CSCI 1133, Fall 2019
Programming Assignment 2
Due: 11:55pm, Wednesday September 18, 2019

Due Date: Submit your solutions to GitHub by 11:55 p.m., Wednesday, September 18th. We will do a pull from this time point. Do not upload anything to Canvas and PLEASE be sure to use proper naming conventions for the file, classes, and functions. We will NOT change anything to run it using our scripts.

Unlike the computer lab exercises, this is not a collaborative assignment. You must design, implement, and test your code on your own without the assistance of anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class (so you are ONLY allowed to reuse examples from the textbook, lectures, or code you and your partner write to solve lab problems). Otherwise obtaining or providing solutions to any homework problem for this class is considered Academic Misconduct. See the syllabus and read section "Academic Dishonesty" for information concerning cheating. Always feel free to ask the instructor or the TAs if you are unsure of something. They will be more than glad to answer any questions that you have. We want you to be successful and learn so give us the chance to help you.

Instructions: This assignment consists of 3 problems, worth a total of 40 points. Solve the problems below by yourself, and put all functions in a single file called `hw2.py`. Use the signatures given for each class and function. We will be calling your functions with our test cases so you must use the information provided. If you have questions, ask!

Because homework files are submitted and tested electronically, the following are very important:

- You follow all naming conventions mentioned in this homework description.
- You submit the correct file, hw2.py, through Github by the due date deadline
- Your file is contained in a folder labelled hw2
- You follow the example input and output formats shown.
- Regardless of how or where you develop your solutions, your programs should execute using the python3 command on CSELabs computers running the Linux operating system.

Push your work into Github under your own repo. The specific hosting directory should be: repo-<username>/hw2, where you replace <username> with your U of M user name. For instance, if your email address is bondx007@umn.edu, you should push your hw2 to this directory: repo-bondx007/hw2, meaning that the path to your file is `repo-bondx007/hw2/hw2.py`

The following will result in a score reduction equal to a percentage of the total possible points:

- Not filling out the documentation template for every function, other bad code style (10%)
- Incorrectly named/submitted source file, functions, or classes (20%)
- Constraints not followed (40%)
- Failure to execute due to syntax errors (30%)

Use the following template for EVERY function that you write (note: a helper function is a function so it gets a template.)

```
#========================================
# Purpose: (What does the function do?)
# Input Parameter(s): (Each parameter by name and what it represents)
# Return Value(s): (What gets returned? Possibilities?)
#========================================
```

Problem A. (10 *points*) **Relativistic Length Contraction**

Einstein's Theory of Special Relativity tells us that distances between objects moving relative to our inertial reference frame are contracted (shortened). The upshot of this is that it means that if we put an astronaut on a spaceship travelling at 99% of the speed of light to Alpha Centauri, which is 4.37 light-years away, it will take about 4.41 years according to observers on Earth, but only about 0.62 years according to the astronaut. This is because Alpha Centauri and Earth would both be moving at about 99% of the speed of light relative to the astronaut, so the distance between the two objects would be contracted greatly from the astronaut's perspective.

If this gives you a headache, don't worry about it: you don't actually need to understand relativity to do this problem. Write a function called `length_contract(dist, speed)`, which takes in two numerical values as parameters: `dist` is the original distance (in meters) between two objects, and `speed` is the speed (in meters/second) at which we are travelling relative to the two objects. The function **returns** (not prints) the contracted distance between the two objects from our reference frame, given by the formula:

$$L = L_0\sqrt{1 - \frac{v^2}{c^2}}$$

where L_0 is the original distance, v is our speed, c is the speed of light (approximate this as 300000000 meters/second, or $3 * 10^8$), and L is contracted distance.

We're ignoring several things here, such as the fact that the objects could be moving relative to each other (especially if they're two different star systems), the time required to accelerate and slow down, and gravitational effects from General Relativity, but this isn't a terrible approximation of how it would actually work.

Hint:

- You can find the square root of x without importing math by using the built-in exponent operator: math.sqrt(x) == x**0.5

Constraints:

- Do not import/use any Python modules.
- Do not use the input() function,
- Your submission should have no code outside of function definitions (comments are fine)

Examples:

```
>>> length_contract(100, 150000000)
86.60254037844386

>>> length_contract(30000, 0)
30000.0

>>> length_contract(0, 200000000)
0.0

>>> length_contract(4.37, 297000000)
0.6164643623113996

>>> length_contract(1000000, 299999999)
81.64965807128421
```

Problem B. (10 *points*) **The Bessel Run**

Few people realize that astronomer Friedrich Bessel is not only still alive, but is currently working as an intergalactic smuggler pilot, running some contraband materials through a particularly dangerous (and very large) region of space. The run is 12 parsecs long, as measured by some "stationary" observer (there's a reason I put that in quotes, but this isn't a Physics class so we're ignoring it). A parsec is a unit of distance equal to about 3.26 light-years. Bessel intends to fly his spaceship at close to light speed, so that the path will appear much shorter from his perspective due to relativistic length contraction.

Write a function `bessel_run(speed)`, that takes a single numerical value speed, representing Bessel's average speed in the run, given in meters/second.

The function must **print** out the time required to traverse the segment, as seen by a stationary observer, in years (this is just distance / speed, no relativity needed).

The function must **return** the time required to traverse the segment, as seen by Bessel, in years (use the `length_contract` function from problem A, and then compute contracted distance / speed).

You'll need to convert from parsecs to meters and from seconds to years to do this correctly:

- 1 parsec = $3.086 * 10^{16}$ meters
- 1 year = 31557600 seconds (using astronomical year, not calendar year)

Note that because we are using approximations, so long as your answers are within about 1% of the correct value, we'll accept them.

Try to roughly match the print format in the examples below (you won't lose points so long as it is reasonable, but it does make it easier to grade).

Hint:

- You may want to consider writing helper functions to do some of the tasks like unit conversions. Remember, you must write documentation for all functions you write, including helpers.

Constraints:

- Do not import/use any Python modules.
- Do not use the input() function,
- Your submission should have no code outside of function definitions (comments are fine)
- You must call the `length_contract` function from problem A within `bessel_run`.

Examples (value in **bold** is returned, text in *italics* is printed):

```
>>> bessel_run(200000000)
58.71385083713851
43.76272056420821

>>> bessel_run(88)
133440570.08440569
133440570.08439995

>>> bessel_run(299999999)
39.14256735523423
0.0031959772405870867

>>> bessel_run(150000000) + bessel_run(100000000)
78.28513444951801
117.42770167427702
178.50881404267247
```

Problem C. (20 *points*) **Who needs loops?**

Your friend Ester, who took this class last semester, is taunting you. She says that because you haven't learned loops yet, you would need more than 100 lines of code to print out 100 lines of text. Prove her wrong.

Write a function called `print_100()` with no parameters that prints out the string "Who needs loops?" exactly 100 times in a row, once per line, subject to the constraints below. This function does not return anything.

Hints:

- You will need to write more than one function for this problem. In fact, it is very difficult to do this problem without at least three.
- Make a function that prints out "Who needs loops?" a reasonable number of times, and then think about what would happen if you called that function multiple times inside another function.

Constraints:

- You can write a maximum of **20** lines of code for this problem. This includes all functions you write for this problem (including the line for the function definition), but does not include comments, blank lines, or the functions for problems A and B.
- Do not import/use any Python modules
- Do not use any loops (for or while)
- Do not use the newline character `'\n'`
- Do not use the input() function
- Your submission should have no code outside of function definitions (comments are fine).
- Remember, you need to document every function you write

Example:

```
>>> print_100()
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
```

Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?

Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?
Who needs loops?