayFilter, which transforms the color pixels (the before pixels) of an image into gray pixels (the after pixels). If you're finished with modifying the image, you can ask for the result of this process as a PIL Image and then save it to disk.

```
from PIL import Image
flowers = Image.open('Irises.png')
filteredImage = GrayFilter(flowers)
grayIrises = filteredImage.image()
grayIrises.save('GrayIrises.png')
```

(Recall that, once saved to disk, you can type

```
(cs135)$ open GrayIrises.png
```

to preview the image.)
Tasks to be done.
You’ll find a file, lab5.py in your repository. I’d like you to add a few new filters to that module. You’ll find a class, Template, that you can cut-and-paste as a start. Typically, a new filter needs to have its name changed (from Template) and you need to modify the after(x,y) method to describe how you’ll transform the before pixels. Here’s what I need from you by Friday at 5pm:

1. Write two new filters: LighterFilter and DarkerFilter. These bring the RGB values closer to white and black, respectively. Test these out, using PNG images of your choice. Make sure your filters can be composed; e.g. GrayFilter(LighterFilter(flowers)).

2. Write a filter that remaps the colors of one of Van Gogh’s images: Bedroom, Sunflowers, or Roses. We have the conservator’s expectation of Bedroom as BedroomExpected, which increases the use of red. In Sunflowers, we hope to make the muddy brown-green more yellow, while in Roses, we hope to make the lightest areas more pink. FadingColors.pdf is a technical discussion of the chemistry.

3. If you wish, write a fourth filter of your choice (credit varies based on effort). Here are some ideas:
   (a) Write a filter that keeps only pixels that are ‘close to’ a target color, while other pixels become white. You’ll have to extend the template class to include a slot that remembers the target color. You’ll also have to think about a distance metric.
   (b) Write a filter that blurs an image. This filter computes a weighted average of pixels in a neighborhood. This is called a kernel. A kernel for blurring is
   
   
   Imagine the pixel you’re computing is located at the center of the kernel (the parenthesized 0), and compute the weighted sum of all the pixels relative to this center pixel. For example, the pixel at (x − 2, y − 2) is weighted 1, but the pixel at (x − 1, y − 1) is weighted 0. After computing the sum, the blurred pixel value is this sum divided by 16 (the sum of all the weights in the kernel). You’ll need to think about how to multiply a color by a weight and how to sum them, of course. You’ll also have to think about how to represent the kernel, itself.

You might think about how this filter works.
   (c) Write an edge-enhancement filter. It has the kernel

   

   The average is computed by dividing by 2. This filter is also interesting to think about.

4. Document your efforts. Make sure your module, its classes, and its methods all have doc comments.

5. Push your repository