CS1110 Summer 2021 — Assignment 1[*]

Due Saturday 6/26 at 11:59pm

Read this before you begin

Group work and academic integrity You must work either on your own or with one partner. If you work with a partner, you must first register as a group in Canvas and then submit your work as a group. For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student's code and it is *certainly* not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, please seek help from the course staff.

Learning Goals The learning goals and outcomes for this assignment are listed below. Following each goal is a reference to where relevant material can be found in Canvas.

1. Use Python as a calculator to solve word problems (lectures 2 and 3)
2. Write function calls (lecture 3)
3. Use Python string formatting to print values (new in this assignment).
4. Implement a function according to its specification (lecture 5)
5. Design a function, including its specification (lecture 5)
6. Execute code by hand to illustrate memory representation (lecture 5).

Experiment It is important to develop a spirit of exploration. If you wonder how Python might deal with a particular kind of input, simply *try it out in the interactive mode*! Throughout the assignment we will ask you to *try* something or *look up* something. When you come across something that you don't know how to do, *treat it as the learning opportunity that it is.* Play with code ideas in the interactive mode, or refer to the course examples, or look up the documentation (start from the Python Documentation). Be ready to explore—that's learning! But of course, if you feel stuck after trying out several ideas, come to office hours—that's why we have office hours!

Restrictions Experiment, but there are limits. Since CS1110 is an introductory course that assumes no previous programming experience, and for the purpose of learning specific concepts, we sometimes restrict the use of certain tools or concepts that we consider too advanced given the contents and current status of the course.

For this assignment, you should only need the `math` module from the standard library. And even if you already know about *conditional* statements and *loops*, do not use them in this assignment.

Submitting *Do not* put your name in the files that you will submit; only include your NetID. The reason is to help promote fairness in grading. Thank you for heeding our request, even if it sounds a little strange.

Submission within 24 hours after the deadline will be accepted with a 10% late penalty. Even if only one file is submitted late, the assignment as a whole will be marked late. Assignment submission on Canvas closes 24 hours after the deadline and no further submission will be allowed unless a previous arrangement has been made.

You should submit the following files: `a1.py`, `conversion.py`, `test_conversion.py`, and a file with your drawings for Part B.

[*]Based on an earlier assignment by Ariel Kellison.

Part A: Coding in Python

For this part of the assignment, you will need the files `a1.py` and `conversion.py`. These are published with the assignment on Canvas. You will add code to both files.

A.1 Word problems [40 points]

In this part of the exercise, we are going to practice writing mathematical expressions in Python through a series of simple word problems. You will do these exercises by writing code in the file `a1.py` that we have started for you.

Getting started Run `a1.py` as a script (type `python a1.py` at the command line in the directory where you put the file; do not use the python interactive shell) and observe what happens. There should be no error. You goal is to keep it error-free as you add/modify code to answer the exercise questions. This means *frequently* saving and test-running the script with newly added code.

The skeleton provided gives print output but the values are not the correct answers. Your job is to change the code so that the printed values are correct. We provided some variables/values and the `print` statements as examples for how to *format* the printing of numeric values.

Before you dive in Let's first experiment with the source code and learn about string formatting.

1. Some `print` statements print a string that begins with `\n`. What is that? Experiment! Pick one `print` statement and add more `\n` (e.g., `\n\n\n`) to the string. Save and run the script. How is the output different? Do you have to put `\n` at the front of the string? What if you put it somewhere in the middle, or at the end?
2. The statement that prints your answer to the first exercise shows a new way to format output. Instead of gluing together strings with the + operator, it uses a *format-string* (or *f-string* for short) to combine variables into a string. This is done by using `f'` instead of a single apostrophe to start the string, and using curly braces around variable names. The `:.2f` part indicates that we want the answer to be printed with two decimal places. Experiment! Try changing the number of decimal places, and see how the output changes.
3. The statement that prints your answer to the second exercise shows how to format integers with some padding: the {`end_minute:02`} part says that we want to print `end_minute` and "pad" it on the left until it is two characters long. Experiment! What happens if you remove `0`? What happens if you change the `2` to `12`?
4. The statement that prints your answer to the third exercise again formats a floating point variable. We just learned how to pad the printed version of an integer variable with zeroes on the left. Try and find out if you can do the same for floating point variables. Experiment! What happens if you put `1` before `.2f` in the format string? What happens if you put `5` there instead? What does this tell you about the way left-padding works for floating-point numbers?
5. We have seen that you can put variables between curly braces in an f-string to substitute them in. Until now, we have also used formatting instructions (e.g. `:.2f`) to control the precision. Experiment! What happens if you remove the formatting instruction, including the colon?
6. Maybe there is even more to f-strings than what we have seen so far. Notice that we have only put variables between the curly braces. We know that function calls accept more than just single variables, we can put whole expressions in there as well. Maybe the same holds for f-strings too? Experiment! What happens if you replace `flour_to_buy` with `flour_to_buy+1` in the final f-string?

Solving the exercises Now you are ready to solve the textbook exercises one at a time. Write code to answer the questions *as stated*, i.e., you do not need to consider "other cases" or get user input. For example, exercise 1 asks you to compute the necessary grade for those specific inputs; you do not need to think about any other possible input numbers. On the other hand, do not calculate the answer separately and type it in literally—the code needs to run your expressions to get there. Each word exercise is worth 10 points.

A.2 Augment the `conversion` module [30 points]

Take a look in the `conversion.py` file provided for this exercise. It contains `to_centigrade`, which converts temperature in Fahrenheit to centigrade. You will extend this module by doing the following.

1. Write a function `to_fahrenheit` that converts a temperature from centigrade to Fahrenheit and returns it. Look up the formula online if necessary [3 points].

2. The Kelvin scale for temperature is used in many scientific disciplines. Write two additional functions for temperature conversion, looking up the formula online if necessary:

   - A function `centigrade_to_kelvin` that converts a temperature from centigrade to Kelvin.
   - A function `fahrenheit_to_kelvin` that converts a temperature from Fahrenheit to Kelvin. Note that you may use the functions that you have developed already.

   Write an appropriate docstring for both functions [12 points].

3. KerMetric time divides a day into 100 units of equal length, called *kermits*. For instance, noon (12:00:00) corresponds to 50 kermits, and 2:33:44pm corresponds to 61 kermits.

   The function `to_kermits` takes an hour, minute, and second, and should return the kermit that corresponds to that time, rounded to the nearest whole number. Implement this function, and give it a docstring.

   Hint: you may use the built-in `round` function to do the rounding [7 points].

4. Test your functions by putting some `print` statements in a new file, `test_conversion.py`, (in the same directory as `conversion.py`) and running it as a script. Remember to `import` the `conversion` module, or your functions will not be available!

   Try to think of at least two sensible testing inputs for each function, and justify why you want to test these values specifically in a comment [8 points].

Part B: Memory Representation [30 points]

Trace the execution of the script on the bottom of this page by drawing the memory representation as well as writing out the print output, if any. Write the print output that would result from executing the code as a script, not in the Python Interactive Shell. The three things you need to include in your drawing (if applicable) are global variables, call frames, and print output.

When you need to cross out an item (e.g., a value, a frame, a line number) in order to indicate that that item is updated or removed from memory, cross it out *neatly* using one diagonal line *so that the original values are visible.*

When drawing a call frame, use the line numbers given in the script below. For example, the first line of the function body of `foo` to be executed is line number 2.

Your solution must be uploaded to Canvas as a single PDF file. You may use the PowerPoint template provided with the assignment, or another drawing tool to create digital drawings.

If you prefer, you can also draw your solution by hand, but please make sure that the scan is of good quality. **Assignments that are not legible will not be graded.** If you do not have a scanner at home, there are several apps available for your phone that can take a picture of a piece of paper and enchance it.

Your submission must be less than 100MB in size, and ideally less than 10MB. If you scanned your work and the file is too large, try lowering the DPI (dots-per-inch) setting on your scanner.

Submit the completed file on Canvas *after* registering your group.

```
 1  def laurel(a, b):
 2      c = a
 3      b = (c - b) / 2.0
 4      return b
 5      d = 1
 6
 7
 8  def hardy(a):
 9      b = 4 ** a
10      print(b)
11
12
13  a = -1
14  c = 7
15  a = laurel(a,c)
16  hardy(c)
```