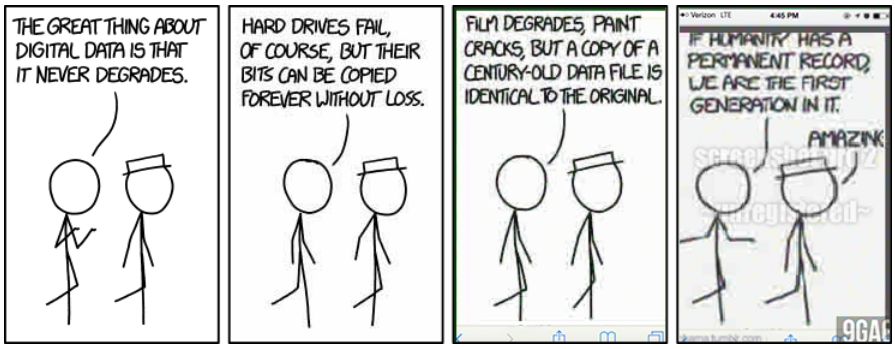
CS51 - Assignment 9
PPM process

Due: November 20th, at 11:59pm



https://xkcd.com/1683/

For this assignment, we will:

- Gain more practice working with lists, especially nested lists, that is lists of lists.
- Write a program that manipulates image files.

1 Getting started

Create a new project, `assignment 9`, 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 will happen automatically if you double-click the file).

You should see a folder named `starter` that contains two files (`ppm_proccess.py` and `ppm_process_tester.py`) and one subfolder named `files`. Copy the two Python files and the folder into the (recently created) assignment folder.

2 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:

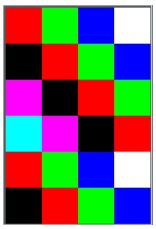
```
P3
4 6
255
```

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. In other words, 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. 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. Given that each pixel holds RGB values (red, green, and blue, so three values), the number of integers on each line will be `3 * c`. 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 r, g, and b 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 provide the r, g, and b values for the pixel to the right (pure green). 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): 

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

3 Implementing `process`

For the first part of this assignment, you will write a function called `process(lines, rows, cols)` that will store the body of an image as a nested list, a list of lists. This function takes 3 parameters. The first parameter, `lines`, is a list of strings consisting of the image body. The second parameter, `rows`, is an `int` representing the number of rows (e.g., the height of the image). The third parameter, `cols` is an int representing the number of columns (e.g., the width of the image).

The `process` function returns a list of lists. The length of this list should be `rows`. Each element of this list should be a list of length `3*cols` representing the r, g, b values for all the pixels in a given row in the image.

The reason why we need to implement the `process` function is that some PPM image files may have their image body not well aligned with the given image width and height as specified in the header. For example, the image can have an image body that contains all the pixels and their RGB values in a single line. Even though this will not affect the image to be displayed or viewed, it may bring up an extra challenge for image manipulations, such as scale (e.g., shrink the image to a smaller scale).

As an example, consider the test file `small.ppm` in the files we provide you. If you open it on VS Code, you will see that this image has a width of 4 and a height of 6. If we opened the file for reading as `file_in`, we could get the image body by executing the single statement `lines = file_in.readlines()` after reading in the three header lines. Then, `lines` is a list of strings that looks like:

```
['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']
```

In this case, he return value of calling `process(lines, 6, 4)` would be:

```
[[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]]
```

Note that here, each sub-list represents one row of pixels, which is well aligned with the image header information specifying that the image has 6 rows and 4 columns.

Of course, the `process` function can allow you to make any manipulation to the image body. For example, if we executed `process(lines, 4, 6)`, the return value would be:

```
[[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]]
```

3.1 Hints and Suggestions

Make sure that you understand what `process` should do before you implement it.

Make sure you understand what the parameters to `process` are. That is, given that the following works, what exactly is the type of that first parameter?

```
file_in = open('files/small.ppm', 'r')
file_in.readline()
file_in.readline()
file_in.readline()
process(file_in.readlines(), 6, 4)
```

Think about the basic structure you'll want your `process` function to have. For example, what loops will you need? What conditionals will you need? What variables will you need? What are some of the cases you will encounter (e.g., what if a row of pixels is broken up over multiple lines in the input file?)

Sketch out pseudocode. Pseudocode is not meant to be complete code; rather, it is supposed to capture the overall structure. It can be almost all comments. It can have for loops and if-statements. But it should be clear how you will translate the pseudocode to actual code. In other words, you should be able to semi-confidently type a first draft of the process function into VS Code based on your pseudocode.

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

You will next implement a program that scales images. Scaling refers to reducing the width and/or height of an image. For example, if we wanted to scale the height by a factor of 2, then we

would take every other row. If we wanted to scale the width by a factor of 2, then we would take every other pixel in a given row. To support this functionality, you will implement four functions: `read_ppm`, `write_ppm`, `scale`, and `main`.

4 Implementing `read_ppm`

This function takes a string as input, which is the file name of a PPM image. The function opens the file for reading, reads it in, calls `process` with the correct parameters as specified in the image header, and returns the list of lists returned by `process`.

5 Implementing `write_ppm`

This function takes as inputs a list of lists of integer numbers, `image`, and a string, `filename`. Every int will have a value between 0 and 255 (inclusive); every sublist will have the same length. The function interprets this list of lists as an image and writes out a valid ppm file to an output file with name `filename`. Note that the function will need to first write out header information before writing the ints in the lists. The header information needs to be abstracted from the 2-dimensional list `image`, e.g., the height of the image should be the number of rows in `image`.

6 Implementing `scale`

This function takes an `image` (a list of lists of ints, as returned by the `process` function) and two scaling factors, `row_scale` and `col_scale`, both ints. The output is a list of lists of ints.

Whatever `row_scale` is, the output should take every `row_scale`-th item in `image`. So the list returned by `scale(image, 3, 1)` on the test file (`small.ppm`) should be:

```
[[255, 0, 0, 0, 255, 0, 0, 0, 255, 255, 255, 255],
[0, 255, 255, 255, 0, 255, 0, 0, 0, 255, 0, 0]]
```

Similarly, the output should take every `col_scale`-th pixel in `image` (keeping in mind that each pixel is represented by 3 numbers - the r, g, and b values). For example, the list returned by `scale(image, 3, 2)` on the test file (small.ppm) should be:

```
[[255, 0, 0, 0, 0, 255],
[0, 255, 255, 0, 0, 0]]
```

Hint: the rows that you want are those where the *index%row_scale* equals 0.

When you believe your function is correct, add test cases to the test file `ppm_process_tester.py` to thoroughly test your function.

7 Implementing `main`

The `main` function should ask the user for an input filename, an output filename, a height scaling factor, and a width scaling factor. It should enforce the restriction that both scaling factors must be positive integers. It should then read from the input file and create a new file with the specified name that contains a copy of the input file scaled down by the specified factors.

8 General Suggestions

- Incremental development and testing will be helpful. We strongly suggest that you start by writing docstrings for each function and then implement and test them one at a time.
- While you're debugging your code, we suggest not worrying about having the user input the name of the input and output files every time you run your program; just have fixed strings in your code. This should save you some time!
- You can the test files we provide. This includes the 4-by-6 image from the Background section, which may be helpful for debugging.
- 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.

9 Feedback

Create a file named `feedback9.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?

10 When you're done

Submit your `pp_process.py` and `pp_process_tester.py`, and `feedback9.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.

11 Grading

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

	points
`process`	2
`read_ppm`	2
`write_ppm`	2
`scale`	2
`main`	1
`tests`	1
total	10