Programming on Purpose (CSC 120)
Winter 2024

Lab 2 – Tic-Tac-Test

1/11/2024

Objectives

- Practice making test cases
- Practice writing helper functions
- Also: some practice debugging and refactoring

1 Preliminary Setup

1. Find your partner. **Remember to practice good pair-programming throughout this lab. That means: Talk to each other and switch roles often.**
2. Create a folder for this lab. **Note: Do not create this folder in your Downloads folder. The Downloads folder is not intended to be used for long-term storage of files that are important or as your main workspace. Instead, create a folder for your class in your home directory or on the desktop.**
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.

2 Have a look around

Now explore to see what you have:

1. You should have two modules. Can you figure out what the main purpose of each one is and how they interact?
2. Read the code. Can you figure out what each line is doing? If there are built-in functions you don't know, look them up in the official Python documentation linked on Nexus.
3. Run `test_tictactoe_board.py`. What happens? Is the output correct?

3 Add a new test

Let's test the `get_winner` function some more by adding a second test case to `test_tictactoe_board.py`. When you are done, your code should run both tests: the one that was given to you in the starter code as well as the new one that you added. Be sure you know what the correct answer is for each of these tests. Is `get_winner` behaving correctly?

4 Refactor and automate the tests

Notice that you have the same or very similar code repeated twice in order to test with both input boards. Also notice that 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 looking at the board configuration 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 called `confirm_result` that takes 2 parameters: a board and an indication of who should be declared winner (either a string 'X' or 'O' or the special value `None`). In other words, `confirm_result` takes an input for `get_winner` (the board) and the answer that `get_winner` is supposed to return (i.e the expected result). The two parameters form a test case.
2. Write the code for `confirm_result`. It should test `get_winner` and print "PASS" if the winner returned by `get_winner` is the same as the expected winner. If `get_winner` returns an incorrect answer, the function `confirm_results` should print "FAIL". To help with debugging, it should also print out the board as well as the actual and the expected answer.
3. Re-write the test functions (`test_1` and the second test function that you added) so that they use your new `confirm_result` instead of calling `get_winner` directly. All printing should be removed from the test functions because `confirm_result` will do the appropriate printing. Notice how much shorter and clearer the test functions have become.
4. Now, add more tests using `confirm_result`. Your goal should be to cover all possible different types of scenarios that `get_winner` may have to deal with. Does `get_winner` work correctly?

   At some point, you should realize that `get_winner` doesn't work. That's deliberate. I want you to see what a FAILed test looks like (so put the bug back if you've already gotten rid of it so you can see a test FAIL). Once you've seen the failed test, debug it: i.e. locate the bug and fix it.

5 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]
```

6 How to submit

Submit the project by uploading both Python files (`tictactoe_board.py` and `test_tictactoe_board.py`) to Gradescope.

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