Simple Image Processing

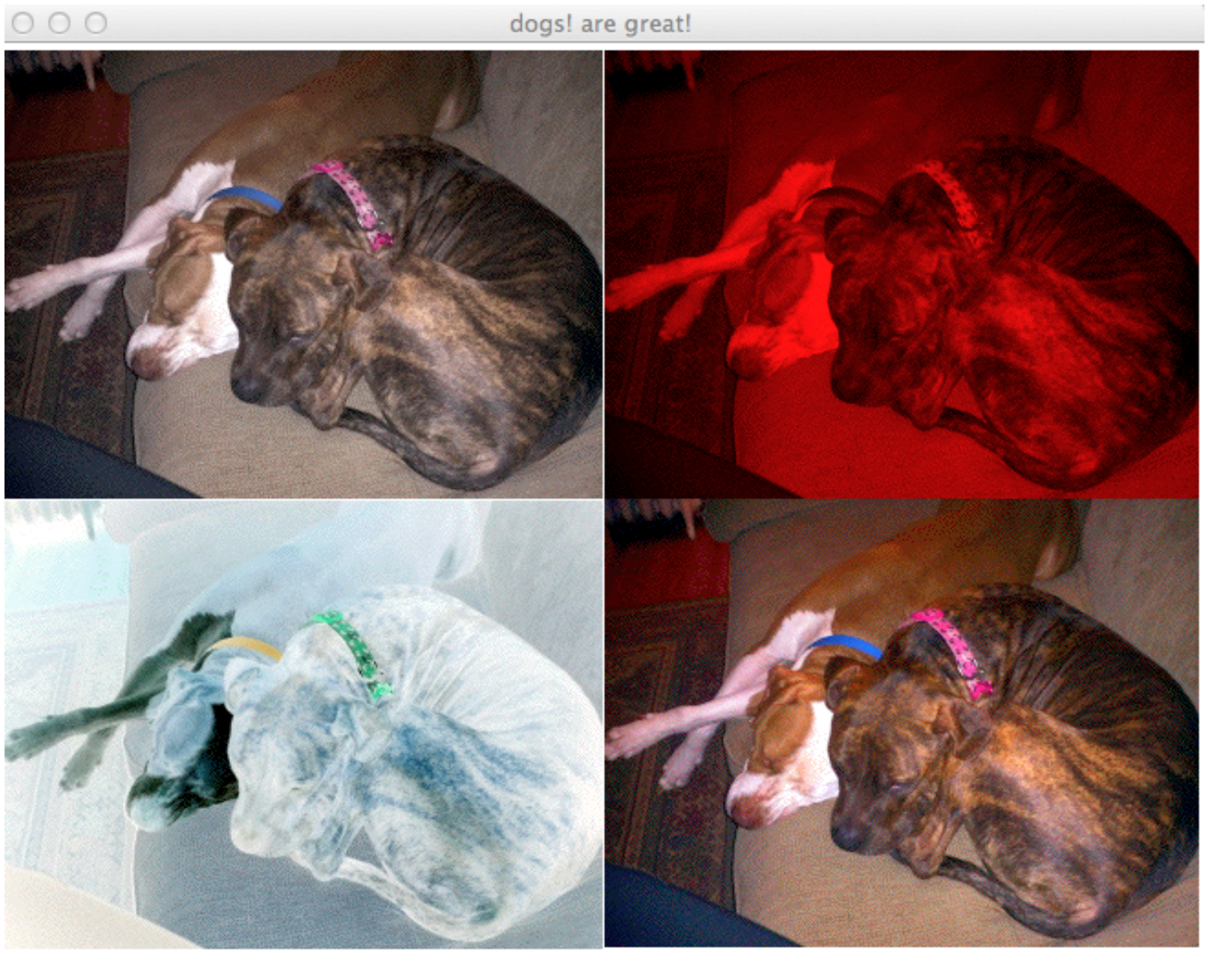
You should complete this assignment with your new partner from partners 1. This is your chance to practice applying some basics filters to an image. It is a warmup to the next assignment, where you'll do some more complicated special effects.

Deliverables: Your `user1_user2.py` file.

Getting Started

To facilitate reading in, storing, and manipulating image files, we will use a module similar to Zelle's graphics but with some added functionality. This module, `cImage.py`, is included in the materials folder for this assignment. Make sure it is in the same directory as your program.

I have also included the program we discussed in class that creates a greyscale version of an image. Start with this program and add the 3 filters described below, each in its own function. Also add code to the main() function to test your new filters by drawing each filtered image to the window with the original. You should display the 4 images (the original and the 3 modified images after applying your 3 filters) in a square such that you can see all 4 no matter what size the original image is (unless it's too big to fit 4 of it on your screen). For example, here is my final window after applying my filters to a picture of my dogs:



The orig image is in the upper left, the oneColor image with red as the color filter is in the upper right, the negative image is in the lower left, and the saturated image with a k value of 2 is in the lower right.

At the end of this assignment are descriptions of several useful functions from `cImage.py`.

You'll also need some images to work with. I have included the one of my dogs in the materials folder but you should try out your program on your own images too!

Filters to add

Your task is to add the following 3 image manipulation functions and complete the `main()` function. Each manipulation function will take a parameter *image* that will represent the original image loaded by the user. You will NOT change this image but rather **create and return** a new image with the desired modifications. The `main()` program will draw the orig image and each of your 3 filtered images as described above.

The functions you will write are as follows:

- `oneColor(image, color)`
  color will be a string, either `"r"`, `"g"`, or `"b"` representing red, green, or blue. Returns an image containing only the red, green, or blue aspects of the original image, depending on the color argument. You can do this by setting the values of the other 2 colors to 0 in each pixel.

- `negate(image)`
  Returns an image that is a conversion of the original image to "negative" form (like a photographic negative). Here, you don't necessarily have to duplicate precisely the same technical visual effects that a photonegative actually uses. But you should modify all colors in a way that turns black into white, white into black, dark colors into light, light into dark, etc.

- `saturate(image, k)`
  Returns an image with appropriate color saturation based on k. k is a float >=0 representing the level of color saturation. A k of 1 will leave the image unchanged, a k between 0 and 1 will result in less color (k=0 results in a grayscale image), and a k>1 will result in more color.

  I have provided you with the function `saturatedRGB(r, g, b, k)` which implements the saturation calculations; you do not need to do any of the math equations (nor should you change the ones I provide). The provided function takes 4 inputs - the r, g, and b values from a pixel plus an intensity value (k) - and returns a list with the new r, g, and b values for the appropriately saturated pixel.

  You will write your own completely separate saturate function, but part of your function will be to call saturatedRGB for each pixel in the image and write a pixel with the new returned saturated r, g, b values to the modified image. The purpose of this is to practice calling a function from a function using different numbers/types of parameters and return values.

- `main()`
  Write a `main()` function that tests the above functions by loading an image file and making oneColor, negative, and saturated versions of it appear on the screen. You will need to get 3 values from the user: the image file to load, the color to use in `oneColor`, and the k value for saturation. Instead of using `input` to request these from the user, we will use command line arguments (read the notes on command line arguments below). When we test your code, we may remove your `main()` and add our own. If you have carefully followed the above specifications, it should work perfectly! As an example, we should be able to write code in our `main()` like:

```
origIm = FileImage(sys.argv[1])
win = ImageWin("testing", origIm.getWidth()*2, origIm.getHeight()*2)
origIm.draw(win)
colorIm = oneColor(origIm, sys.argv[2])
colorIm.draw(win)
```

Note that all of the functions should create and return a NEW image that is the conversion of the original, they should NOT change the original image itself! Each of these functions should be no more than a few lines of code long, don't make them more complicated than necessary =)

Command line arguments

An often more convenient way to get input from a user than asking for it in your program is to allow them to use command line arguments. This allows the user to type one line to run your program with the appropriate inputs. For example to run my `photolab.py` program I could type something like

```
python3  photolab.py  SherriAddie.gif  b  3
```

and the program should load the image file `SherriAddie.gif`, use "b" for the `oneColor` function, and a `k` of `3` for the `saturate` function. How do you get these values in your actual program? First you must import the `sys` module. This module provides a list named `argv` of all the command line arguments . The first argument, `sys.argv[0]`, is always the name of your program (in this case `photolab.py`). The rest of the list is any other arguments the user enters. So as you can see in my sample code above, you can access each argument using `sys.argv[index]` with the appropriate index. Note that just like `raw_input`, these arguments are all stored as `strings` in the `sys.argv` list, so you must convert them if you wish to use them as a number or other type.

Good luck, and have fun!

`image = FileImage(fileName)`
– creates a new image object from fileName.

`win = ImageWin(title, width, height)`
– create a window of given width and height to draw images to

`image.copy()`
– return a copy of the image

`image.draw(window)`
– draw the image in the given window.

`image.getNumPixels()`
– return the # of total pixels in the image

`image.getHeight()`
– return height of the image as # of pixels

`image.getWidth()`
– return width of the image as # of pixels

`image.getPixel1D(index)`
– return the pixel at the given index in the image. In this case the pixels are considered as one long list containing all pixels in the image, so index must be in the range [0, getNumPixels()-1].

`image.setPixel1D(index, pixel)`
– Set the value of a pixel at position index. In this case the pixels are considered as one long list containing all pixels in the image, so index must be in the range [0, getNumPixels()-1].

`pixel.red, pixel.green, pixel.blue`
– given a pixel, use this to either get or set the value of the r, g, or b color of that pixel.