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

# Lab 4 - Introduction to Image Processing

## Objectives

To develop some Python functions that manipulate digital images.

## Demo/Grading

When you have finished all the exercises, call a TA, who will review your solutions, ask you to demonstrate some of them, and assign a grade. For those who don't finish early, a TA will grade the work you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

## Prerequisite

Complete the Lab 4 prelab exercises before attempting this lab.

## Getting Started

**Step 1:** Create a new folder named Lab 4 on the lab computer's hard disk (on the desktop or in your account's Documents folder) or in the M: drive on the network server.

**Step 2:** Download Cimpl.py and the image (JPEG) files from cuLearn to your Lab 4 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. If another version is running, ask a TA for help to reconfigure Wing to run Python 3.6.

**Step 4**: Click the New button in the menu bar. A new editor window, labelled untitled-1.py, will appear.

**Step 5:** From the menu bar, select File > Save As... A "Save Document As" dialogue box will appear. Navigate to your Lab 4 folder, then type lab4.py in the File name: box. Click the Save button. Wing will create a new module named lab4.py.

## 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**

A colour in the RGB colour model can be represented by a triplet of integer components, each of which ranges from 0 to 255. The first component is the quantity of red, the second component is the quantity of green, and the third component is the quantity of blue.

Each pixel in a display screen has three light sources placed close together, so we perceive them as a single "dot" in the display. One source emits red light, another emits green light and the third emits blue light. When a pixel is illuminated, the display hardware uses the RGB triplet for that pixel to set the intensity of light that is emitted from each of the three sources. Our vision system interprets the intensities of the red, green and blue light as one of the 16,777,216 colours that can be represented by an RGB triplet.

We can think of an RGB image as being composed of three overlapping monochromatic images (one consisting of shades of red, another consisting of shades of green, and the third consisting of shades of blue). These monochromatic images are often referred to as *channels*. In this exercise, you'll build filters that reveal the separate red, green and blue channels in an image.

**Step 1:** In `lab4.py`, define a filter named `red_channel`. The function header is:

```
def red_channel(image):
```

This filter has one parameter, `image`, which will refer to the image that the function will modify. This function should set the green and blue components of each pixel to 0, leaving the red component unchanged.

**Step 2:** Use the shell to test `red_channel`. To do this:

- Save the edited file, then click the `Run` button. (If you forget to do this, you won't be able to call `red_channel` from the shell, because you haven't saved the modified module and loaded it into the Python interpreter.)
- Notice that clicking `Run` resets the shell, which means that all variables and objects created in previous shell sessions are lost. You'll have to choose an image file and load the image before calling `red_channel`. Remember to bind the `Image` object returned by `load_image` to a variable.
- Call `red_channel`.
- After `red_channel` returns, display the modified image. What colours are in the modified image?

**Step 3:** In `lab4.py`, define a function named `green_channel` that is passed an image. This function should set the red and blue components of each pixel to 0, leaving the green component unchanged.

**Step 4:** Use the shell to test `green_channel`; that is, load an image file, call `green_channel` to modify the image, and display the modified image. What colours are in the modified image?

**Step 5:** In `lab4.py`, define a function named `blue_channel` that is passed an image. This function should set the red and green components of each pixel to 0, leaving the blue component unchanged.

**Step 6:** Use the shell to test `blue_channel`. What colours are in the modified image?

**Exercise 2**

A simple way to reduce an image's brightness by some percentage is to reduce every pixel's red, green and blue components by that percentage.

**Step 1:** In `lab4.py`, define a function named `reduce_brightness` that is passed an image as an argument. The function header is:

```
def reduce_brightness(image):
```

This function should reduce the image's brightness by 50% by setting the red, green and blue components of every pixel to half their current values.

**Step 2:** Use the shell to test `reduce_brightness`.

**Step 3:** Modify `reduce_brightness` so that it reduces the image's brightness to 75% of its original value.

**Step 4:** Use the shell to test the function. The modified image should be darker than the original image, but brighter than the image produced in Steps 1 and 2.

**Step 5:** Modify `reduce_brightness` so that it reduces the image's brightness to 25% of its original value.

**Step 6:** Use the shell to test the function. The modified image should be darker than the the images produced in Steps 1-4.

**Step 7:** Editing `reduce_brightness` every time we want to change the extent to which an image's brightness is reduced, is tedious. We should modify the function to take a second argument, a real number multiplier between 0.0 and 1.0, which will be applied to every component of every pixel.

Change the function header to:

```
def reduce_brightness(image, multiplier):
```

To reduce an image's brightness by 50%, we call the function this way:

```
reduce_brightness(image, 0.5)
```

To reduce an image's brightness to 25% of its original value, we call the function this way:

```
reduce_brightness(image, 0.25)
```

Make the required changes to the function's docstring and body. Use the shell to test the function.

**Exercise 3**

In `lab4.py`, define a function named `swap_red_blue` that is passed an image as an argument. This function should swap each pixel's red and blue components. For example, if a pixel's red and blue components are 127 and 41, respectively, then the red and blue components should be set to 41 and 127, respectively.

Use the shell to test `swap_red_blue`.

**Exercise 4**

We can design filters that hide an original image by distorting the red, green and blue components. Here is one approach:

For every pixel:

- calculate the average value of the pixel's red, green and blue components. Then, replace the red component with this average, divided by 10. For example, if the average value of a pixel's red, green and blue components is 190, change the red component to 19.
- replace the green component with a random integer in the range 0 to 255, inclusive. Do the same thing with the blue component. Use a new pair of random integers for every pixel.

  The easiest way to generate a random integer is to call the `randint` function in Python's `random` module. For example:

  ```
  import random

  i = random.randint(a, b)
  ```

  `randint` returns a random integer `n` such that `a` <= `n` <= `b`.

In `lab4.py`, define a function named `hide_image` that is passed an image and hides it, using the approach described here. All the steps can be placed inside a single `for` loop; that is, your function doesn't need to iterate over the image several times.

The green and blue "snow" (the random component values) in the modified image will obscure the original image.

Use the shell to test `hide_image`.

**Exercise 5**

In `lab4.py`, define a function named `recover_image` that is passed an image that was prepared by your `hide_image` filter. The function header is:

```
    def recover_image(image):
```

This function recovers an approximation of the original image. (The recovered image won't be identical to the original one - the range of colours will be different - but it will be good enough for you to recognize the original image.)

Hint: Fixing the red components is straightforward, but how should the function fix the green and blue components? Remember that all of the original green and blue components were lost: they were replaced by random values. We need to replace these random values with a known value, but what value should we use? 255? 128? 0? Another value? You'll need to experiment to discover which approach yields a recovered image that is the best approximation of the original one.

**Wrap-up**

1. Remember to have a TA review your solutions to Exercises 1-5, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the attendance/grading sheet.
2. Remember to backup your project folder before you leave the lab; for example, copy it to a flash drive and/or a cloud-based file storage service.