Hussein Salami

S371192

PRT582 Software Engineering: Process and Tools

Software unit testing report

***Introduction:***

The goal of this project is to write a Python program to calculate the Scrabble score for any given word. We made the project using a method called Test-Driven Development (TDD), which essentially means writing tests before writing code, actually, to make sure everything works correctly.  
  
Python was chosen because it is a very user-friendly language with an excellent built-in testing system called unittest, which helps in testing the working aspects of the code. In following TDD, every single building block of this program was made sure to be tested and reliable before progression to the next step was considered, thereby helping create better and more dependable code.  
  
In this report following, we explain how exactly we designed the program, how we tested it, and finally, what we learned along the way.

***Objectives and Requirements***

The main goal of this project is the design of a program that should be able to compute any Scrabble score for any given word using official Scrabble rules, where all letters are given shown with official Scrabble value.

To satisfy this objective, the program must:

* Proper Scoring: The program must be able to accurately calculate the total sum by summing all letter values of a word.
* Case Sensitivity: The program should not be case sensitive, hence giving equal value to both uppercase and lowercase letters of the alphabet. For instance, "A" should score the same as "a".
* Word Check: It has to validate the user's inputted word as an actual word according to the existing dictionary.
* Timer Input: It should have an input form for setting a strictly limited time for user input of a word.
* Multi-Rounds: The program must have the feature of allowing the game to go on for various rounds as a score of the user is maintained from round to round in a new game.

The specifications guarantee that the program will be helpful for the users.

***Automated Unit Testing Tool***

The project utilized the unittest framework in Python of reliable and methodic testing. Unittest provides an effective and disciplined approach to writing and running tests, ensuring that any flaws are detected as early as possible during system development. The framework allowed for design test cases that tested individual pieces of code, ensuring that individual functions were working and delivering the prosecuted results accordingly.

Different test suites were developed based on the unittest framework, which focused particularly on improving the functionalities of the program, such as:

Score Calculation Test: testing the correctness of the program in calculating the score for different letters, of which the score value must be correctly summed.

Case Insensitivity Test: checking that both an uppercase and a lowercase letter are treated exactly the same regarding its value as part of the calculation in the score.

Validating Word Entries: It should be made sure that only dictionary-based valid words are accepted and in case of an invalid entry, it would appropriately prompt the user. Timing User Input: The timer has to be tested, such that the user gets the specified time to enter a word, and the score is adjusted with means to the speed in the response.

In this project, using unittest, careful verification was done on the accuracy and dependability of Scrabble Score regarding meeting laid-out requirements and functioning properly in a host of situations.

***Process: Test-Driven Development and Automated Unit Testing***

The Scrabble Score program follows the TDD approach. This approach puts immense emphasis in writing the tests before actually writing the code. This way, we could ensure that each function of the program was correctly implemented and met the requirements.

Given below are the steps to carry out TDD:

Write a Test: We began the process by writing a test case for the functionality with the unittest framework before writing the code to implement said functionality. For example, these test cases included getting the score with different word inputs correctly.

Run the Test and Fail: At this point our mostly empty implementation was being tested, so all these tests were expected to fail. The point was to make sure that the tests would detect the unimplemented or incorrect functionality.

We then implemented whatever code was required to make the tests passᅳsuch as the function that scored Scrabble words based on the individual letter values.

The next step was to run the tests, and sure enough, we now got a green light and the tests passed.

In this way, when the same process is followed for each individual feature of the program, we were able to develop the entire program in a systematic and efficient manner.

Use of the Automated Unit Testing Tool

The unittest framework had been crucial to the whole process of TDD. This is because it had facilitated the development of organized test suites, those which target individual areas of the program:

Score Calculation Tests: These verified if, indeed, the scores calculated were of correctness for many words.

Case Insensitivity Tests: These ensured that the calculation of the score was done irrespective of the case, i.e., with case insensitivity.

Word Validation Tests: It was ensured that only the words containing the dictionary could be passed into the program by validation tests.

Timer functionality tests: tests that the program, for instance, imposed the correct upper bound of time available for word entry and that scores are subsequently adjusted.

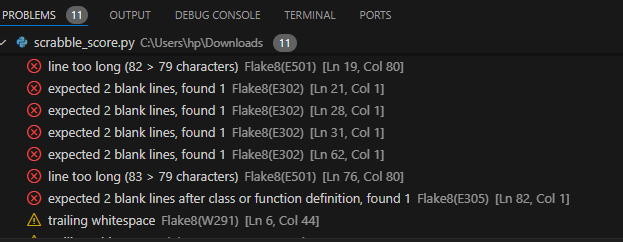
Such an iterative process is an exception for every block within the Scrabble Score program. This then made sure that every part is up and functioning how it should, which results in a dependable and robust application.

***Code Quality and the Linting Tools (Pylint and Flake8)***

The challenge was keeping code quality high, for which we employed Flake8, a Python linting tool that checks against PEP 8, the style guide of Python. Flake8 helped us spot and correct many problems regarding code formatting and style. The other powerful tool in terms of linting was Pylint. We ran Pylint on top of the already performed quality assurance mechanisms. It gives extensive feedback on code, comprising probable problems and their improvements.

**Flake8**

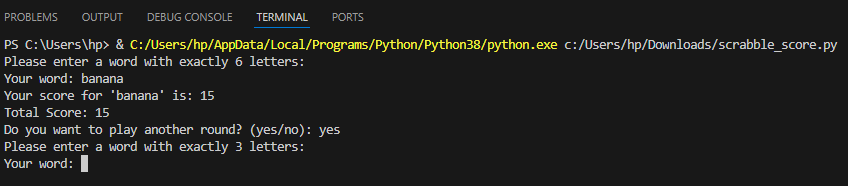
Below are the screenshots of tests run using flake8,



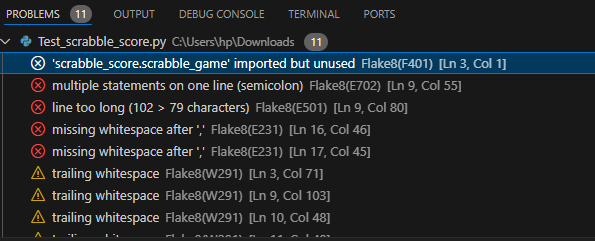
After the test, the following errors were encountered.

* **E501**: Line too long should be 79 characters
* **E302**: Expected 2 blank lines before the def main() function.
* **E305:** indicates that there should be two blank lines after the function or class definition.
* **W291:** indicates that there are unnecessary spaces at the end of a line of code.

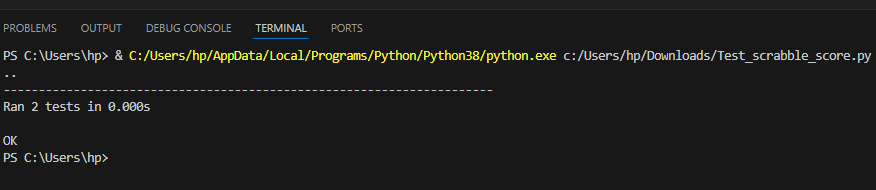
After resolving all the issues and running the flake8 test, the result looked like the screenshot below,



Similarly, several flake8 tests were run on file: test\_scrabble\_score to get the accurate code, which is demonstrated by the screenshot below,

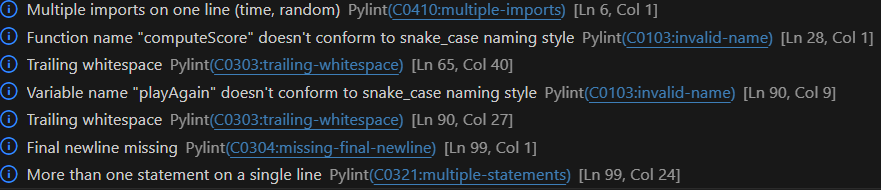


All the issues were figured out using flake8 and resolved.



**Pylint**

Pylint was also used to check the errors. Below are the screenshots of the tests and how I resolved the issues



**C0410 (Multiple imports on one line):** Split import time, random

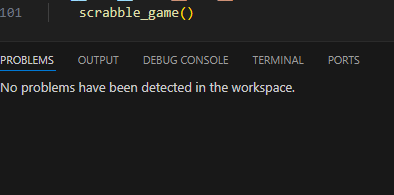
**C0103 (Function name doesn’t conform to snake\_case):** named computeScore instead of compute\_score.

 C0103 (Variable name doesn’t conform to snake\_case): named playAgain instead of play\_again.

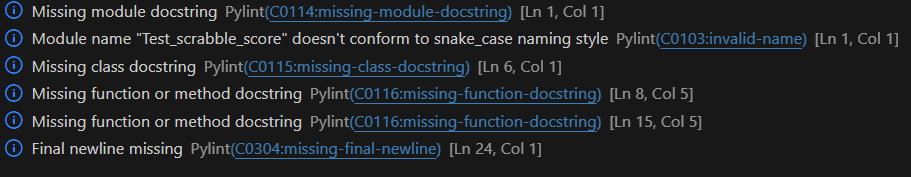
 **C0321 (More than one statement on a single line):** if \_\_name\_\_ == "\_\_main\_\_": scrabble\_game

C0303 (indicates that there are unnecessary spaces at the end of a line of code.)

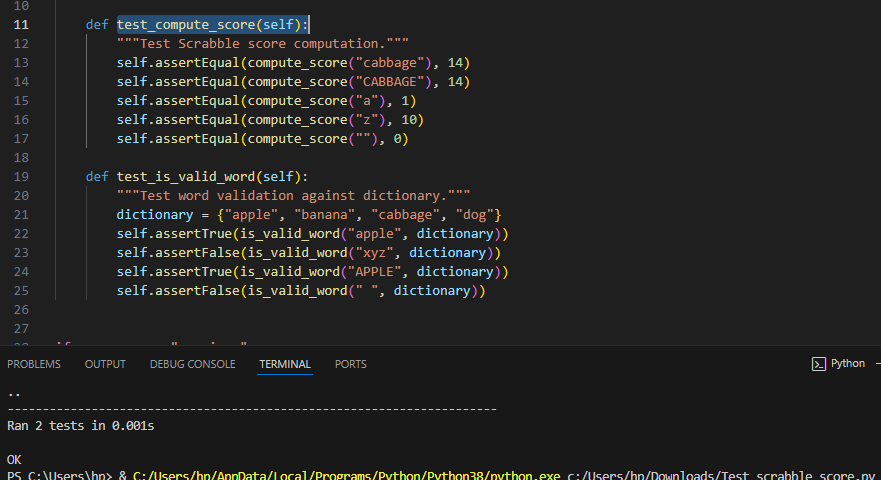
After resolving all the issues, the Pylint test result was obtained, as shown in the screenshot below,



Similarly, several Pylint tests were run on the ‘test\_scrabble\_score.py’ file to get the accurate code, which is demonstrated by the screenshot below,



After resolving all the issues, below is the screenshot of test\_compute\_score and test\_is\_valid\_word



***test\_compute\_score Method:***

Purpose: This method tests the compute\_score function to ensure it correctly calculates the Scrabble score for different words.

How It Works:

It calls compute\_score with different word inputs and compares the result with the expected score using self.assertEqual.

Test Cases:

* "cabbage" should have a Scrabble score of 14.
* "CABBAGE" (uppercase) should also score 14, ensuring that the function is case-insensitive.
* "a" should return a score of 1.
* "z" should return a score of 10.
* An empty string "" should return a score of 0.

***test\_is\_valid\_word Method:***

Purpose: This method tests the is\_valid\_word function to check if it accurately determines whether a word is valid based on a given dictionary.

How It Works:

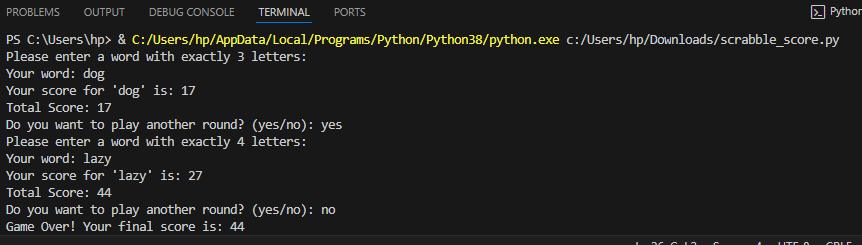
It uses self.assertTrue and self.assertFalse to verify that the function returns the correct Boolean value (True or False) for different word inputs.

Test Cases:

* "apple" should be recognized as a valid word (returns True).
* "xyz" should be identified as invalid because it’s not in the dictionary (returns False).
* "APPLE" (uppercase) should still be recognized as valid, testing case insensitivity (returns True).
* " " (a space) is not a valid word and should return False.

In summary, test\_compute\_score checks that the score calculation is accurate, and test\_is\_valid\_word ensures that word validation against a dictionary works correctly.

Below is the final output of the game,



***Conclusion:***

This Scrabble Score game implementation is a good skeleton for any word-based game; it shows clean, maintainable code with good testing practices. It is a good balance between functionality and simplicity, thus serving as a good example of how Python can be applied in order to solve real-world problems within a game context. Because of unit tests, the code is reliable and ready for expansion on-whether it be adding more features or fitting into a larger game system.

Several major lessons developed along the course of the project:

Test-Driven Development is effective, because for every part of this program, tests had to be written beforehand, thereby guaranteeing their correct implementation and whether they met requirements. This minimizes bugs and makes the development process much smoother.

The Value of Automated Testing The transparent use of the unittest framework has allowed us to carry out a systematic check on our code regarding its functionality. The automated tests enabled us to catch the mistakes that, under other conditions, could have disrupted the operation of our program just as it was supposed to from variant conditions.

Linting tools like Flake8 and Pylint played a very important role in helping to maintain good quality in code. This helped to fix several problems with the coding style, making the code was more readable, maintainable, and followed best practices.

Scrabble Score Balancing Simplicity with Functionality: The final implementation of the Scrabble Score game was a go-ahead with the simple but well-tested codebase, though providing strong functionality. It is in this balance that software development ensures a product is effective and easily maintained.

In summary, it has been shown that the base for a thoroughly tested and the consequent attention to coding detail is a reliable and expandable program.