Lab 3: Help!

Objectives

- More practice writing *good* helper functions, and refactoring to produce better helper functions.

Preliminary Setup

1. Create a new PyCharm project.
2. Download the starter code and an input file:
    - **badmain.py**
    - **input.txt**
3. **Make a copy of badmain.py, calling it goodmain.py**. You'll be making changes to **goodmain.py** so that it is improved over **badmain.py**.

Have a look around

Now that you have a project and some code, explore to see what you have:

1. Skim the code. There is some main-line code and several helper functions. Note that some of the for loops use Python's **enumerate** function. See here for info.
   Note: don't let yourself get lost in the parts you don't understand.
2. Run the code. What happens? Make sure you put the input file in the right place.

Fix it

The main task of this lab is to improve the helper functions (and the code that calls them) in several ways. By the end of the lab, **goodmain** should do all that **badmain** does, but it should be better written.

In all of the following steps, **change goodmain** while leaving **badmain** unchanged.

Step 1: Remove uses of global variables.

1. In each of the functions, carefully look at each use of a variable. Determine if the variable used is one of the parameters to the function, a local variable, or a variable defined in

the global scope. For any function that uses globals, rewrite the function so that it no longer uses globals:
    - There are two main ways to remove uses of globals. If the global is being used to give the result of the function, perhaps it can be returned instead? If the global is being used to give input to the function, perhaps we can add it as a parameter to the function?
    - Feel free to change the names of variables. You might find it easier to make the locals and parameters have different names than the globals.
    - Rewrite the ***calls to the function*** so that the program still works.
2. **Note:** don't just search for the 'global' keyword. Functions can still use global variables without those variables being explicitly marked as 'global'.
3. Make sure that **goodmain** still gets the same answers as **badmain**.

Step 2: Move the main-line code

1. Create a **main** function that contains the main-line code, and make new main-line code that calls the new function:
```
if __name__ == '__main__':
    main()
```
2. Make sure that **goodmain** still gets the same answers as **badmain**. **Note**: if the program crashes or has different behavior now, that probably means you missed a use of a global variable in the previous step.

Step 3: Add appropriate documentation

1. For each of the functions, add a docstring immediately after the function signature. Recall that a Python docstring should be of the form:
    does this, returns that
    In addition to the short description, if the function has parameters, or returns something, you should have **:param:** and **:return:** tags, respectively.
    Make sure each of your docstrings explains the **behavior** (i.e. *what* the function does), *not* the **implementation** (i.e. *how* it does it).
2. Make sure that **goodmain** still gets the same answers as **badmain**.

Step 4: [Extra Credit] Abstract some helpers

1. Several of the helpers all have the same logic. Improve the code by deciding which helpers can be combined and then replacing them with only one (that might be called in different places). You'll need to change the calls to those functions.
2. Make sure that **goodmain** still gets the same answers as **badmain**.

How to turn in this lab

Before turning in any program in this class, remember this mantra:

Just because it works doesn't mean it's good.

Part of your grade will also come from things like how understandable and readable your code is. You will also be graded on the neatness, presentation, and style of your program code.

Turn in the lab via gradescope --- upload the code files and any supporting files needed to run them.

Ask for help if you're having problems!