Problem Set 5: Using Libraries

Due: October 12th, 5pm
Checkoff due: October 19th
You can receive a checkoff any time after submitting your code.

The problem set will introduce you to the topic of library use, that is, using existing libraries in order to accomplish a goal. This problem set involves writing very little code; the learning process is finding useful functions that do the work for you in the existing libraries. You are expected to make use of web search to locate the necessary functions.

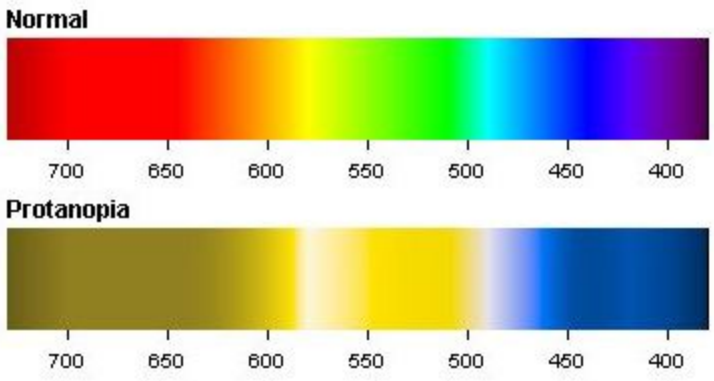
Note on Collaboration:
You may work with other students. However, each student should write up and hand in his or her assignment separately. Be sure to indicate with whom you have worked in the comments of your submission.

Note on Grading:
The grade from this problem set will be primarily based off of the check-off; **there will be no auto-graded portion.** Take into account the fact that the check-offs for this problems set will require more time than the previous ones. Please be sure to leave adequate time to get your check-off and keep in mind that that the queue is often much more busy in the final days of the check-off period. Plan accordingly. This assignment will be worth 10 points in total.

Problem 1: Colorblindness Filters

The human eye contains two type of photoreceptive (light sensitive) cells: rods and cones. Rods are responsible for vision in low light environments, such as at night, and cones are responsible for color vision. There are three types of cones, each responsible for a particular color, that is, red, green and blue. The red, green and blue cones all work together allowing you to see the whole spectrum of colors. For example, when the red and blue cones are stimulated in a certain way, you will see the color purple.

The condition known as colorblindness typically manifests as a deficiency in one of these types of cones, for example, protanopia is a lack of sensitivity to red light, so the following is an example of the difference in viewing an image.



The objective of this part is to create filters to simulate these differences. To do this, we will make use of Python's Image Library (https://pillow.readthedocs.io/en/latest/reference/Image.html) or PIL for short. From PIL we will import Image, which is a sublibrary, or a set of functions and methods, related to image processing. With Image, we can convert images to a list of pixels, and manipulate the RGB (red, green, blue) values of these pixels.

We can replicate the effects of colorblindness with a matrix multiplication between a colorblindness transformation matrix and the RGB values of a particular pixel, which are represented as a vector with 3 entries.

You do not need to know anything about matrix multiplication; however, if you wish to pursue more information on it, see here: https://en.wikipedia.org/wiki/Matrix_multiplication

Here is a general breakdown of the image transformation process:

1. Open the image in python
2. Retrieve a list (of tuples) of the pixel information
3. Iterate through pixels and multiply them by the transformation matrix using **matrix_multiply**
4. Save transformed pixels as an image

For a general description of the Image class with **many** helpful methods and descriptions, refer to: http://effbot.org/imagingbook/image.htm#tag-Image.Image

Helper Functions (Implement these!)

a. convert_image_to_pixels(image)

The first helper function will be to convert the image to a list of pixels. The input is a string representing an image file, like 'example.jpg'. The output is a list corresponding to the pixels in that image ex: [(0,0,0),(255,255,255),(38,29,58)...]. The list will consist of 3-element tuples that correspond to the RGB values of that pixel.

You want to open the image (check the documentation on how to do this), and get the data of that image.

This should be **very short** if you can find the appropriate functions.

b. convert_pixels_to_image(pixels,size)

The next helper function is to convert a list of pixels to an image. The input will be pixels, in the same format as the output of the previous function, and a size parameter which is a two-element tuple that describes the dimensions of the output image. Assume that size is a valid input such that size[0] * size[1] == len(pixels). In other words, you want to **copy pixel values from a sequence object into the image**, so you should check the Image module documentation on how to do this. If you wish to use the **Image.new** method, keep in mind that the mode is 'RGB'.

This should be **very short** if you can find the appropriate functions.

c. apply_filter(pixels, color)

This function takes in a list of pixels in RGB form, such as [(0,0,0),(255,255,255),(38,29,58)...], as well as a color: 'red', 'blue', 'green', or 'none'. The purpose of this function is to apply a transformation to the pixels in the input list that simulate impairment in the cones of the input color. Return the list of transformed pixels. In order to do the transformation, multiply each pixel by the appropriate matrix. We have provided a **generate_matrix** function that will supply a matrix representing the appropriate transformation depending on the input string. For example, generate_matrix('red') will return the matrix representing a red deficiency in one's vision.

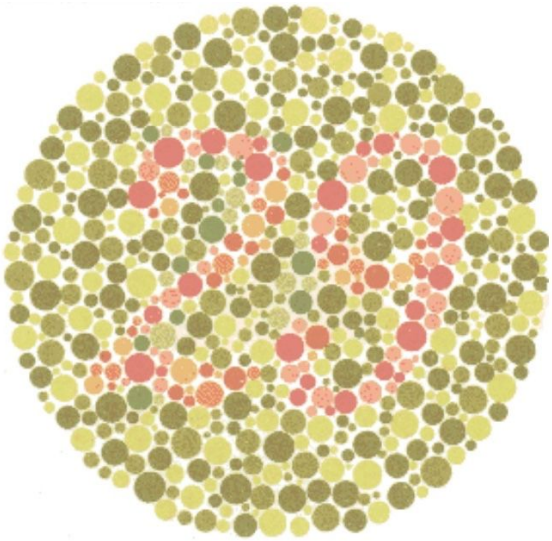
Remember that you must **iterate through the list of pixels and multiply each one by the appropriate matrix**. Use the provided **matrix_multiply(matrix1, matrix2)** method

to do this. Please keep in mind that we want the **transformation matrix to be the first argument and the RGB pixel vector to be the second**, or else the resulting image will be incorrect. Take note that **matrix_multiply** returns a **list of floats** and that the pixels **must be tuples of ints**.

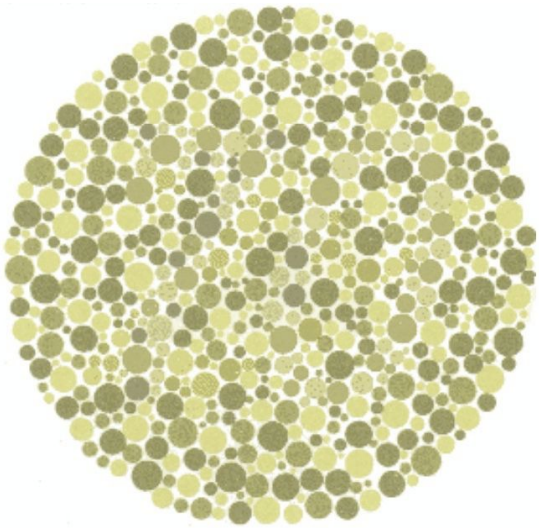
This should be **very short** if you can find the appropriate functions.

In order to check your code, uncomment the 8 lines in the main() function at the bottom of the file. If all is done correctly, then running the program with the test image ('img_29.jpg') and a red deficiency should result in the following:

The original image (you see 29):



The transformed image (you see no numbers):



Images like these are known as Ishihara tests. To receive credit, have these images visible on your computer for your TA/LA to check during your check-off.

Problem 2: Hidden messages in images

In an image, it is possible to hide a second (secret) image that can only be recovered by certain mathematical operations. This is known as *steganography*. In this section, you will go through some examples and write your own steganography module to hide images.

Original Image

Secret Image

An example of steganography is provided above. The image on the left contains the image on the right. When you are done with this problem set, you will understand how to extract the secret image from the original.

For this part, assume you are dealing with images that have 8-bit color depth. In other words, every pixel value is between 0 and 255.

a. Recovering a binary and color image

The most common technique for embedding secret images is to use the least significant bit (LSB) of each pixel value. We can modify the LSB in each pixel without making any noticeable difference. This way, an image can carry another secret image that is completely invisible to the naked eye.

Binary Numbers Review:

Computers represent numbers using **binary digits**. A binary digit can only be **0** or **1**. A "binary digit" is often shortened to the word **"bit"**. For a refresher on representing numbers in binary, please refer to https://www.mathsisfun.com/binary-digits.html.

Can you figure out why the number 43 is represented as 101011 in binary? What is the least significant bit?

What is the least significant bit of the number 44? What is the least significant bit of the number 45? Can you see a pattern?

In an image, pixels are represented by numerical values that indicate brightness. The minimum value for each pixel component is 0 (none of the color appears in the pixel), while the maximum value (highest level of the color appears in the pixel) depends on the number of bits used for each pixel. In our examples, we will use 8-bit images. This means the pixels can take values between 0 and 255, inclusive (can you see why?).

In **black and white images**, each pixel is represented by a single integer between 0 and 255. A pixel that is black has a value of 0, and a pixel that is white has a value of 255. Values in between represent different shades of gray.

In **RGB images**, each pixel is represented by a vector of three integers, each with a value between 0 and 255. The first value corresponds to how much red is represented in the pixel, the second value corresponds to how much green is represented in the pixel, and the third value corresponds to how much blue is represented in the pixel. For example, a pixel that is red is represented by (255, 0, 0).

In ps5.py, complete the functions **get_BW_lsb** and **get_RGB_lsb.** Both of these functions take as input a list of pixels (BW images are represented by a list of ints; RGB images are represented by a list of tuples of 3 ints).

b. Revealing an image

The reveal_image function takes in a filename and a mode. The filename is the name of the file in which there is a hidden image. The mode (as described in the docstring) is 'RGB' for a color image or '1' for a black and white image. This function should make a call to either get_BW_lsb or get_RGB_lsb depending on whether a black and white or color image is passed in. It should then use the list of least significant bits returned by get_BW_lsb or get_RGB_lsb to construct the hidden image and return an Image object.

Important: The hidden image is embedded in the least significant bit and can be recovered with a straightforward arithmetic operation on each pixel value. However, using the raw result from this operation will produce an image with very low contrast. Once you recover the information, you should carry out another operation to rescale the pixel values, so they can make use of the full range of available values. Note that for RGB images, you will need to rescale all 3 integers in each tuple.

Reveal black/white image:

In the problem set folder, you will find an image named **hidden1.bmp**. Using your function, find the secret image in this file and save a copy of it (by a screenshot or by using the library). Since we use a grayscale image, the secret image should also be grayscale.

Reveal color image:

In the problem set folder, you will find an image named **hidden2.bmp**. Using your function, find the secret image in this file and save a copy of it (by a screenshot or by using the library).

This completes the problem set!

Hand-in Procedure

1. Save
Save your solution file with the name that was provided: **ps5.py**. **Do not ignore this step and do not save your file with a different name! This will lead to extended delays in your grading.**

2. Time and collaboration info
At the start of each file, in a comment, write down the number of hours (roughly) you spent on this problem set, and the names of whomever you collaborated with. For example:

```
# Problem Set 5
# Name: Alyssa P. Hacker
# Collaborators: Ben Bitdiddle
# Time: 3.14 hours
#
... your code goes here ...
```

3. Submit

To submit a file, upload it to the Problem Set 5 submission page. You may upload new versions of the file until the 5:00pm deadline, but anything uploaded after that time will be counted towards your late days, if you have any remaining. If you have no remaining late days, you will receive no credit for a late submission.

You do not need to submit the images you produced but you will need to show the images or run your code to produce each image during your checkoff.