COMP 1510 Programming Methods Lab 08

Christopher Thompson
cthompson98@bcit.ca

BCIT CST — November 2019

Welcome!

This lab is due at the end of your lab period.

Last week I released A3. Have you played aardwolf? Have you developed some ideas? This week's lab will help you get started. You'll build some of the infrastructure you need for a successful text-based adventure game.

1 Grading

Figure 1: This lab is graded out of 5

This lab will be marked out of 5. For full marks this week, you must:

1. (3 points) Correctly implement the coding requirements in this lab.
2. (1 point) Correctly format and comment your code.
3. (1 point) Correctly generate unit tests for your code.

2 Project Setup

Please complete the following:

1. For your eighth lab, all files must go into the Lab08 folder.
2. After you complete each task, commit and push your change to version control. In order to earn full marks, you must commit and push after each function is complete. Your commit message must be a short message to me that describes what you completed for this commit, i.e., "deconstructed base_conversion", or "debugged function_name", etc.
3. Since I am already a collaborator I will pull your future work automatically.

3 Requirements

Please complete the following:

1. Implement the following functions. The name you must use is below. Do not modify the name I have provided.

2. Include correctly formatted docstrings that include comprehensive doctests (where possible) for each function. These docstrings must explain to the user how the function is meant to be used, and what the expected output will be when input is provided.

3. Each function must also be tested in a separate unit test file by a separate collection of unit tests. Ensure your test cases and functions have descriptive, meaningful names, and ensure that every possible valid input is tested for each function.

4. Ensure that the docstring for each function you write has the following components (in this order):

   (a) Short one-sentence description that begins with a verb in imperative tense

   (b) One blank line

   (c) Additional comments if the single line description is not sufficient (this is a good place to describe in detail how the function is meant to be used)

   (d) PARAM statement for each parameter that describes what it should be, and what it is used for

   (e) PRECONDITION statement for each precondition which the user promises to meet before using the function

   (f) POSTCONDITION statement for each postcondition which the function promises to meet if the precondition is met

   (g) RETURN statement that describes what the function returns

   (h) One blank line

   (i) And finally, the doctests. Here is an example:

```
def my_factorial(number):
    """Calculate factorial.

    A simple function that demonstrates good comment construction.

    :param number: a positive integer
    :precondition: number must be a positive integer
    :postcondition: calculates the correct factorial
    :return: factorial of number

    >>> my_factorial(0)
    1
    >>> my_factorial(1)
    1
    >>> my_factorial(5)
    120
    """
    return math.factorial(number)
```

5. Let's model a simple game. The user controls a character who starts in the upper left hand corner of a playing board. The goal of the game is to move the character to the bottom right corner of the board. We can pretend they are in a maze or a crowded room, even though there are no walls or dead ends. That's it!

6. Start by creating a module called maze.py. Inside maze.py, create a main function and our standard if statement that instructs the Python runtime to execute the main function if the module is executed, not imported.

7. The main method must call the game method. Yep, that's it!

8. The game method will do a few things:

   (a) Create a board and assign it to a variable called board. The board must be a 5 x 5 grid. Each location in the board should store its coordinates. Assume the user begins in the upper left hand corner which is (0,0), and the user seeks to move to the bottom right hand corner which is (4, 4).

(b) Create a character and assign it to a variable called character. A character only needs to store its current coordinates.

(c) Create an infinite loop. Use while loop. For the sentinel value, let's assume the game will loop until the user reaches their goal. How might you check this? Perhaps a boolean called reached_goal?

(d) Inside the loop you must do four things:

i. Tell the user where they are

ii. Ask the user whether they wish to move up down left or right

iii. Validate their choice and, if the move is valid, move the user's character. If the move is invalid, print a suitable message.

iv. Check if the user has won.

(e) To validate the move, create a separate function called validate_move. This function must accept the board, the character, and the user's move choice. Use the information passed in the parameters to determine if the move is in fact allowed. Do not let the user leave the board. If the move is valid, return True, else False.

9. That game method seems really big. How can be refactor it so it is a collection of functions that work together? We have already extracted a move validation method. Should we create a move_character method? (Hint: yes). What about a get_user_choice method? We want the game method to ultimately look a little like this:

```
def game():
    board = make_board( )
    character = make_character( )
    found_exit = False
    while not found_exit:
        // Tell the user where they are
        direction = get_user_choice( )
        valid_move = validate_move(board, character, direction)
        if valid_move:
            move_character( )
            found_exit = check_if_exit_reached( )
        else:
            // Tell the user they can't go in that direction
    // Print end of game stuff
```

10. That's it. Not so hard, was it?