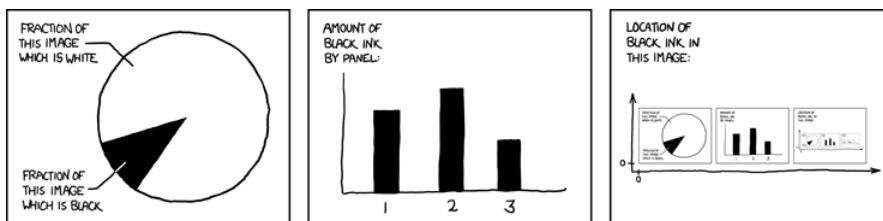


CS51 - Assignment 8

Modifying PPM Files

Due: April 9th, at 11:59pm



<https://xkcd.com/688/>

For this assignment, we will:

- Gain more practice working with lists.
- Write a program that manipulates image files.

1 Before you come to lab

Read the entire handout and bring your answers on a piece of paper with your name on.

1. Draw a picture of what the following PPM image would look like.

```
P3
3 4
255
0 0 0 255 255 255 0 0 0
255 255 255 255 255 255 255 255 255
0 0 0 255 255 255 0 0 0
255 255 255 0 0 0 255 0 0
```

2. If we “negated” all of the pixels in this file, what would the image look like?
3. What would be the “grey” value of the pixel in the lower right corner?
4. Write one test case to test `remove_color` (that is different than the example we list below). The test case should state what the input is and what the expected output should be.

2 Getting started

Create a new project, `assignment8`, using VS Code in your CS51 workspace and set it up so that you can run Python. Double-check that you are creating the project in the correct location, as you will likely have trouble finding your files later. Then, download the starter code and unzip the file (on a Mac, unzipping happens automatically when you double-click the file).

You should see a folder named `starter` that contains two files (`assignment8.py` and `assignment8_tester.py`) and one subfolder named `files`. Copy the two Python files and the subfolder into your `assignment8` folder.

3 Background: PPM files

The acronym ppm stands for “Portable Pixel Map,” which is a text format for storing images. It is incredibly inefficient: an image stored in ppm format will be much larger than the same image stored in, say, jpeg format. However, it is also relatively easy to read and write files in ppm format, which is why we’re using it for this assignment.

The first three lines in a (basic) ppm file are called the header. They will look like this:

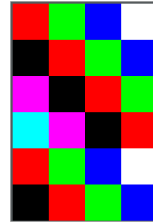
```
P3
4 6
255
```

The first line specifies the encoding. In this assignment, the P3 encoding will always be used. The second line specifies the width and height of the image in pixels: this image will be 4 pixels wide and 6 pixels tall. Finally, the third line specifies the maximum value for the red, green, and blue values. In this assignment, the value will always be 255.

After the header comes the body. As we saw with the last assignment, we can represent any color by listing what proportion of red, green, and blue (RGB) it represents. For the PPM format, we specify what color each pixel (i.e., point) in the image is by three RGB values. The image itself, then, is a combination of `r` rows (the height of the image) and `c` columns (the width of the image). The body will therefore contain $3 * r * c$ integers, each between 0 and 255 (inclusive). Each number will be separated by at least one whitespace character. Given that each pixel holds RGB values the number of integers on each line will be $3 * c$, representing each line of the image. For example, our file might contain the following:

```
255 0 0 0 255 0 0 0 255 255 255 255
0 0 0 255 0 0 0 255 0 0 0 255
255 0 255 0 0 0 255 0 0 0 255 0
0 255 255 255 0 255 0 0 0 255 0 0
255 0 0 0 255 0 0 0 255 255 255 255
0 0 0 255 0 0 0 255 0 0 0 255
```

The first three numbers provide the RGB values for the pixel in the upper left corner of the picture. In this case, the pixel in the upper left corner of the picture has a red value of 255, a green value of 0, and a blue value of 0 (i.e., that pixel is pure red). The next three numbers on that line provide the RGB values for the 2nd pixel on the first row. And so on, pixel by pixel. The pixel in the lower right corner of the sample picture corresponds to the last three numbers, which have a red value of 0, a green value of 0, and a blue value of 255 (i.e., it is pure blue). Note that the line breaks in the file don't necessarily correspond to the end of a row of pixels in the image; the only requirement is that the file must contain the correct total number of integers after the header, each separated by whitespace.



As a result, the image looks as follows (only much smaller):

The `files` directory contains a number of sample PPM files. If you open them in VS Code, you can see the raw file format (i.e., RGB values). If you double click on the PPM files from your file browser, it *should* open them as images and you can see what they look like!

Important: Make sure you understand the specification for the ppm format before continuing! If you aren't sure, ask!

4 Part 1: Warming Up

Implement a function `decode` that takes two strings as parameters (`in_filename` and `out_filename` representing PPM files). The function should read from `in_filename` and create a new file named `out_filename` that satisfies the following properties:

The file named `out_filename` will be a ppm file. `out_filename` will have the same header as `in_filename`. The body of `out_filename` will be generated from the body of `in_filename` line-by-line using the following rules:

- If a number modulo 3 is equal to 0, it will be replaced by the number 0.
- If a number modulo 3 is equal to 1, it will be replaced by the number 153.
- If a number modulo 3 is equal to 2, it will be replaced by the number 255.

You can test your function, by running it on one of the test images. For example:

```
>>> decode('files/part1.ppm', 'files/part1.decoded.ppm')
```

After running this, you can look at the file `part1.decoded.ppm` in the `files` directory to see the result. If you've done everything correctly, you should be in for a nice surprise :).

5 Part 2

For Part 2, you will implement an image processing program. Your code must contain the following four functions:

5.1 Function 1: `negate(line)`

We can “*negate*” a pixel’s values by taking each of its RGB values and subtracting them (individually) from 255. For example, if we negate white (255 255 255) we get black (0 0 0) (and vice versa).

Fill in the function `negate`, which negates all of the values on a line. The function takes a `line` (of type `str`) as input. The input parameter `line` is guaranteed to contain a sequence of integers, each with a value between 0 and 255 (inclusive), separated by whitespace. In addition, the number of integers will be a multiple of three. The function returns a string with the same number of values, but with each one of them negated. For example, if the first value in `line` was 155, then the first value in the returned string should be 100 (since $100 = 255 - 155$). Another example can be:

```
negate('1 2 3 200 100 150')
```

should return the string

```
negate('254 253 252 55 155 105')
```

When you believe your function is correct, add test cases to `assignment8_tester.py` to thoroughly test your implementation.

5.2 Function 2: `grey_scale(line)`

This function takes a single parameter `line` (of type `str`) as input. The parameter `line` is guaranteed to contain a sequence of integers, each with a value between 0 and 255 (inclusive), separated by whitespace. In addition, the number of integers will be a multiple of three. The function returns a string with the same number of values. However, each set of three RGB values is replaced by three, identical values, turning the color into a shade of grey. To mathematical formula to calculate the grey value for a set of RGB values, the formula is:

```
grey = int(sqrt(r**2+g**2+b**2))
```

Note that shades of grey are exactly those pixels that have equal RGB values, so you should set all three of those equal to the computed grey value for a pixel. Below is an example of the `grey_scale` function:

```
grey_scale('1 2 3 200 100 150')
```

should return the string

```
'3 3 3 255 255 255'
```

Note that the formula above can result in values that are greater than 255, which is not valid. Any grey value greater than 255 should be set to 255. The `sqrt` function is in the `math` module, which you can import using:

```
from math import sqrt
```

When you believe your function is correct, add test cases to `assignment8_tester.py` to thoroughly test your implementation.

5.3 Function 3: `remove_color(line, color)`

This function takes two parameters as input. The first parameter, `line`, is a string, which is a valid line of numbers, defined in the same way as for the previous two functions. The second parameter, `color`, is also a string but is required to be one of the strings `'red'`, `'green'`, or `'blue'`. You may assume that the arguments it gets are valid.

If, for example, the `color` is `'red'`, then every red component should be set to 0 in the output string while the green and blue components remain the same. Similarly, if the parameter passed is `'blue'`, then every blue component should be set to 0.

As an example:

```
remove_color('1 2 3 200 100 150', 'red')
```

should return the string

```
'0 2 3 0 100 150'
```

When you believe your function is correct, add test cases to `assignment8_tester.py` to thoroughly test your implementation.

5.4 Putting it all together

Write a function called `modify_ppm` that takes three inputs:

- an input ppm file (as a `str`)

- an output ppm file (as a `str`)
- an integer between 1 and 5 (inclusive) indicating what image modification to perform

The integer operations are as follows:

1. negate
2. greyscale
3. remove red
4. remove green
5. remove blue

For example, if we ran the following:

```
>>> modify_ppm('files/small.ppm', 'files/small.grayscale.ppm', 2)
```

the function would generate a file called `files/small.grayscale.ppm` that would be the input file with greyscale applied.

If an invalid integer is input, it should print out an error message: “Invalid modification”.

6 Details, Hints, and Suggestions

Details, Hints, and Suggestions

- Incremental development and testing will be helpful. We strongly suggest that you start by writing docstrings for each function and then write and test them one at a time.
- The `files` directory contains a number of PPM files for you to experiment with. While you’re debugging your code, we suggest you use `files/small.ppm` to test your function. After it works, you can continue to use the large files, e.g., `files/pomona.ppm`, to perform more testing. This should save you some time!
- There are some test files, including the 4-by-6 image from the Background section, available in the `files` subfolder. All of these files satisfy the property that each row of the body contains a multiple of three numbers (that is, no pixels are split across lines). Remember that you will have to tell Python where the files are located (e.g., to read from the file `small.ppm` in subfolder `files`, use the filename `files/small.ppm`)
- If you open a `.ppm` file in VS Code, it will show you the contents of it. If you want to see what the image looks like, open it with the default image viewer provided by your operating system.
- Make sure your parameter and return types match the specification!

7 Feedback

Create a file named `feedback8.txt` that answers the questions:

- How long did you spend on this assignment?
- Any comments or feedback? What things did you find interesting, challenging, or boring?

8 When you're done

Submit your `assignment8.py` and `assignment8_tester.py`, and `feedback8.txt` online using Gradescope. Be sure you're uploading them to the right assignment. Note that you can resubmit the same assignment as many times as you would like up until the deadline.

9 Grading

We reserve the right to remove points for not adhering to the Style Guide.

	points
Warm-up	1
<code>decode</code>	1
<code>negate</code> execution and testing	1
<code>grey_scale</code> execution and testing	1.5
<code>remove_color</code> execution and testing	1.5
<code>modify_ppm</code>	2
testing	2
total	10