SHP Spring 2018
February 24th 2017 Assignment (Accelerated)

Intro to PPM Image Formats
The PPM (or Portable Pix Map) image format is encoded in human-readable ASCII text. When you open a ppm file using the default program, most likely a picture viewer will pop up. But if you specify to open it with a text editing program, such as NotePad or TextEdit, the same file looks like this:

```
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
4 4
255

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**).

- first number is number of columns, i.e. how many pixels are in each row

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. (without empty lines in b/t)

<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 are present. The sequence of 3 numbers (0 to 255 or some other max#) for each pixel shows the RGB value. (first number for red, second for green, third for blue) By varying the levels of the RGB values you can come up with any color in between.

e.g. **0 0 0** represents black
255 255 255 is white.

- Note that color values must be separated by a space, but additional whitespace is ignored by the image viewer.
- It counts the first 3 numbers as the RGB sequence for the first pixel, the second set of 3 numbers as the RGB sequence for the second pixel, etc. regardless of how much white space in between
- So it doesn't matter if everything is on one line (b/c the dimensions of pixels are already specified for the image viewer in the file header)
- The ppm files for this assignment have been written so that each row of pixels equals one line. Thus, if you read in one line using `.readline()`, you will have read all the RGB values for all pixels in one row

How to view PPM files

You may need to install a program to view the images for this assignment on your machine (especially for Windows). GIMP has worked very well for me: https://www.gimp.org/downloads/

- It seems ppm files are visible automatically with Preview on Mac. But it never hurts to download GIMP

This program will also allow you to convert your own images to PPM text files so you can practice with pictures of your own (although you can never be sure GIMP will create the same white space as I provided. i.e., one row might not equal one line in that file). You can do this by opening any image file in GIMP and exporting it with the .ppm extension.

Assignment

Write `effects.py`, an application in Python that contains the following functions for manipulating any ppm image file:

1. `shades_of_gray`: converts a color image to a black and white image
   - replace each pixel's individual RGB values with the average of the three values
   - e.g. if a particular pixel had RGB values 100 200 300 you would change this to 200 200 200 for the black and white version.
2. `negate_red`
   - change just the RED color numbers into their "negative"
   - "negative" = not negative numbers, but (maxColorDepth - x) if the original red number was x
   - e.g. if max color depth specified in the header is 255, the 0 would become 255 ,and 100 would become 155
   - The code should work for any file, not just ones with max depth 255 so make sure you read that in from the header
3. `negate_green`
4. `negate_blue`
5. `mirror`: flip the picture horizontally
   - pixel that is on the far right end of the row ends up on the far left of the row and vice versa
   - remember to preserve R-G-B order within each pixel

Each function above should read in a ppm file (this should be an argument of the function) and write out a new ppm file.

Then write `tester.py` that imports the `effects` module you wrote and prompts the user to input an image for modification and choice of modification.

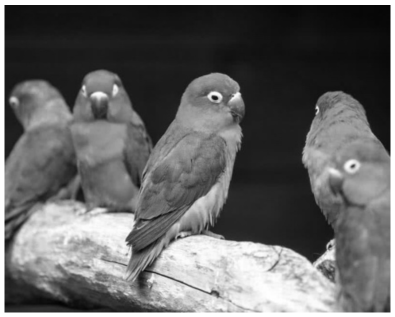
Example session with a user:

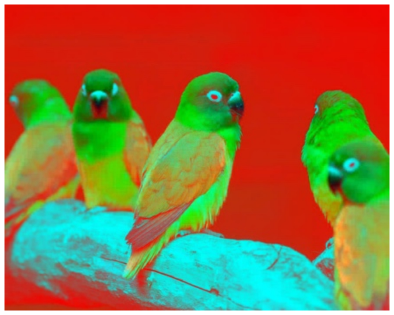
```
Welcome to the Portable Pixmap (PPM) Image Editor!
Choose the effect you would like to try:
1) shades_of_gray
2) negate_red
3) negate_green
4) negate_blue
5) mirror
Enter a number: 2
Enter an input file name: test.ppm
Enter name of output file: out.ppm
out.ppm created.
```

TIPS:

- Make sure you're saving the project files in the same folder as the image files. The output file will also be in the same directory.
- The `tiny.ppm` file is a small 4x4 pixel image provided for initial testing. You can also easily tell if your negation methods are successful. When using GIMP or other image viewers, you really need to zoom in to the picture to see the different color boxes and check your effects Remember it's a 16 pixel image. If you note how most pictures printed now are at least 1200*1800 pixels, it's a really small picture.
- `pics.zip` contains several files for you to test your code. Again, note that each row of pixels in picture = each line of text file. Try smaller pictures first. The largest is `bird.ppm` and the results should look like this:

original


shades of gray


negate red

Optional Assignment
Modify the tester so that the user can choose more than 1 (however many) functions to implement on the input image. Remember those variable length parameters! This is where those come in handy! ☺

Referenced:
Professor Adam Cannon (Columbia University)

Images: https://themysteriousworld.com/top-10-most-colorful-animals-in-the-world/, http://www.bu.edu/tech/support/research/training-consulting/online-tutorials/imagefiles/image101/