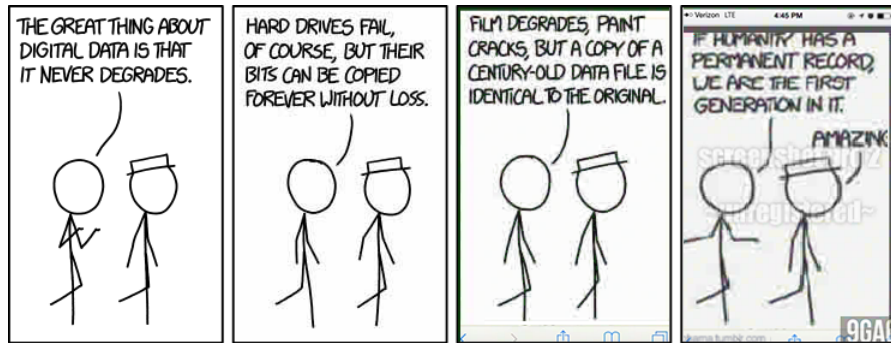


CS51 - Assignment 9

PPM Scaling

Due: April 23rd, at 11:59pm



<https://xkcd.com/1683/>

For this assignment, we will:

- Gain more practice working with lists, particularly lists of lists.
- Play more with writing programs that manipulate 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. What would be the printed out if we ran the following code?

```
lines = ["1 2 3 4 5 6", "1 2 3 4 5 6", "1 2 3 4 5 6"]
print(process(lines, 3, 2))
```

2. You call `read_ppm` and receive the following data:

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9], [11, 12, 13, 14, 15, 16, 17, 18, 19]]
```

What are the width and height of this image?

3. What would be printed out if we ran the following code?

```
image = [[1, 2, 3, 4, 5, 6, 7, 8, 9], [11, 12, 13, 14, 15, 16, 17, 18, 19]]
print(scale(image, 2, 2))
```

2 Getting started

Create a new project, **assignment9**, 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 (**assignment9.py** and **assignment9_tester.py**) and one subfolder named **files**. Copy the two Python files and the folder into the (recently created) assignment folder.

If you need a refresher on how PPM files work, please read the Background section in assignment 8.

3 Overview

In our last assignment, we played a bit with reading PPM files as well as how we can manipulate the individual pixels in an image. In this assignment, we're going to play more with images and look at some additional manipulations (scaling of an image) that require us to have a better representation of the image, in our case, a *list of lists*.

4 Implementation details

For the assignment, you'll implement five functions that will allow us to read/write PPM files and then scale them.

4.1 process

Write a function called **process(lines, rows, cols)** that takes three parameters and that returns the body of an image as a *list of lists*. 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.

Recall from last assignment that the PPM file format is fairly flexible with how the body of the image is represented. The *header* information defines the actual shape of the image and the body is just a list of RGB values for each pixel in the image.

The `process` function will be used to read in the body of the image and return it as a list of lists with the proper dimensions, regardless of how the body is actually laid out in the PPM file. 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 how the image is 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 in 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, the 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.

The `process` function also allows you to manipulate the dimensions of 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]]
```

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. For example, here is one example of how you could call the function:

```
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 **assignment9_tester.py** to thoroughly test your function.

4.2 read_ppm

Write a function **read_ppm** that takes as input a **string** representing the name of a ppm file and returns the body of the image as a properly sized (as specified by the header information) list of lists. The function should process the header information of the PPM file and then use that along with **process**.

For example, if we were to call this function with **files/small.ppm** we would end up with a list with 6 entries, where each of those entries had 12 values (4 pixels each represented by an RGB value).

4.3 write_ppm

Write a function **write_ppm** that takes as input two parameters: 1) a list of lists of ints representing the body of an image, and 2) the filename as a string. Every int will have a value between 0 and 255 (inclusive) and every sublist will have the same length. The function should interpret the list of lists as an image and writes out a valid ppm file to the filename. The function should write a proper PPM file which needs to have the header information before writing the image body. The

header information should be determined from the size of the list of lists, e.g., the height of the image should be the number of rows.

For example, if you were to call:

```
write_ppm(read_ppm("files/small.ppm", "out.ppm"))
```

and look at the output in `out.ppm`, the file would be the same as `files/small.ppm` (minus, possibly, minor whitespace differences).

5 scale

Write a function called `scale` that takes: 1) an `image` represented as a list of lists of ints (i.e., as returned by the `process` function), as well as two scaling factors 2) `row_scale` for scaling the rows and 2) `col_scale` for scaling the columns (both ints). The function should return a list of lists of ints, representing the scaled input image.

Whatever `row_scale` is, the output should take every `row_scale`-th item in `image`. For example, if we scaled called:

```
image = read_ppm("files/small.ppm")
scaled = scale(image, 3, 1)
```

and then printed out `scaled`, we would see:

```
[[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 `files/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 `assignment9_tester.py` to thoroughly test your function.

6 scale_ppm

The last function will put it all together so that we can scale images. Write a function called `scale_ppm` that has four parameters: 1) an input filename (string), 2) and output filename (string),

3) a height scaling factor (int), and 4) a width scaling factor (int). The function should scale the input image using the scaling factors and write the result to the output filename. Most of the work of this function should be done by your previous functions.

Your function should check to make sure that both scaling factors are positive integers. If they are not, you should raise an exception:

```
raise Exception("Scaling factors must be positive integers")
```

7 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.
- 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.

8 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?

9 When you're done

Submit `assignment9.py`, `assignment9_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.

10 Grading

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

	points
Warm-up	1
process	2
read_ppm	1
write_ppm	2
scale	2
scale_ppm	0.5
tests	1
style/comments	0.5
total	10