COSC 101: Introduction to Computing I
Homework 7: Image Collage
Fall 2014

In this homework you will write a series of functions to perform different manipulations of digital images and ultimately make a collage! The goal of this homework is for you to get practice with `for` loops, nested `for` loops, conditional statements, and most importantly, learn how to define and use functions.

This homework is **due on Wednesday, October 22, 2014 11:55pm**. As before, please use good variable names and include a few descriptive comments. In addition, each function should include a *docstring*.

1 Getting started

- This assignment is a continuation of the previous assignments on images. If you need to review the background information on image transformations, consult the reference material included in the previous assignments and also included in the .zip file for this assignment.
- The .zip file contains `hw7.py`. Open `hw7.py` in IDLE. Make sure this program runs successfully and displays the "boring collage" before continuing further. You should write all of your functions in this file.

2 Your task

This homework has four parts.

1. First, recall the previous image homework and lab assignments in which you wrote several image transformations (grayscale, color cycle, negative, brightness, etc.). Choose **any two** of these transformations and write each one as a function. For example, if you choose grayscale as one of your transformations, you would write a function called `grayscale` that has one parameter, an image object, and the function should return a copy of the image in grayscale. For guidance on how to do this, look at how `red_filter` is defined and called inside `hw7.py`. Importantly, notice how `image.load_from_file` does **not** appear inside the body of the function; instead it appears inside `main` where `red_filter` is called. (Your function can take additional parameters as needed – e.g., brightness might take a brightness amount).
2. Second, write two new transformations. Choose **any two** transformations from among the transformations described in Section 3 and write each of them as a separate function.
3. Third, write a function called `copy_into(imgA, imgB, x, y)` that takes two image objects, `imgA` and `imgB`, and copies `imgA` into `imgB` with the upper left corner positioned at `x,y` in `imgB`. Important: *if* `imgA` *is too big to fit, your function should copy as much of* `imgA` *as will fit into* `imgB` *at the given position.*

4. Finally, use all of these different image functions to create and display a collage! The collage is described in more detail in Section 4.

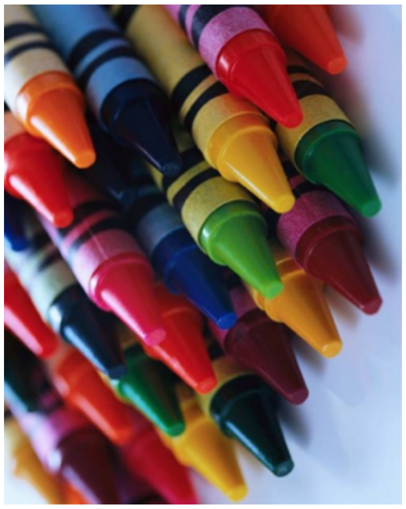
3 Transformations

For part 2 of your task, implement **any two** of the following four transformations. All of your image transformations must be written as functions inside `hw7.py`.

1. **Rotate left**. Rotate left by 90 degrees. The crayons picture is 360 (wide) × 286 (high) but when rotated left it becomes 286 (wide) × 360 (high). Hint: since the dimensions of the rotated image are different than the original image, you cannot start by copying the original image. Instead, you will need to use the `image.new_image(w, h)` function to make a new image object with the appropriate dimensions.

original

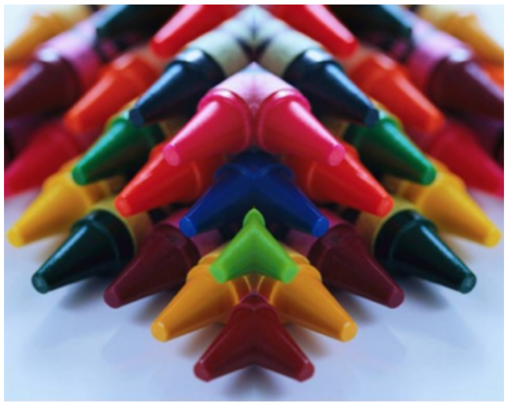
rotate left



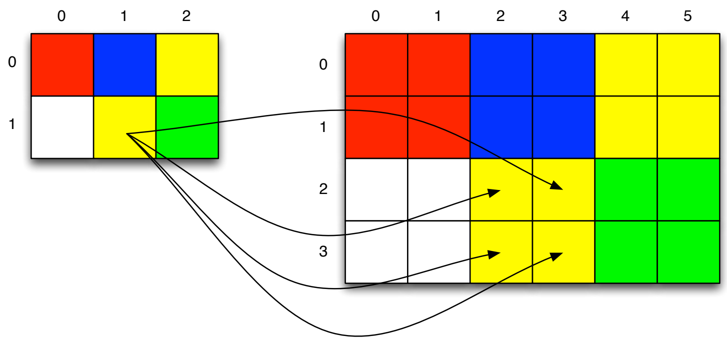
2. **Mirror**. Mirror the image around a vertical axis at its center.

original

mirror



3. **Scale up**. Blow up the image by a factor of 2. The new image is twice as wide and twice as high. Each pixel in the original image is mapped to the corresponding 4 pixels in the new image. For example, in this 3 x 2 image, the yellow pixel at (x, y) = (1,1) is copied over to four pixels in the new image at (2, 2), (2,3), (3,2), and (3,3).



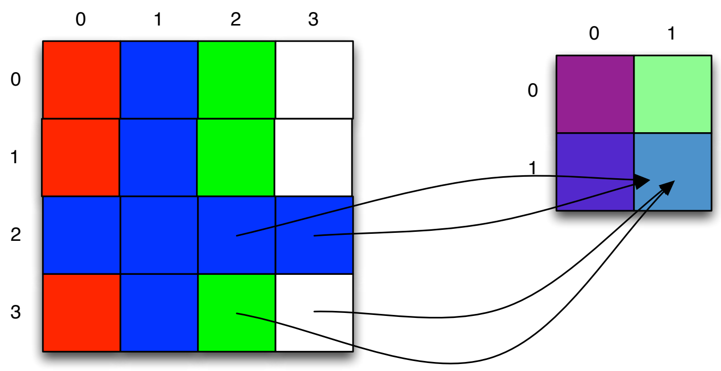
Here is what the crayon image looks like scaled up:

original

scale up

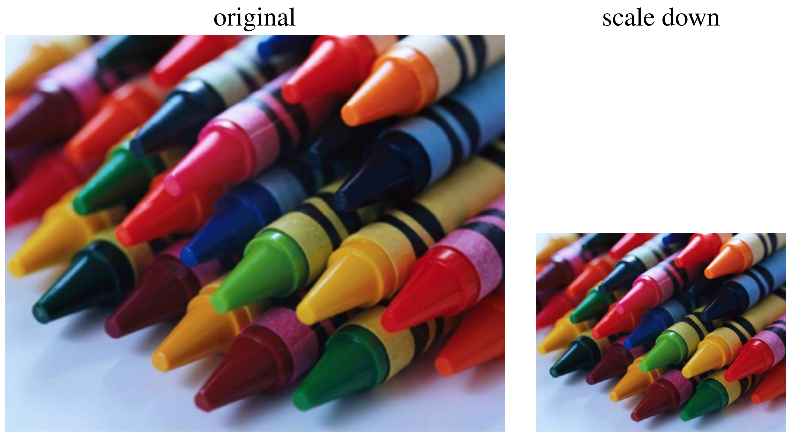
4. **Scale down**. Shrink an image by a factor of 2. This is easier if the height and width of the image are both divisible by 2. If the height or width is odd, then simply "ignore" the last row/column of the original image.

   Scale down is almost the inverse of scale up: four pixels in the original image are mapped to a single pixel in the scaled down image. To map four to one, each color in the one pixel is the average value of that color in the four pixels from which it is mapped. For example, consider the image on the left and its scaled down image on the right.



The pixel in the scaled down picture at (x, y) = (1, 1) is an average of four pixels: two blues (0,0,255), one green (0,255,0), and one white (255,255,255). When you average these together and convert each average value to an `int`, you get an (R, G, B) = (63, 127, 191). This appears as a kind of light blue.

Here is what the crayon image looks like scaled down:



4 Collage

All of your image transformations must be written as functions inside `hw7.py`. You must have at least 5 functions: 2 for the old transformations, 2 for the new ones, and the `copy_into` function (part 3 of your task, see page 1). You should test each function independently before to attempt the collage.

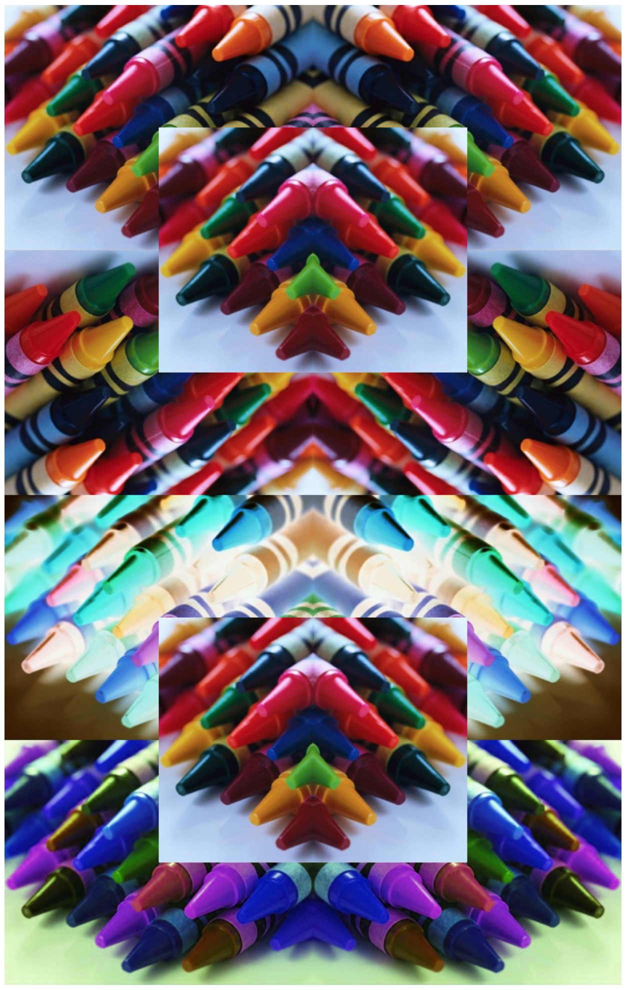
Your last task is to write a main program that *calls* these functions, making a collage. You should write this program inside the function called `main` which has already been defined in `hw7.py`.

Here are the guidelines for the collage. These guidelines represent the minimum requirements for full credit. More creative collage designs can earn you challenge problem bonus points.

- You can use any images you like, provided that they are in .png format and they are not offensive (G or PG rating, please!). If you have a JPEG or other image format, you can easily convert it to .png using tools online. A google search for “convert image to png format” turns up several options. In addition, make the images small! We recommend no bigger than 100 x 100 pixels, otherwise your program will take a long time to run.
- It must also include at least 4 transformations (two old, two new).

- Compose the transformed images into a collage. A simple composition is shown below but feel free to get more creative! In building the collage, you **must** use your `copy_into` function.

The image below shows a collage of crayons. Along the left half, from top to bottom, there are four copies of the crayons image: original, upside down (rotate left twice), negative, and color cycle. Then two copies of the original crayons image are copied on top of these tiles. Finally the left half is mirrored onto the right half. This image is twice the width of the original crayons image and four times the height.



Be sure to write a comment in `main` describing your image. Here is the comment from my code:

```
# Collage Description
# Dimensions: 2*w by 4*h of crayons image
# 4 crayon images arranged like tiles from top to bottom:
# 1. original
# 2. upside down (rotate left twice)
# 3. negative
# 4. color cycle
# On top of these tiles, two copies of the original, offset
# Then the entire image is mirrored!
```

Hints:

- Build up gradually... each time you add something new (e.g., a new transformation), run your code and make sure it works before continuing on.
- Conceptually, it may help to think of creating a new image of a given size and call this your canvas. To make the canvas, use the `image.new_image(w,h)` function and pass in the appropriate dimensions of the size canvas you want.
- Take your original image and transform it several times, saving the transformed images as variables in your main program.
- Then build the collage by copying these transformed images onto various positions on this canvas (using your `copy_into` function).
- When designing your collage, you might start with a really tiny image. For example, `tiny_smiley.png` is 45 x 45 and might be a good test image to work with. If your programs are well designed, changing which image you use is as easy as changing a single line in `main`.

5 Challenge problem

The challenge problem this week is to make a more creative collage. Design new transformations, or make a more complex collage, be creative! We will select the best collages and, with your permission, showcase them for others to see.

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