CS 115 Homework 4: ASCII Image Rendering

Breaking news from *The Stute*: there is buried treasure somewhere in a small but dangerous archipelago off the coast of Hoboken. You and your friends want to get to it before anyone else does.

One friend works at the library and knows there's a treasure map in an old computer in the library. Your goal is to view the treasure map, then obfuscate it so no one else can find the treasure

As the 31337 H4X0R (elite hacker) of the group, you manage to find the computer and find the map, but your console is text-only, just as an elite hacker's must be (Fig. 1).

Figure 1: Map of dangerous archipelago off the cost of Hoboken. Shows boat, sea monster, but no treasure.

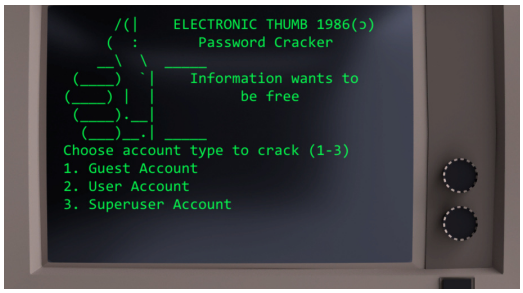

Figure 2: Decorative: Green text on a black image showing an ASCII thumb. Image source: `https://community.gamedev.tv/t/80s-style-cracking-tool/61805/2`

In other words, your task in this lab is to

1. **Task 1:** Analyze the image format used
2. **Task 2:** Write code to render the treasure map and other images on a text-only terminal.
3. **Task 3:** Edit the image

Task 1: Working with the image data

In this task, you will inspect an image file directly, then inspect it in Python represented as a 2D list of RGB pixels.

The treasure map is stored as a PPM (Portable PixMap) file, an cross-platform image file format like PNG and JPEG that was created in 1988. The header at the top specifies

- File type (`P3`) — meaning "ASCII RGB image."
- Width and height (number of pixels per row and column).
- Maximum color value (usually 255).

The remaining lines list pixel values in `Red Green Blue` (RGB) order for each pixel, left-to-right and top-to-bottom. Since there is no color (except neon green) on your screen, we will be immediately converting the image to grayscale, but if you're curious you can learn more about the RGB color model here: `https://www.britannica.com/science/RGB-color-model`.

Inspect the provided `small_secret_image.ppm` file in a text editor and answer:

Report question

1. According to the header, how many pixels wide and tall is the image?
2. How many rows of image data are there (excluding the heading)?
3. How many columns of image data are there?
4. What are the RGB values of the top-left pixel? (Just report: no need to describe the color)
5. Explain as you would to your non-hacker friends how the RGB values are encoded in this image format.

Next, you will load the N by M image into Python as a list-of-list-of-lists: a list of N rows containing M tuples with 3 values in each tuple.

You do not need to report anything for this, but inspecting the dimensions of `list_of_lists_of_rgbs` (such as by printing out the `len` of `list_of_lists_of_rgbs`, of each row, etc ...) may help you with the rest of the lab.

Task 2: Rendering PPM to ASCII

In this task, you will convert the list-of-list-of-lists into a string, which you can print with the `print` function!

Converting to grayscale

First, the image needs to be converted to grayscale, meaning the RGB tuple needs to be converted to a single brightness value based on the color. In this homework, we will use the Rec. 601 standard used for digital television: brightness $= 0.299R + 0.587G + 0.114B$

Complete the function `rgb_to_grayscale` so that it **mutates** a 2D list of RGB pixels into a 2D list of grayscale values (integers). You should use the `map` function to apply your conversion across each row.

Rendering to ASCII

Try to print your `grayscale_image`. It still doesn't look like anything readable. In this next function, (`image_to_ascii_string`), you will construct a string that you can `print` using the image represented as a list-of-lists.

Each pixel's brightness (0–255) will be mapped to a character that visually represents that brightness. Darker (lower brightness) pixels correspond to denser symbols (like `@`), and lighter pixels to lighter ones (like spaces). Use the following table to implement `brightness_to_ascii`:

Brightness range	ASCII character	Character name
0–25.99	`@`	at
26–51.99	`#`	hash
52–76.99	`%`	percent
77–102.99	`*`	asterisk
103–127.99	`=`	equal
128–153.99	`+`	plus
154–179.99	`-`	hyphen
180–204.99	`:`	colon
205–230.99	`.`	period
231–255		space

Table 1: Table mapping brightness values to ASCII characters

First, implement the function `brightness_to_ascii`, which maps an brightness to a ASCII character arranged from darkest (`"@"`) to lightest `" "`: `"@#%*=+-:. "`. (*Hint: you only need to specify one number from each range*)

Then, call `brightness_to_ascii` on each pixel in the list-of-lists to convert each brightness value to its corresponding ASCII character, then combine the results into a *single string*. Remember to add a carriage return (`"\r"`) and a newline character (`"\n"`) to the end of each row: `"\r\n"` to make sure the newlines show properly on Windows. For example, the grayscale image (same values in each row) `[[0,0,0, 100, 100, 100, 170, 170, 170, 220, 220, 220],[0,0,0, 100, 100, 100, 170, 170, 170, 220, 220, 220], [0,0,0, 100, 100, 100, 170, 170, 170, 220, 220, 220]]` would be written as follows:

```
ascii_image = (
    "@@@***---...\r\n@@@***---...\r\n@@@***---...\r\n"
)
```

and rendered like this during `print`, which you can do using `print(ascii_image)`

```
@@@***---...
@@@***---...
@@@***---...
```

Report question

Submit your `ascii_image` by uncommenting and running the line: `save_string_to_txt(ascii_image, "treasure_map.txt"`

In addition to the treasure map, we also provided a few other images for you to test your ASCII renderer.

- `pumpkin.ppm`
- `jolly_roger.ppm`
- `sea_monster.ppm`
- `gold_coins.ppm`

In order to view larger images, you may need to set your IDLE console to be small enough to show all characters in one row. You can do that by going to Settings / Font / decrease font size (but use a monospace font!) or Settings / Windows / then toggle settings for window width.

Task 3: Image Editing

Your final task is to edit the map so no one else can find the treasure. Implement **at least two of the following editing functions**. (You do not need to implement all four for full credit). Then, you will edit the code in the function `main` to edit the grayscale image so the treasure map location cannot be read. (In real life, you would save `treasure_map.ppm` with your edited version).

All of the functions should **mutate** the image input rather than returning a new one.

- `invert`: Converts the brightness i into $255 - i$
- `lighten`: Increases the brightness for all pixels in the image by 30%, but no higher than 255
- `darken`: Decreases the intensity for all pixels in the image by 30%, but no lower than 0.
- `erase`: Erases the image and leaves it with all 0s

Implementing the three functions you chose in `hw4.py` and call them to edit the file, printing the image as you go along. Uncomment the final line of `main.py` to save the string as a PPM.

Labor log

We would like you to share your software development **process**. Especially, we want you to write about how you **debugged** your code, which is one of the most important parts of creating code.

In your labor log in `hw4_report.txt`, write about:

Required components:

1. How did you locate and correct any mistakes in your code? What strategies did you use? (At least 2 sentences)
2. Which strategies for correcting your code were the most and least effective? (At least 2 sentences)

In particular, we are looking at how you model what you *expect* your code to do and how you observe what your code *actually* does.

In addition to the required reflection, we also suggest logging other important aspects of software development. This is not graded, but we hope it helps you raise awareness of your engineering process.

Recommended optional components:

1. Timing: when you started the homework, how many hours you worked on it
2. Environment: where you worked on the homework, with whom, ...
3. What you will continue and/or do differently for next homework
4. What you're proudest of?

Submission Instructions

For this homework, you must submit the following files, with exactly the following names:

- `hw4.py`
- `hw4_report.txt`
- `treasure_map.txt`

Remember to put your name and the Stevens Honor Pledge on every file you submit! Failure to put the pledge can result in marks being deducted, with repeat penalties potentially getting stricter between assignments.

Submissions must be handled through Gradescope (accessible through the homework page on Canvas). Grades will be released through Gradescope once the homework is graded.

Hints and Suggestions

- Spread function implementation out over multiple days.
- When debugging, start by testing on smaller examples before moving onto larger examples.
- Make sure functions used by other functions, such as `rgb_to_greyscale`, and **especially** `brightness_to_ascii` pass all tests before investing much time in later functions.
- You can run a few tests at a time using the `-k` flag in IDLE with the name of the functions you want to test. For example to test just the `convert_to_grayscale` tests, go to IDLE, Custom Configurations, and type in `-k convert_to_grayscale`. You can even include full test names to run one test at a time.
- Come to office hours to work on your homework!

Notes and Reminders

- Keep track of the work you put into this homework, **especially in how you overcame difficulties**. Share in the "labor log" of the report.
- **Remember to include docstrings!** If your submission does not have docstrings, or has docstrings that are not related to your code, you will not get credit for them.
- Additionally, if your functions are more complicated than usual, you should add comments inside them explaining how they work.
- We have provided a test file, `test_hw4.py` for you to test your solutions to ensure that files are named correctly and to check the correctness of your code. Try out many different cases to verify that your code runs. Our provided test file may not be comprehensive, and so you should ensure to test your code on many other cases as well.
- Code that doesn't compile (throws a `SyntaxError` on being run) will receive a 0 for the assignment! Make sure you submit code that can be run, even if it does not pass all the test cases.
- **Don't forget to add your name and pledge!**