COMP 1900 - Fall 2021

Lab 1: Number Systems, Basic Python Programs

Total Points: 35

Due: Fri., Sept. 10, by 2359 CDT
Please carefully read the submission instructions at the end of the assignment. As with all other assignments, this should be done *individually*.

Grader: The lab TA who supervises your registered lab section will grade your submission. Grades will generally be posted within 1-2 weeks of the assignment due date. Questions about grading? Please contact them first.

Part 1: Number Systems and Binary Numbers

Please type or neatly write your solutions to these problems. *Show all work* for full credit!

1. (3 points) Convert $(10111001)_2$ into base 10.
2. (3 points) Check your answer for the previous problem by converting it back into base 2.
3. To store text characters in binary, a computer uses an **encoding scheme** such as ASCII or Unicode. The idea is to **encode** characters as certain sequences of bits — A might be represented by 0001, B might be represented by 0010, and so on. When the computer sees a particular sequence of bits, it interprets that sequence as its character equivalent.

   The number of bits used in an encoding scheme determines how many unique characters can be represented. For example, consider three encoding schemes: scheme S_1 uses 1 bit per character, scheme S_2 uses 2 bits per character, and scheme S_3 uses 3 bits per character. Because each individual bit has only two possible values (0 or 1), S_1 can represent up to two unique characters (encoded as 0 or 1). S_2 can represent up to four unique characters (encoded as 00, 01, 10, or 11), and S_3 can represent up to eight unique characters (encoded as 000, 001, 010, 011, 100, 101, 110, or 111).

   (a) (2 points) In an encoding scheme that uses 4 bits per character, what are the possible sequences of bits? There should be 16 in all.

   (b) (1 point) So far we've observed the following:

   - Using a 1-bit encoding scheme allows 2 unique characters to be represented.

- A 2-bit encoding scheme allows 4 unique characters.
- A 3-bit encoding scheme allows 8 unique characters.
- A 4-bit encoding scheme allows 16 unique characters.

In general, for an encoding scheme that uses n bits per character, how many unique characters can be represented? Your answer should be an expression involving n.

(c) (3 points) Suppose you're developing a new, super-complex language that contains 119 consonants and 52 vowels. Each consonant and vowel can be upper or lower case. What is the smallest possible number of bits per character you could use in an encoding scheme for this language? Clearly explain how you arrived at your answer. Assume a fixed-length encoding scheme (i.e., every character uses the same number of bits), as in our examples above.

4. Colors can be represented in a computer using an **RGB** color scheme, where each color is a combination of red, green, and blue components. In 24-bit RGB, each component uses 8 bits, from a low of $(00000000)_2$ to a high of $(11111111)_2$. This translates to a range of 0-255 in base 10. The 8 bits of red are stored first, followed by the 8 bits of green, and finally the 8 bits of blue.

   For example, the color green has RGB values (in base 10) of red 0, green 255, blue 0. A computer stores this as 24 bits: $(00000000\ 11111111\ 00000000)_2$. Meanwhile, purple is a mix of red and blue: red 255, green 0, blue 255. This is stored in binary as $(11111111\ 00000000\ 11111111)_2$.

   24-bit RGB colors are usually expressed in hexadecimal (base 16) for conciseness. Recall that when converting base 2 into base 16, the bits are grouped in sets of 4. Therefore, these **hexadecimal color codes** are 24 / 4 = 6 digits in length. The first 2 digits represent the amount of red, the middle 2 digits represent the amount of green, and the last 2 digits represent the amount of blue. The 6-digit hexadecimal code is usually preceded by a pound sign (`#`) to clarify that it's expressed in base 16. For example, the code for green is `#00FF00`, while the code for purple is `#FF00FF`.

   Consider a color with the following RGB values in base 10: red 240, green 121, blue 46.

   (a) (6 points) What is the hexadecimal code of this color? Be sure to show your work!

   (b) (2 points) What is the sequence of 24 bits that a computer would store for this color?

   You can use a website like `https://www.webfx.com/web-design/color-picker/` to help verify your answers, but you must *show your work* to get full credit for this problem.

5. In the previous problem, we saw that each color can be represented using 24 bits of data. It's pretty easy to extend that idea to images. An image can be stored in a computer as a rectangular grid of **pixels**, each of which is assigned one color.

   You've probably heard the term "4k" before. A 4k image has a resolution of 3840 pixels (horizontally) by 2160 pixels (vertically).

   (a) (1 point) How many total pixels are there in a 4k image?

   (b) (1 point) Assuming that each pixel takes 24 bits to store its color information, how many bits does it take to store the colors for all the pixels in a 4k image?

(c) (1 point) Convert your previous answer into *bytes.* (By definition, 1 **byte** = 8 bits.)

(d) (2 points) A video is simply a sequence of images (called **frames**) that are shown rapidly to produce the illusion of movement. Suppose you want to make a video consisting of 20 minutes of raw 4k gameplay at 75 frames per second for your YouTube channel. In theory, how many *bytes* of data does it take to store this video?

For full credit, *show all your calculations* for these questions!

Part 2: Basic Python Programs

If you haven't already, walk through the instructions in the Software Setup Guide posted on eCourseware to make sure that Python is installed on your computer. If you don't already have a `1900` folder on your desktop, create one. Within the `1900` folder on your desktop, create a new folder named `Lab1`. All your source code files for this assignment should be saved in this `Lab1` folder.

1. (5 points) Within your `Lab1` folder, write a program named `FavoriteQuotes.py` that displays three of your favorite quotes, within double quotation marks. Each quote should be followed by a line showing who said/wrote it. Skip one line between each quote. Example program output:

```
"And so even though we face the difficulties of today and tomorrow, I
still have a dream. It is a dream deeply rooted in the American dream."
 - Martin Luther King, Jr.

"Horns, horns, horns, in dark Mindolluin's sides they dimly echoed. Great
horns of the north wildly blowing. Rohan had come at last."
 - J.R.R. Tolkien, The Return of the King

"A well-built physique is a status symbol. It reflects you worked hard
for it. No money can buy it. You cannot inherit it. You cannot steal it.
You cannot borrow it. You cannot hold on to it without constant work. It
shows dedication. It shows discipline. It shows self-respect. It shows
dignity. It shows patience, work ethic, passion. That is why it's
attractive to me."
 - Pauline Nordin
```

(The last quote also seems often attributed to Arnold Schwarzenegger. I haven't been able to find definitive information on who actually said it first.)

2. (5 points) **ASCII art** involves making “pictures” using text characters. For example, here’s a cute one of a sloth:

```
‘""==,,__
  ‘"==..__"=..__ _    _..-==""_
       .-,‘"=/ /\ \""/_)==""‘‘
      ( (    | | | \/ |
       \ ’.  |  \;  \ /
        |  \ |   |   ||
   ,-._.’  |_|   |   ||
  .\_/\     -’   ;   Y
|  ‘  |        /    |-.
’. __/_    _.-’     /’
        ‘’_.._____._’
```

Find a nice piece of ASCII art on the Internet. Or, if you’re feeling particularly artistic, create your own! It can be anything you want, as long as it satisfies these requirements:

- The art must be at least 10 characters wide by 10 characters tall.
- You should be comfortable showing it to a random grandma.

Within your `Lab1` folder, write a program named `ASCIIArt.py` that prints your art onto the screen. You’ll likely have to use one or more escape sequences in your code to display certain characters.

Be sure to verify that both of your programs successfully run, as demonstrated in the Software Setup Guide.

Need Help?

- Attend your weekly lab meeting. Your lab TA can give you live help during this session.
- Email your lecture instructor, and/or attend office hours.
- The UofM offers free online tutoring through the Educational Support Program (ESP): `https://www.memphis.edu/esp/onlinetutoring.php`

Submission Instructions

- Put your work and answers from Part 1 into a *single PDF file.* If you wrote your answers on paper, please scan them into a single PDF file. There are a number of free phone apps that you can use for this. I’ve had good experiences with CamScanner on Android; the software is available for iOS as well. *Points may be deducted for submitting in a format other than PDF.*

- Create a zip file containing your PDF file from Part 1 and your Python source files from Part 2. You can use any Python development environment you like, as long as you submit the source files.
- Upload your zip file to the appropriate dropbox folder on eCourseware. The dropbox *will* cut off submissions at precisely the stated deadline, so please submit with some time to spare. Late submissions are not accepted. You may submit as many times as you want up to the deadline. Each submission overwrites the previous one.
- Contact your lab TA to schedule a one-on-one code review via Zoom. During this short meeting (15-30 minutes), your TA will ask you to run your code. They may also ask you to explain your code and/or run different test cases. The code review must be completed for you to get a grade for the lab.

  Note that since the code review happens after your submission, its purpose is *not* for you to get help with writing your code! Getting help should be done during the scheduled lab sessions, or by reaching out to your lecture instructor/online tutors *before* the due date.