Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2017

Prelab Exercises for Lab 5
Incremental Development of an Image Processing Module - First Iteration

Have your solutions to these exercises available when you attend Lab 5 (save them on a flash drive or a cloud-based file storage service, or email the Python file to yourself).

Getting Started

Step 1: Create a new folder named Lab 5 Prelab.

Step 2: Download filters.py, Cimpl.py and the image (JPEG) files from cuLearn to your Lab 5 Prelab folder.

Step 3: Launch Wing IDE 101. Check the message Python displays in the shell window and verify that Wing is running Python version 3.6.

General Requirements

All the functions that you develop must have a docstring containing:

- the function's type contract,
- a brief description of what the function does, and
- an example of how we can interactively test the function from the shell.

Exercise 1

Step 1: Open filters.py in a Wing IDE editor window. This module contains four filters: `grayscale`, `solarize`, `black_and_white` and `black_and_white_and_gray`. Don't modify or delete these functions!

Step 2: Click the Run button. This will load the module into the Python interpreter and check it for syntax errors.

Step 3: Read the definition of `grayscale` (the header, the docstring and the function body). Using the approach you learned in Lab 4, call `Cimpl` functions from the Python shell to select an image file and load it into memory. Call `grayscale` to modify this image, then display the modified image.

Exercise 2 - Converting an Image to Grayscale, Revisited (Improving the Algorithm)

This exercise introduces an important software engineering technique: after we design, code and test a function, we sometimes *refactor* it to improve it; for example, make it more understandable and maintainable, increase its efficiency, or improve the results it produces.

The `grayscale` filter converts each pixel's colour to a shade of gray by using this formula to calculate the pixel's *brightness* (the average of its red, green and blue components):

```
brightness = (red + green + blue) // 3
```

and setting all three components to this average.

This calculation is (almost) equivalent to this formula:

```
brightness = red × 0.3333 + green × 0.3333 + blue × 0.3333
```

In other words, the red, green and blue components are weighted equally when calculating the average, with each component contributing 33-and-a-third percent.

This way of calculating grayscale does not take into account how the human eye perceives brightness, because we perceive blue to be darker than red. A better way of averaging the three components is to change the weight of the red component (to 29.9%), the green component (to 58.7%) and the blue component (to 11.4%):

```
brightness = red × 0.299 + green × 0.587 + blue × 0.114
```

In filters.py, make a copy of your `grayscale` function and rename this copy `weighted_grayscale`. **Don't modify your original grayscale filter.**

Edit `weighted_grayscale` so that, for each pixel, it calculates the weighted average of the pixel's red, green and blue components and uses that value to create the shade of gray for the pixel.

Use the shell to test `weighted_grayscale`. Compare the images produced by this function with the images produced your original `grayscale` function. In your opinion, which grayscale filter produces better-looking images?

Exercise 3 - Negative Images

To create a negative image, we change the red, green and blue components of each pixel to the "opposite" of their current values. What do we mean by "opposite"? Recall that a component's value can range from 0 (lowest intensity) to 255 (highest intensity). If the red component is 0, its opposite value is 255. If the green component is 255, its opposite value is 0. If the blue component is 140 (slightly above the midpoint in the range of intensities), its opposite value is 115 (i.e., slightly below the midpoint).

Using this information, we can derive a mathematical function that maps a component, v, to its corresponding opposite value:

$$f : v \rightarrow f(v), \text{ where } f(v) = 255 - v$$

In filters.py, define a function that is passed an image and converts it to a colour negative of the original image. The function header is:

```
def negative(image):
```

Use the shell to test `negative`.

Wrapup

Your filters.py module should now have these six functions: `grayscale`, `solarize`, `black_and_white`, `black_and_white_and_gray`, `weighted_grayscale` and `negative`.

In Lab 5, you'll experiment with the `solarize`, `black_and_white` and `black_and_white_and_gray` filters and add functions to your `filters` module. Remember to bring a copy to your next lab session.