Programming on Purpose (CSC 120)
Winter 2024

Lab 3 – Vanquish the Evil (Global Variables)

1/18/2024

Objectives

- Practice writing behavior-focused docstrings
- Refactor to avoid global variables
- Practice writing good helper functions

1 Preliminary Setup

1. Find your assigned partner.
2. Create a folder for this lab. (Not in the Downloads folder.)
3. Download the starter files and save them into the folder you just created for this lab.
4. Open the folder in Visual Studio Code.
5. Copy `badmain.py` to `goodmain.py`. You'll be making changes to `goodmain.py` so that it is improved over `badmain.py`.

2 Have a look around

`goodmain.py` isn't so good right now. It uses global variables and, as we've discussed in class, global variables are evil. You're going to fix this by ridding this file of all variables that have global scope. But first, study the code. Trace it to predict what it will do. If you see code that doesn't make sense, stop and figure it out by reading documentation (linked from our Nexus page) or by asking me. In particular, notice that this program contains examples of different ways of looping over a list (plain for-loop, for loop with `range`, for loop with `enumerate`) and of list comprehensions (e.g. `get_back_slash` and `get_forward_slash`). Make sure you understand how these constructions work.

Only after you've studied the code should you run it. Make sure you understand the output based on the input file. Try changing the data in the input file to see how the output changes

3 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: Add appropriate documentation

Add docstrings to the module and each function definition.

For each of the functions, add a docstring immediately after the function signature. Recall that a Python docstring should be focused on behavior and of the form:

> Do this, return that.

When the function name, parameters, and behavior are pretty obvious, a single line is enough as documentation. But sometimes more explanation is needed. In that case, use the following format.

```
Short, single line summary formulated as a command.

More elaborate description, if necessary.

Explanation of parameters and return value.
```

There is no fixed format for the explanation of parameters and return values. One popular format is the one used by numpy. For example:

```
Parameters
----------
left_pips : int
    the left number
right_pips : int
    the right number
Returns
-------
tuple
    a domino tile represented as a tuple of two integers
```

Make sure each of your docstrings explains the *behavior* (i.e. what the function does), not the *implementation* (i.e. how it does it).

Make sure that `goodmain` still gets the same answers as `badmain`.

Step 2: Destroy the Evil (global variables)

Carefully look at each use of a variable and determine whether it has local or global scope. For any globals, rewrite the code so that there are no more gobal variables.

To do this, you'll need to add parameters, return statements, and a main() function. You'll need to change the function calls too. Remember, a variable does not need to be declared with the "global" keyword in order for it to have global scope! After refactoring, run `goodmain.py` to make sure it still gets the same output as `badmain.py`.

- 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.
- Re-write the calls to the function so that the program still works.

- PITFALL ALERT: Removing global variables is one place where you might change a function's behavior and not just its implementation. For example, a function may return something now where it didn't used to. Because of this, some of your docstrings may now be incorrect. Analyze all docstrings and make changes where appropriate so they reflect the new behavior

Step 3: Better helper functions.

Several of the helpers all have (almost) the same logic. Improve the code by replacing those helper functions with one. (You'll need to change the code where those functions are called, too). Make sure that `goodmain` **still gets the same answers as** `badmain`.

4 Revise your code

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.

Make sure that all modules and functions are documented (even those that you didn't write).

Don't forget to cite who you received help from and include the honor code affirmation in the docstring of each module that you wrote or modified:

```
I affirm that I have carried out my academic endeavors
with full academic honesty. [Your Name]
```

5 How to submit

Submit the project by uploading `goodmain.py` to Gradescope.

Make sure to add your partner to the submission so that you both have access to it through Gradescope.