Homework 6 : PPM Image Processing

Overview

The goal of this assignment is to give you practice working with lists by writing programs that manipulate files in various ways. To make this interesting, the lists you manipulate will represent images and the processing you perform will alter these images in interesting ways.

Deliverables

Part	Section	Submission
1	Planning Questions	None
	Decoding a PPM file	check-off in lab & github
2	Modifying PPM images	github
	Testing	github

Part 1: PPM Files and Modifications

Getting started

You will create new directory called Assignment6 inside your repository. The Assignment6 folder should contain two files (`ppm_modify.py` and `test_ppm_modify.py`) and a sub-directory called `files`. The `files` directory contains the sample image files.

Part 1: Planning Questions

When you read through the programming assignment in Part 1, please keep in mind the following questions and try to figure out the answers for them. These questions will help you plan and design your solutions to the coding problems in the lab and assignment.

1. In a PPM file, if the second line of the header is `4 6`, how many integer values will be there be the body of the PPM file? why?
2. Given a line of RGB values `1 2 3 200 100 150 4 5 6`, which of these values represent red?
3. A program can read the PPM file line by line. When a line is read, what process do you need to perform in order for you to access and manipulate the RGB values that are separated by whitespace (the ASCII whitespace characters include space, tab, linefeed, return, formfeed, and vertical tab?

Decoding a PPM file

Your task for Part 1 is to implement a decoder (a function) in `ppm_modify.py` and then use that on the file `files/part1.ppm` to decode an image.

Background: PPM files

The acronym ppm stands for "Portable Pixel Map" and 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 something like this:

```
P3
4 6
255
```

The first line specifies the encoding. In this assignment, it will always be P3. The second line specifies the width and height of the image in pixels. In other words, this image will be 4 pixels wide and 6 pixels tall (look at the sample image Figure-1 on the next page). 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*. If there are `r` rows and `c` columns, then the body will contain `3 * r * c` integers, each between 0 and 255 (inclusive). Each number will be separated by at least one whitespace character (this means that when you read the file, you would have to account for multiple spaces when separating the values and not just single space). The number of integers on each line will be a multiple 3. 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 3 numbers give the r, g, b values for the pixel in the upper left hand corner of the picture. In this case the pixel in the upper left hand 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 3 numbers give the r, g, b values for the pixel to the right (pure green). And so on, pixel by pixel. The pixel in the lower right hand corner of the sample picture corresponds to the last three numbers, so it has 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 numbers after the header, each separated by whitespace.

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

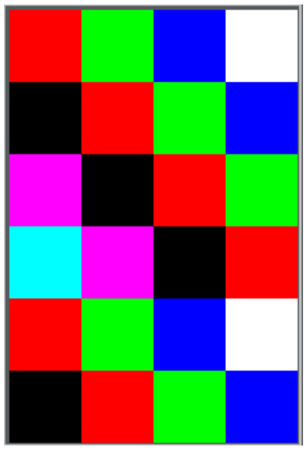

Figure 1: test

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

Hint There are several ppm files in the `files` directory. You should be able to view these with almost any image viewer. If necessary, you can use this on-line viewer (https://www.cs.rhodes.edu/welshc/COMP141_F16/ppmReader.html)

Background lists

In class you were introduced to lists. A natural way to process PPM files is to read the files one line at a time and to split the line into elements. An important python tool for performing this with strings is `split()`. Here is a small demonstration:

```
>>> s = "1 2 33 44 stuff\n"
>>> l = s.split()
>>> l
['1', '2', '33', '44', 'stuff']
```

Remember that you can iterate with lists in the same ways that you can iterate with strings:

```
>>> l = ['1','2','3','4']
>>> for e in l:
...     print(int(e))
...
1
2
3
```

```
4
>>> for i in range(len(l)):
...     print(int(l[i]))
...
1
2
3
4
```

Implementing the Decoder

The decoder is implemented as two functions – a "filter" called `color_translate` and a wrapper function called `ppm_process`. The `color_translate` function takes as its input a "stripped" string (a string with the leading and trailing whitespace removed - you may want to use the strip() function from the last lab on file input and output) and returns a string where the input has been translated according to 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.

Keep in mind that the input is a string so you will have to extract the text representing individual numbers and convert that text to an integer in order to perform the processing.

Hint: Remember that the modulo operator is `%` in Python.

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

Processing

`ppm_process` takes as input two file names (really strings of file-paths) and the name of a processing function to apply. For decoding you will be using `color_translate` (that you already implemented) as the processing function.

1. The file named `out_filename` will be a ppm file.
2. `out_filename` will have the same header as `in_filename`.
3. The body of `out_filename` will be generated from the body of `in_filename` line-by-line using the processing function `color_translate`

Decoding part1.ppm

In `main_part1`, run your function `ppm_process` on the file `files/part1.ppm` using `color_translate` as the filter. The output file should be `files/part1_output.ppm`. Please note that `main_part1()` (to call the decoder function) function is different from the normal `main()` function (to handle file input/output from the user). The starter code contains comments to help with distinguishing the two functions.

Hint: You might want to start by writing a filter that just returns its input unmodified and hence copies `in_filename` to `out_filename`.

Checking In

You must complete this part of the assignment before lab and check-in during lab. Make sure your part 1 code satisfies good style (including a docstring) before checking in! **If you are ill and cannot attend lab you may attend one of the virtual office hours on Thursday or Friday to check off your work**

Part 2: Modifying a PPM File

In the following you will implement an image processing "filter" called `grey_scale`.

grey_scale(line)

This function takes a single parameter `line` (of type `str`). The parameter `line` is guaranteed to contain a sequence of integers, each with 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 r, g, b values is replaced by three grey values. The formula is:

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

Recall that shades of grey are exactly those where the r, g, and b values are all equal, so you should set all three of those equal to the computed grey value.

Also keep in mind that each value should be an integer and that you'll need to make sure that each value is no more than 255 (so anything greater than 255 should be set equal to 255). Finally, the `sqrt` function is contained in the math package so you will want to import that.

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

main()

To put this all together, your program's `main` function should:

- Ask the user for an input file.
- Ask the user for an output file.
- Perform the grey_scale manipulation on the input file and write the result to the output file in ppm format (don't forget to write out the header information!).

Sample Run

```
input file name:
    test.ppm
output file name:
    test_out.ppm
done
```