COSC 101 Homework 5 Fall 2014

Due date: **Wednesday, October 8, 11:55pm**

In this homework you will write a series of programs to perform different manipulations of digital images.[1] The goal is for you to get practice with `for` loops, nested `for` loops, and conditional statements.

As before, please use good variable names and include a few descriptive comments (e.g. you are required to label the purpose of each `for` loop: the transformation each performs).

1 Getting started

- If you have not already, read the background information on image transformations, contained in the `image.pdf` document.
- If you have never seen rutabaga paired with garlic and bok choy, then you probably need to re-read the background information before continuing.
- Open the demo program, called `image_demo.py`, in IDLE and make sure it runs properly.
  Do not attempt the tasks below until you can run the demo programs successfully.
- Open `hw5.py` in IDLE. You should write all of your programs in this file. Be sure that your file is saved in the same folder as `image.py` and the other files we have provided.

Note that the image `crayons_small.png` is provided for you to use when you are developing your code. Each transformation takes time so it helps to work with a smaller image; you can also comment out the transformations you are not currently working on. Remember to change your final program so that `crayons.png` is used and to uncomment all the transformations.

2 Your task

Your task is to complete the following seven image transformations.

Important: your final program should have one, or perhaps two, calls to the `display_images` method at the *end* of the file so to display all **seven** images you created. (If you run your program on a small screen, such as a laptop, you may need two calls to `display_images` because all seven images won't fit in one window.)

A challenge problem is mentioned in Section 4.

3 Pixel-by-pixel transformations

The following image transformations alter the image by applying the same transformation to each pixel in the image. You can solve each one by writing a pair of nested loops similar to what is done in red filter example program described in Section 2.3 of `image.pdf`. The main difference is in which color values are examined and how the colors are changed.

[1] Acknowledgments: we adapted these assignments from previous assignments by Tom Wexler, Nick Parlante, Mark Guzdial.

The first six transformations are applied to the `crayons.png` image, the seventh one is using `obama.png`.

1. **Cycle Color Channel**. Use a transformation that moves the red value to the blue channel, the green to the red channel, and the blue to the green one.

original

color cycle

2. **Grayscale**. Shades of gray have the same red, green and blue value. To convert an image to grayscale, set each of the red, green and blue channel to the average value of the original pixel.

original

grayscale

3. **Negatives**. The negative of an image is created by inverting each color channel. So if the red value is 255, it becomes 0; if it is 254, it becomes 1, and so on, down to 0, which becomes 255. Similarly for green and blue.

original

negative

4. **Brightness**. Prompt the user for an integer change to the brightness: a positive or negative number. Adjust the color values by the entered amount. Just be sure no values go over 255 or below 0. Below first the brightness is increased by 70; second it is decreased by 70 from the original.

original

positive brightness

original

negative brightness

5. **Increase Contrast**. When increasing the contrast,

   - color values at 128 are unchanged and
   - for any other value x, the difference between x and 128 is scaled a factor of 2.

   For example, a color value of 129 (1 above 128) becomes 130 (2 above 128); 125 (3 below 128) becomes 122 (6 below 128). Just remember that values need to stay in the range $[0 - 255]$.

original

contrast

6. **Posterize**. Typically a pixel use one of 256 value (any integer in the range [0 – 255]) for each color channel. In a posterized image, the set of numbers a pixel used is drastically decreased: only multiple of 32 are used using the same range. Use a transformation to round *down* to the nearest multiple of 32 the value of each color channel.

original

posterize

7. **Obamafy**.

   For this transformation every pixel is assigned one of four colors:

   - yellow with `(R, G, B) = (252, 227, 166)`
   - dark blue with `(R, G, B) = ( 0, 51, 76)`
   - light blue with `(R, G, B) = (112, 150, 158)`
   - red with `(R, G, B) = (217, 26, 33)`

   The assignment is based on the pixel's *grayscale value*.

   - Grayscale above 182 is yellow.
   - Grayscale between 182 and 121 is light blue.
   - Grayscale between 121 and 60 is red.
   - Grayscale below 60 is dark blue.

original

Obamafy

4 Challenge problem

1. **Chroma-key blending** For this problem, you will need to blend *two* images using a "chroma-key." The left and center pictures below show a stop sign and leaves, respectively. What you'll need to do is to create an image similar to the right one, in which the red of the stop sign is replaced by pixels from the image of leaves.

   Chroma key blending works based on pixel color, not position. In other words, *every* "red" pixel in the *entire* image is replaced, regardless of whether that pixel is located within the octagonal stop sign shape. Chroma key blending is also known as "green screening" – e.g., TV weather persons stand in front a green screen, which is then digitally replaced with a weather map. (For more information, see `http://en.wikipedia.org/wiki/Chroma_key`.)

   background foreground chroma key blended

   

   To perform the blending, you will have to develop a method to detect pixels that are red (or "reddish"). You cannot simply examine the red component of a pixel; you should try to detect situations in which the red component is significantly larger than the other components. *You will need to experiment, trying different notions of "red," until it looks about right.* When you find a red pixel at coordinates x,y in the stop sign image, you can overwrite that pixel with another pixel from the leaves image, retrieved from the same x,y coordinates.