Assignment 1

Objectives: Practice string formatting and perform string and byte-array addition and concatenation.
Note: In each python script you submit, modify the DocString at the top and add your name. (Notice the two empty lines.)

```
"""This program makes a BMP file with a plus sign of a specified color


Submitted by Mauricio Arias, NetID ma6918
"""
```

Part 1. Demoing a progress bar.
Later on in the semester we will write code for a progress bar in the command prompt that updates as more and more tasks of a total number of tasks are completed. For the moment being we will write code that shows **two snapshots** of a progress bar indicating the progress of an imaginary task.

We will use parameters for making the bar. Use variables to store the following **parameters**:

- Progress character: This character covers the "completed" part of the bar: '>' in the first example below, '▓' in the second. It is your choice which one to use.
- Fill-in character: This character covers the remaining section of the bar: '-' in the examples.
- Length of the bar: total length of the bar in characters; 50 was used in the bars below. Make sure it fits in a standard line.
- Total to complete: a number indicating the size of the imaginary task the progress bar is illustrating, for example, 10000 total calculations, 500 employees to process, 10000000 pixels to encode, etc. Ask the user what the total number of tasks is. (Note: at this stage we won't verify that the value is acceptable. Warn the user that it is an integer greater than 0 though.)
- For this demo, the amount completed for one of the progress bars will be given by the user: how far along the total we are in the imaginary task, for example, 3000 calculations, 10 employees processed, 1000000 pixels encoded, etc. **Ask** the user how many of the total tasks have been performed and **convert** that information to the appropriate data type. (Note: at this stage we won't verify that the value is acceptable. Warn the user that it should be an integer between 0 and the total supplied in the previous bullet point.) The other progress bar will be shown first and corresponds to 0% completed.

Use f-strings. Notice how the percentages are aligned in the examples below.

The code has to generate two snapshots by copying the code manually: neither loops nor conditionals are allowed in this assignment. Copy only what is strictly necessary to make a second bar, assuming they are both snapshots of the same process but at different time points. Progress for the second snapshot will be specified by modifying the "amount completed" variable after the first print() section is executed. Submit your code and include your NetID. Name your script `[NetID]_progress_snapshots.py`.

Example 1

```
Snapshot 1
Progress |--------------------------------------------------|      0.0% completed

Snapshot 2
Progress |>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>|     80.0% completed
```

or example 2 with a different fill-in character

```
Snapshot 1
Progress |--------------------------------------------------|      0.0% completed

Snapshot 2
Progress |▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓|     60.0% completed
```

Part 2. A simple bitmap using byte-arrays

In Python, types that define an addition operation, usually also define a multiplication operation that enables intuitive applications. As seen in class, for some data types, addition corresponds to concatenation. (For other data types it means something else.) For example, as you learned in Module 1, two strings can be concatenated by "adding" them together. In the same way, two byte-arrays can be concatenated by adding them together. Multiplication becomes then the repetitive addition of the same byte-array a specified number of times: for example if the variable `pixel` contains a byte-array, then `pixel * 3` is the same as `pixel + pixel + pixel`, or three identical copies of `pixel` concatenated.

Bitmaps are files that describe a figure by specifying the color to be used for each pixel. They have dimensions in pixels: for example 200 pixels wide x 300 pixels high. As illustrated in Fig. 1, these files have two main sections: the header, where the details necessary for interpretation of the file are included, and the data, where the information for all the pixels is included.

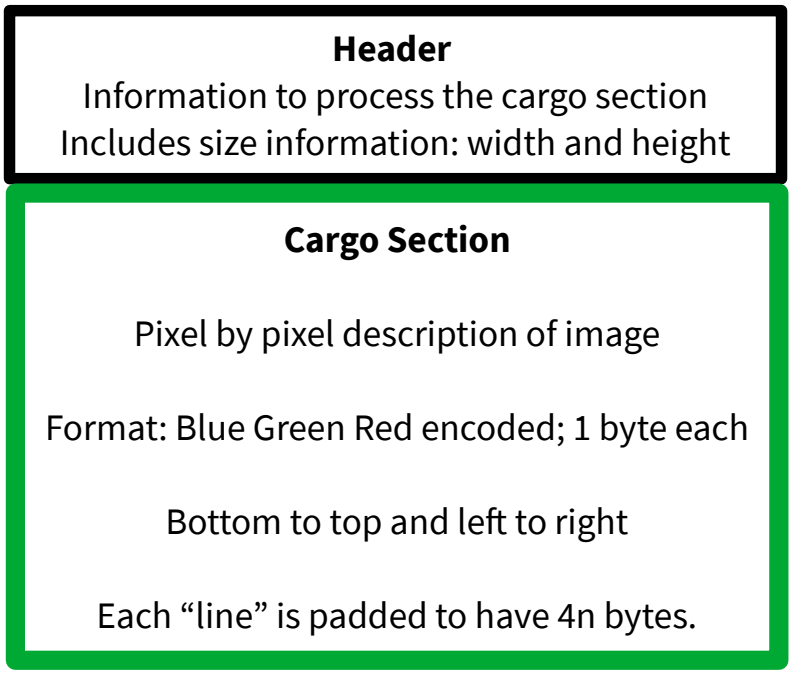

Figure 1. Overview of the main two sections of a bitmap file. Notice that the size of the image is defined in the header.

Two additional files are included with this assignment to help you with this project as well as two template files you'll use to code. The templates include some code to handle the minutiae of putting together the header section and saving the file. We'll cover the tools to perform those tasks later in the

semester but for now they are provided. The template also includes a section labeled "`# TODO`" indicating where your code goes.

The first additional file, `utilities_BMP.py`, provides the functionality to specify the header and is used in the template file. Even though this is a simple task, it is somewhat tedious. So I have provided it. The second file, `colors_BMP.py`, includes information for the colors that can be used. Each color is packed as a byte-array and is labeled with an approximate description of how they'll be perceived. By using the import instruction at the beginning of the template file (we'll cover that later), they are available in the main file and can be chosen by name when you code: open this file to see the colors available.

Task 2.1. Write code to generate a square 500 pixels in length. (Use input to request the size of the square.) The sides of the square as well as the top and the bottom are 5 pixels wide each and are of a different color than the rest of the square. See Figure 2. Do not use loops or conditionals in your code.

For reference, binary numbers are specified by bits (binary digits). Similar to how we say 1034 has four digits, in binary 1101 has four bits. To make counting easier, 8 bits constitute a byte, which is more commonly used than bits. The information for pixel colors is stored in 24 bits, which corresponds to 3 bytes. One byte specifies the "amount" of blue necessary to make the pixel look a certain color, while another byte specifies the amount of green and another one the amount of red.

The image is created by **concatenating** pixels to the variable `image_binary`, which already contains the header. When specifying the pixels, each horizontal line has to contain a multiple of 4 bytes. Notice that each pixel contributes 3 bytes (24 bits). Hence, if the figure is 50 pixels wide, for example, each horizontal line has to be padded with "ignorable bytes" to reach the next multiple of 4: for a total of 152 bytes in this case. A padding byte is available for your use: `bmp.pad_byte`. The formula is provided in the template. Make sure you understand how it works. Notice that these extra bytes, if necessary/present, are ignored by the software reading the image. However, they are read and then discarded. This image does not require or use separators between horizontal lines as they are all concatenated together.

Once you are done, open the file `my_bitmap.bmp` with any software capable of displaying images so you can check your work: firefox, photoshop, safari or paint for example. You can also verify the details of the figure by checking its properties to verify its size. Make the code general enough so that other sizes can be used. Rename the first image using your NetID as `[NetID]_square_1.bmp`. Now, generate a 501 x 501 square by choosing any 2 colors and label it `[NetID]_square_2.bmp`. (Do not use white or black for the top or bottom layers to facilitate visualization.) Submit your code and the two images. Name your script `[NetID]_square.py`

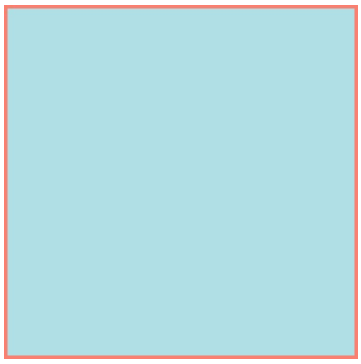

Figure 2. "Square" to be generated for task 2.1.

Task 2.2. Write code to generate an image 900 pixels wide and 1100 pixels tall with a “plus sign” that uses 2 colors: dim grey for the background and steel blue for the plus sign. The thickness of the sign should be 100 pixels. See Figure 3. Request the sizes from the user: height, width and thickness. Notice that the template has variables for these sizes. Use those variables.

When specifying the pixels, each horizontal line has to contain a multiple of 4 bytes as in the previous task. Once you are done open the file `my_other_bitmap.bmp` with any software capable of displaying images so you can check your work: firefox, photoshop, safari or paint for example. Make the code general enough so that other sizes can be used. Rename the image using your NetID as `[NetID]_plus_sign_1.bmp`. Generate a 550 x 1000 “plus_sign” that is 50 pixels thick by choosing any 2 colors and label it `[NetID]_plus_sign_2.bmp`. (Do not use white or black for the background to facilitate visualization.) Submit your code and the two images. Name your script `[NetID]_plus_sign.py`.

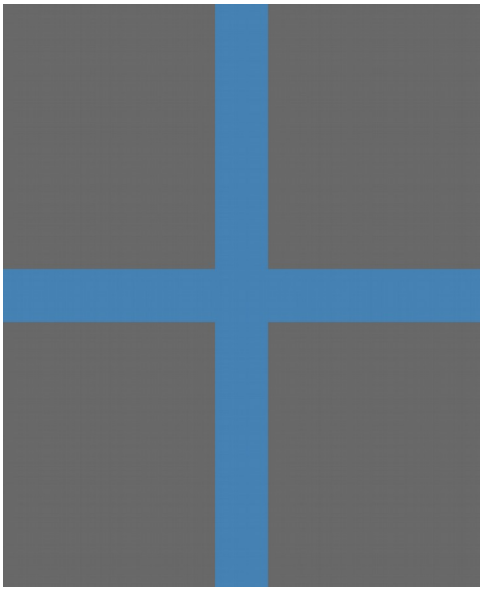

Figure 3. Plus sign as expected for task 2.2.