Lab 2: Breaking Bad (Code)

Objectives

- Practice making test cases and test suites
- Get more practice writing helper functions

Preliminary Setup

1. Create a new PyCharm project.
2. Download the starter code (from the google drive folder) and input file and add it to your project.

Have a look around

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

1. Read the code. Can you figure out what each line is doing? If there are functions you don't know, look them up at the documentation official python documentation site.
2. Run the code. What happens? Is the output correct?

Add a new test

1. Let's test the **get_winner** function some more by adding a second test. Create a new file "input2.txt" with a different board state than the one given in "input.txt". Make sure the two boards have different winners.
2. Add code to **main()** so that your input2.txt is tested IN ADDITION TO input.txt (i.e. don't remove the test using input.txt). Be sure you know what the correct answer is **before** you run the code.

Refactor and automate the tests

Notice two things:

1. You have the same or very similar code repeated twice in order to test with both input boards.
2. We need to read and interpret the output to know if the program worked right; if it says X wins, we need to confirm that X should have been considered the winner, and this requires ***manually*** looking at the input file and requires ***thinking***.

It would be nice if a) we didn't have repeated code and b) we could have the program tell us whether it works without having to interpret the results.

Let's fix this by extracting a helper function to run a single test and tell us if it passes.

1. Create a function in `main.py` called **confirm_result** that takes 2 parameters:

    a. A representation of a tic tac toe board and

    b. an indication of who should be declared winner (either a string 'X' or 'O' or the special value `None` -- note that `None` and ‘None’ are not the same thing).

    In other words, **confirm_result** takes the a parameter for **get_winner** (the board) and the answer that **get_winner** is supposed to return (i.e. the expected result). These two together form a **test case**.
2. Write the code for **confirm_result**. It should test **get_winner**, print information about what is being tested, and print "PASS" if **get_winner** got the right answer (i.e. if the actual results from **get_winner** matches the expected results) and print “FAIL” otherwise.
3. Re-write **main()** so it uses your new **confirm_result** instead of calling **get_winner** directly. All printing should be removed from **main()** because **confirm_result** will do the appropriate printing. **Notice how much shorter** and clearer **main()** has become.
4. [Extra credit] Re-write **main()** so it does not read the boards from files (hard-code the boards directly into the python source instead). Now, add more tests to **main()** using **confirm_result** as appropriate.

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 on gradescope by 4pm the day after lab. Note that the autograder will run your lab (by running main.py) and save the output for my review -- the autograder will not directly award any points.

Ask for help if you're having problems!