Kristen Tan
Professor Kempinski
SSW-567
02 September 2018

Homework 01

Deliverable 3

- What challenges did you encounter with this assignment, if any?
    - The task to be accomplished by the method, classifying triangles as equilateral, isosceles, or scalene as well as stating if they are right triangles, seemed simple mathematically. When implementing the method as a Python function, though, I did not necessarily find myself having difficulty figuring out how to perform the act of classification in general, but I did find myself overlooking certain potential parameters for the method. These oversights caused me to have to go back and implement additional functionality/ correct existing functionality. For instance, the first time I wrote the function, I did not account for the possibility of decimal sides forming a right triangle, and as a result, I had to go back to my comparison using the Pythagorean theorem and round ($a^2 + b^2$) and round c^2 before checking the two for equality. Furthermore in regard to right triangles, I forgot to ensure that my method would recognize a triangle as a right triangle even if the sides were not entered in the order (leg, leg, hypotenuse). Thus, I had to add that recognition capability to my code when I saw that it was not working properly during testing.
    - Testing was a bit more challenging for me, as I have very little experience in this area. I defined six sets of tests to ensure that the classify_triangle() function worked properly under a number of different circumstances. In each of these tests. I called that classify_triangle() function with values, and compared it to the expected result using assertEqual() or an unexpected result using assertNotEqual() in order to demonstrate the functionality of the counterpart of assertEqual(). However, I was a bit unsure as to whether this was the most efficient way to test the function or if it would be more efficient to somehow incorporate the calls to classify_triangle into the main or another function to simply run classify_triangle() with varied arguments. However, in considering this approach, I did not see any great advantage over the approach to testing that I took, and I in fact found the approach I took to be simpler, as it allowed me to see and understand each individual set of parameters I was testing and see exactly where any errors were occuring based on the output of the test script.
- What did you think about the requirements specification for this assignment?
    - I believe that, while the requirements given provide a good idea of what the user wants, they fail to detail some important information about how the function should perform. A key functional requirement that is missing is to check if the three sides entered can indeed form a triangle (can easily be accomplished by using the Triangle Inequality Theorem, which states that the sum of any two sides must be greater than the value of the third side). Furthermore, if the sides do not

form a triangle and cannot be classified, what output should the function return? Should a String error message be displayed to the user?
Also missing is an answer to the question, how precise must the system be when classifying a triangle as a right triangle? If the legs and the hypotenuse are decimal values, there is potential for some discrepancy in the decimal place values of the result of $a^2 + b^2$ when compared with c^2. Though precision can be classified as a nonfunctional requirement, it is still a critical piece of requirements information to have when writing a program to accomplish a goal. In this homework, I arbitrarily decided to round ($a^2 + b^2$) and c^2 to two decimal places before comparison. This appeared to produce results that reasonably rationalized when the squares of two decimal values for the legs produced a sum, that, when possibly truncated, was "equal to" the square of the hypotenuse, when also possibly truncated. However, this is not to say that 2 decimal places is the best choice for rounding. In reality, precision should most definitely have been specified in the requirements.

- What challenges did you encounter with the tools?
    - Thankfully, I did not face any major challenges with the tools. I have programmed in Python during previous classes and projects. I developed my program in Eclipse, and I have also used Eclipse with the PyDev plugin prior to this course, so I was comfortable utilizing that IDE. The tool newest to me was the Unit Test tool. However, it seemed fairly intuitive to use when writing tests to make comparisons. My only issue with this tool is that I do not necessarily know the functionality of all the different assert methods or when one might use them, but hopefully with more testing experience I will be exposed to these scenarios and uses in a hands-on manner.
- Describe the criteria that you used to determine that you had sufficient test cases, i.e. how did you know you were done?
    - Unfortunately, for such a homework assignment, more definite markers of when testing is complete are not necessarily available. User acceptance testing, for instance is not a viable practice in this scenario; there are no trouble reports open with this simple practice program being written; defect density for the remainder of the program is not a truly useful for such a short program; and projected reliability after launching is also irrelevant due to the limited scope of the class/ distinct border between this and a real world product scenario. Because these somewhat more solid indicators of completeness are missing, my criteria were to ensure that all lines of code were executed a minimum of one time and to test both integer and decimal inputs, as these are the two types of numbers that could be side values of a triangle. The goal of executing all lines of code at least once prompted me to test sides that would form each of the three different types of triangles- equilateral, scalene, and isosceles. For equilateral triangles, I performed tests on all integer value sides and, separately, all decimal value sides. For isosceles and scalene triangles, I performed tests on all integers, all decimals, and combinations of the two. This was the baseline. Essentially, it ensured that the function worked to classify the simplest types of triangle, whether or not they had whole number sides. I then wrote tests to validate the function's reliability in

detecting when a triangle was a right triangle in addition to being either isosceles or scalene (as equilateral triangles cannot also be right triangles by nature of their equivalent sides and equivalent angles, all of which are 60 degree angles). For the isosceles right triangles, I wrote tests for both triangles with integer value legs and tests for triangles with decimal value legs. For the scalene right triangles, I performed the tests on all integer value sides, all decimal value sides, and a combination of integer and decimal side values. Additionally, for both types of right triangles, I ran tests with a given triangle's legs input into the function in all possible orders in order to ensure that a right triangle was recognized, even if its hypotenuse value was not entered after its two leg values. Essentially, code coverage prompted me to establish testing all individual triangle types and all viable combinations of triangle types as an important criterion, and general knowledge of potential issues that can occur when comparing decimals in programming prompted me to establish testing both integer side values, decimal side values, and combinations of the two as a second crucial criterion.