Problem Solving using Python - Week 5 - Homework

Pipeline Design, String Manipulations, Evaluation

Part I - One Liners

In the python file `one_liners.py` you will find multiple documented functions. You task is to implement all these functions by using **only one line of code**. You are not allowed to use the semicolon (`;`) to put multiple python statements in one line.

Hint: list/dict/set comprehensions.

You can test your functions by running this command from the terminal/command line (not from Python shell!):

```
python ok -q <function>
```

Where `<function>` is the function name to test from the file `one_liners.py`.
For example the command `python ok -q unicode_codes` will test the first function in the file: `unicode_codes`.

Part II - Image Editor

In this part you are going to develop an image editor for the PPM image format. Your program will be able to apply powerful image filters and save the transformed image to a file. First we will describe the PPM image format, and then we will give a detailed description of you tasks.

PPM Image Format

The PPM (or Portable Pix Map) is a way to encode images as human-readable ASCII text. For those of you who wish to gain experience reading real documentation, the full image specification can be found here.

Sample ppm file:

```
P3
4 4
255
0  0  0   100 0  0       0  0  0    255   0 255
0  0  0    0 255 175     0  0  0     0    0  0
```

```
0  0  0    0  0  0       0 15 175    0    0  0
255 0 255  0  0  0       0  0  0    255  255 255
```

Image Header

You can think of the image as having two parts, a **header** and a **body**. The **header** consists of four entries:

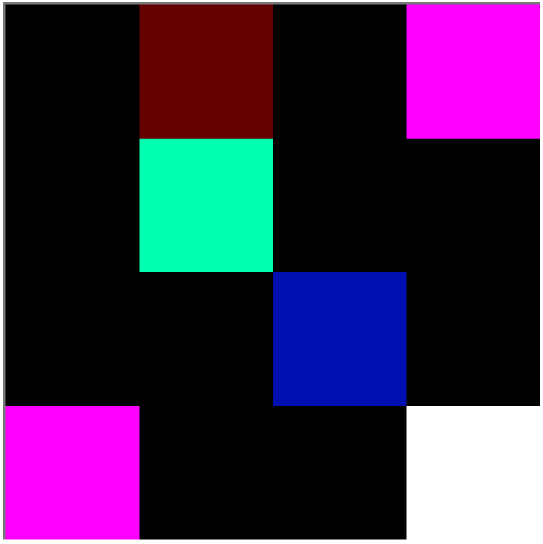
P3 is a **"magic number"**. It indicates what type of PPM (full color, ASCII encoding) image this is. For this assignment it will always be P3.
Next comes the number of columns and the number of rows in the image (4 x 4).
Finally, we have the maximum color value `255`. This can be any value, but a common value is 255.
The way you see the header presented is how it should be spaced out.

Image Body

The **image body** contains the actual picture information. Each pixel of the image is a tiny colored square. The color is determined by how much red, green, and blue is present. So, `0 0 0` is the first color of the image, which is black, and the last pixel in the image is `255 255 255`, which is white. By varying the levels of the RGB values you can come up with any color in between.
Note that color values must be separated by a space, but after than additional whitespace is ignored by the image viewer. In the sample ppm above we used additional whitespace to format the image so that it is easy for a human to understand, but the computer doesn't care if everything is on one line, or if there is one line per line of the image, or some mix (we will keep 1 line of text = 1 line of the image for this assignment to make the images easier to read in a text editor).

Putting it all together

The example image above would look something like this:



Keep in mind, each square is one pixel, so the real thing is much smaller (the rendered image was blown up by 5000%).

How to view PPM files

While PPM files are easy to view as text (you can use Notepad, for instance), and easy to work with in code, they are highly inefficient. Most modern image formats use some kind of compression to make their size reasonable while preserving the image appearance. This is not to say that PPMs don't still have some life in them – one modern use for PPM is an intermediate format when converting images from one type to another.

You will need to install a program to view these images on a Windows machine. We find that Irfanview is a small download and works quite well. It will also allow you to convert your own images to PPM so you can practice with pictures you took in the past (keep in mind that you may need to make them very small or the resulting PPM will be **quite large**!).

Your Assignment

Whew! That may sound like a lot of work, but it's really pretty simple. To make it easy to code up we've broken the program into three phases. In the first you are reading the whole image and writing it to a file without making any changes to the image in the process. Believe it or not, at this point you've written most of the program! In phase II we add some effects, and in phase III we add a menu for the user.

Guidelines

- Use the *Programming Problem Solving Model*.
- Your final delivery is a *command line program* and not a Jupyter Notebook. In other words, your solution should be a Python file `image_editor.py`, which is run from the command line.
- We encourage you to use the *incremental development* methodology. You can develop the Image Editor in a Jupyter Notebook, and at the end just copy the code into a new file (as we did with the "Subtitles Synchronization" problem at the lecture). **Make sure that you test also the complete Python file that you submit**.

Phase I:

The user of your program should be able to specify the name of the image file. The file will be a text file in PPM format as described in the discussion above.

Bonus: If the image file cannot be opened, the program should report that fact and abort. We have not learnt how to do this in class yet. Therefore you can use the Internet and Python's documentation to figure out how to do that.

The user will specify an output filename. The purpose of the program is to make an exact copy of the input file and save it as the output file. The "catch" is that you are limited to read and write only one ROW of the picture each time. This is not an unrealistic restriction. A picture of a normal size could easily become hundreds of thousands of numbers. You will not be able to read all of the image into memory at one time. So you have to deal with one piece of the file at a time.

Another Hint: this is not a problem in string processing! Note that many of the functions involve doing arithmetic with the pixels. You cannot do that with strings!

Your program is going to have to read in one row, write it out to the output file, and repeat the process until the input is exhausted.

Your output file can actually be formatted as you like. Whitespace includes newline characters, so you can put them in where you wish. The format allows for it.

Example interaction with the user:

```
Portable Pixmap (PPM) Image Editor!

Enter name of image file:  feep.ppm
Enter name of output file: out.ppm
out.ppm created.
```

And the output file created would be in a file called `out.ppm` and would be identical to `feep.ppm`. Your program is NOT responsible for displaying the image in the file, just for manipulating the pixels and creating an output file in the proper PPM format.

Test this with small files and large files. You can check to see if they are identical by loading them both into Notepad / Jupyter Notebook or another text editor, and comparing number by number.

Example ppm files can be downloaded as a ZIP Archive (1.2 MB) containing the following images:

- cake.ppm - A picture of a slice of cake on a plate
- squares.ppm - Some boxes
- blocks.ppm - Some solid blocks of color
- tinypix.ppm - The example image from above

Pre-Phase II - Design Approval

Before you start to code your solution to phase II, you should make its design, i.e., apply the second step in the *Programming Problem Solving Model*. Its format may be text (plain English or pseudo-code), bullet points or even a drawing. Pay extra attention to which data structures you plan to use. Send us your design in a private message in Piazza, and we will review it within 24 hours. **Do not start to code your solution to phase II before we've approved your design**. Therefore, we strongly recommend doing this early so as to give yourself time to correct your design after our feedback.

Phase II

Write a function called `negate_red`. It will change just the RED color numbers into their "negative". That is, if the red number is low, it should become high and vice versa. The maximum color depth number is useful here. If the red is 0, it will become 255; if it is 255 it will become 0. If the red is 100, it will become 155.

When you have this function written, insert it into your code for Phase I so that every pixel of the picture has its red color negated in the output file.

View this picture - does it look as you expected? Write equivalent functions for the other colors (green and blue).

Write a function called `flip_horizontal` which will flip the picture horizontally. That is, the pixel that is on the far right of the row ends up on the far left of the row and vice versa (remember to preserve RGB order!).

Write a function called `grey_scale` which will change the picture into a grey scale image. This is done by averaging the values of all three color numbers for a pixel, the red, green and blue, and then replacing them all by that average. So if the three colors are 25, 75 and 250, the average would be 116, and all three numbers are set to 116.

Write a function called `flatten_red` which will set the red value to zero. Write equivalent functions for the other colors (green and blue).

In summary, you must implement the following functionality:

- `negate_red` - Its job is to negate the red number of each pixel.
- `negate_green`, as above but change the green
- `negate_blue`, as above but change the blue
- `flip_horizontal` that flips each row horizontally
- `grey_scale` sets each pixel value to the average of the three
- `flatten_red` sets the red value to zero
- `flatten_green` sets the green value to zero
- `flatten_blue` sets the blue value to zero

Functions Checklist

You can run a checklist on your code to validate you have implemented all the function of phase II. It **will not** check the correctness of your functions, only that they exist.

To do so, run `python ok -q phase-2-checklist`. Run this in the command line from the homework directory.

Phase III

The last part of the problem is to add a menu so that the user can decide which of these effects will be applied to an image.

Example of user interaction:

```
Portable Pixmap (PPM) Image Editor

Enter name of image file:  cake.ppm
Enter name of output file: outcake.ppm

Here are your choices:
[1]  convert to greyscale [2]  flip horizontally
```

```
[3]  negative of red [4]  negative of green [5]  negative of blue
[6]  just the reds   [7]  just the greens   [8]  just the blues

Do you want [1]? (y/n)  y
Do you want [2]? (y/n)  n
Do you want [3]? (y/n)  y
Do you want [4]? (y/n)  n
Do you want [5]? (y/n)  n
Do you want [6]? (y/n)  n
Do you want [7]? (y/n)  n
Do you want [8]? (y/n)  y

outcake.ppm created.
```

You should apply the desired effects to the image one at a time.

Testing:

Your input routine should avoid reading from a broken input stream. What if the file is not as large as the row and column numbers indicate it will be? Does your code handle that gracefully (without falling into infinite loops, for example)?

All your manipulations should NOT cause a color number to be less than 0 nor larger than the maximum color depth specified in the file.

Bonus:

Implement up to 2 of these other functions. Each will be worth up to 5 points.

- `horizontal_blur` which will take the values of the red numbers of three adjacent pixels and replace them with their average - note that this is different from grayscale! it also does the same with the greens and the blues of 3 adjacent pixels. Pixels on the edges will have to be handled specially.
- `extreme_contrast` which will change each color number to either the highest color number possible or to 0. This change is based on whether it is greater than the midpoint of the color range, or less. If it is greater than half of the color depth, replace it with the colordepth. If it is less, replace it with zero.
- `random_noise` adds a random number to each color number or subtracts a random number. It would have a parameter which would represent the size of the random number range; i.e. a

value of 10 would add or subtract numbers in the range of 0 to 9, a value of 50 would add or subtract numbers in the range 0 to 49. Another random number would decide whether it was an addition or a subtraction. Remember that the values in the buffer should not exceed the colordepth nor get less than 0. You can use the extreme values instead.

OK Tests

Important: The ok tests don't always tell you that your answer is correct. More often, they help catch careless mistakes. It's up to you to ensure that your answer is correct (Test Phase!). If you're not sure, ask someone (not for the answer, but for some guidance about your approach).

Submission

When you are done, submit with `python ok --submit`. Run this in the command line from the homework directory. You may submit more than once before the deadline; only the final submission will be scored.

Credit

Originally written by Joshua T. Guerin and Debby Keen from the University of Kentucky.