MSOE EECS Department
CS2911: Week 2 Lab Grading Checklist
Dr. Yoder Name:

Item	Points
Introduction: Describe the lab in your own words. (You may use the space below.)	/ 1
Problems 1 and 8. (See requirements in exercises)	/ 1
Problems 2 and 9	/ 1
Problems 3 and 10	/ 1
Problems 4 and 11	/ 1
Problems 5 and 12	/ 1
Problems 6 and 13	/ 1
Problem 14. Write the two types you found.	/ 1
Problem 15	/ 1
Problems 16 and 17	/ 2
Problem 18	/ 1
Problems 19 and 20. Do not use str, int, etc.	/ 1
Problems 21- 23. Do not use str, int, etc.	/ 1
Problems 24 and 25	/ 1
Problems 26	/ 1
Problem 27 (excellent credit)	/ 1
Problem 29(optional)	/ 0
Summarize what you learned during this lab. (You may use this space.)	/ 2
Things you liked or suggestions for improvement (Required; you may use this space)	/ 1
Total	/ 20

Follow these instructions for full credit:

- ***Staple*** this lab cover sheet on top of all the materials you are submitting.
- Submit your work in the ***order*** listed above.
- In addition to the materials above, submit any other supporting materials you create while working the lab where they fit best in your report.
- Your lab packet is due at the start of the following week's lab period. There is a 2 point per day late penalty on the packet. Slip your submitted lab packet under my office door or submit your packet to me during the laboratory.

Lab 2: Python Encoding

Work through the first few problems on paper before starting Python. ***Box*** your answers. I encourage you not use a calculator, using the space provided or extra paper.

1. Predict how the `bytes` object `b'2 Faced'` will be stored in Python. ***Write*** your answer in hexadecimal shorthand.

2. Predict how the bytes object b'\r\n' will be stored in Python. ***Write*** your answer in both **hexadecimal shorthand** and **bits (binary)**.

3. Predict how the number `104` will be stored in Python. ***Write*** your answer in **binary**, then ***write*** it in **hexadecimal shorthand**.

4. Predict how the number 0xfe19 will be stored in Python. ***Write*** your answer in **hexadecimal shorthand**, then ***write*** it in **binary**.

5. Predict how the bytes object b'\xbeef' will be stored in Python. ***Write*** your answer in **hexadecimal shorthand**, then ***write*** it in **binary**.

6. Predict how the number `1055` will be stored in Python. ***Write*** your answer in **binary**, then ***write*** it in **hexadecimal shorthand**.

7. Set up the `showbits` library: (See lab page for video version of these instructions)
   a. Go to the Lab 2 webpage and download the python module `showbits.py`.
   b. Place the file directly inside the top-level of your Python project.
   c. Open the Python console using Tools -> Python Console.
   d. In the console, type `from showbits import bits, shorthand`. (If you use this in a file, use `import showbits` instead, and use `showbits.bits()` with the package-name when calling `bits()`.)

As you check your answers to your previous problems, either ***write*** a check-mark if the answer was correct, or ***write*** the difference between your prediction and the actual value and ***write*** what you learned from it.

8. ***Check*** your answer to Problem 1 by typing `shorthand(b'2 Faced')`.

9. ***Check*** your answer to Problem 2 by typing `shorthand(b'\r\n')`.

10. ***Check*** your answers to Problem 3 by typing `bits(104)` and `shorthand(104)`.

11. ***Check*** your answers to Problem 4 by typing `shorthand(0xfe19)` and `bits(0xfe19)`.

12. ***Check*** your answers to Problem 5 by typing `shorthand(b'\xbeef')` and `bits(b'\xbeef')`.

13. ***Check*** your answers to Problem 6 by typing `bits(1055)` and `shorthand(1055)`.

14. Set `i = 3`. Use Python to ***determine*** the type of `i`. (See Java/Python table for hints.) Also ***determine*** the type of `int`. ***Write*** the **two** types.

Python bytes objects represent the bytes used to send data over the network. Writing Python bytes object literals is a good way to exactly specify what you plan to send over the network in examples.

15. Consider the number 196_{10}
    a. ***Write a literal bytes*** object to hold the number in **raw binary**. That is, the internal representation is simply binary. Do this by hand.

    a. ***Write a literal bytes*** object to hold the number in **ASCII decimal**. That is, the internal representation is ASCII codes for each byte. (See the hints on writing literal values from the previous part.). Do this by hand.

    Consider a message in this format: a header consisting of two ASCII digits is followed by a body holding arbitrary payload data. The digits stored in the header form a decimal number.
16. An application message consists of a list of raw binary numbers. Each number uses a variable number of bytes. A single raw binary byte sent before each number tells how many bytes are in the number. After the last number, a 0 size byte is sent.
    a. Write the hex shorthand for a message in this format that sends the numbers in this list: [4, 5, 10, 7]

    b. Write a **bytes object literal** that holds the bytes in part a. Write this in the most convenient way to write it in Python.

17. A message holds several numbers. Each number is stored with a variable number of bytes; a single byte sent before each number stores how many bytes are in the remainder of the number. Both this size and the number itself are stored as ASCII numbers. After the last number, a "0" size byte is sent to indicate the message is over.
    a. Write the hex shorthand for a message in this format that sends the numbers in this list: [4, 5, 10, 7]

18. Imagine you just received a bytes object `b` from a sender over the network. The bytes object holds the bits with hexadecimal shorthand `42 41`. If we run the command `int.from_bytes(b,'big')`, we get the python `int` `16961`. If we run the command `b.decode('ASCII')`, we get the python `str` `'BA'`.

    Describe how you could determine whether it is a raw binary number or ASCII text that is meant to be stored in the `bytes` object. In other words, did the user mean to send the `int` or the `str`? Be creative – there are multiple ways you might know. But you must think outside the bytes object. The goal is critical thinking, not remembering some fact from class or the book.

To prepare data for sending or to understand data that has been received, it is moved in and out of a bytes object. To avoid unnecessary data conversion, in Problems 17-23, do not use the str, int, hex, format, etc. functions. Instead use the commands introduced by your instructor during the lab period (See the Java-Python Translation Table to work ahead). The phrase "Instruct Python to copy" indicates that you should use a Python command that converts an object of one type into another rather than writing a literal value in the new type. There is no need to write your variables in hexadecimal shorthand unless the problem specifies to do so.

19. ***Assign*** the number 1000_{10} to a variable. Instruct Python to copy the stored bits contained in this variable into a 16-bit Python **bytes** object. (The bytes object should store the same bits as the int internally.) Looking back at your notes, check that the bytes object has the correct values in it. ***Write*** the Python code you used here:

20. In Problem 8, you created the `bytes` object `b'2 Faced'` by simply typing it into Python. Now, create the `bytes` object by ***specifying*** the hexadecimal shorthand you found in Problem 1 in an `int` literal. For example, to store the hexadecimal shorthand 12 34 FF into a variable, you could type `i=0x1234ff`. Use an `int` literal like `0x`… rather than a bytes object like `b'\x__...'`. You should get an `int` object that, when you look at its bits in hex shorthand, stores your answer to Problem 8.

21. Next, instruct Python to ***copy*** the bits stored in a variable (in Problem 16) into a Python bytes object, just as you did with the number 1000 in Problem 15. ***Display*** the bytes object to check if it is `b'2 Faced'`. ***Write*** the Python code you used just for the transfer here:

22. In Problem 14, you found the hexadecimal shorthand for 1055. ***Pad*** this with zeros (if needed) to create a two-byte number and write it as a bytes object literal: `b'\x__ __\x__ __'`. Next, ***copy*** the contents of the bytes object back into an integer. (The `int` should store the same bits as the `bytes` object.) It should be the number 1055. ***Write*** the Python code you used here:

23. Consider that $1000_{10} = 03e8_{16}$. ***Circle the one*** bytes object literal that will be produced by the code. Do this by hand. (After you have committed to an answer, you can run the code – you can, of course, change your answer.)

    i = 1000
    i.to_bytes(2,'little')

    b. b'0001'
    c. b'1000'
    d. b'\x30\x8e'
    e. b'\xc0\x17'
    f. b'\xe8\x03'

The restrictions laid down for the previous problems are now lifted as conversions between formats may be needed for Problems 24 and 25.

24. The command `user_says = input('Please enter the length of the file')` will prompt the user to enter the length of a file, but it returns it as a `str` rather than an `int`. ***Write*** some python code to store the actual number as an `int` in the variable `file_length`.

25. Suppose `file_length = 100`. ***Write*** some Python code that produces the bytes object `b'File length: 100'`. The number 100 should change with the file_length variable. (You can concatenate bytes objects just like strings.)

26. Both `to_bytes()` and `encode()` produce a bytes object. ***Describe*** the difference between these methods.

27. (Excellent credit!) Write a Python implementation of the `str()` method that takes an `int` as the argument and converts it to the equivalent decimal `str`. Use only `to_bytes()`, `from_bytes()`, `encode()`, `decode()`, and arithmetic operators like < and +. Do not use str() the str, int, hex, format, etc. functions. ***Cite*** any external or internal sources, including hints received from other students.

28. (Just for fun!) Write a Python implementation of the `int()` method that takes an ASCII decimal `bytes` object and converts it to the equivalent `int`.. Use only `to_bytes()`, `from_bytes()`, `encode()`, and `decode()`, and arithmetic operators like < and +. Do not use str() or any other command like it in Python.