

Software Unit Testing Report

[Github](#)

Introduction

This project aims to create a Python programme that, while following certain game rules and specifications, calculates the Scrabble score for a given word. In addition to calculating the user's Scrabble score based on letter values and prompting the user to input words within a set time limit, the programme also verifies that each word entered is present in the English dictionary. Because of its ease of use, readability, and accessibility to tools like unittest for testing and enchantment for dictionary validation, Python was selected for this project. We are using the TDD approach, where unit tests are written before implementing the actual functionality. Python's integrated unittest framework is used to automate the testing procedure, verifying user interactions, score computation, and input validity to make sure the programme runs as intended. This approach helps identify faults early in the development process and gives confidence in the quality of the code.

Process:

The Test-Driven Development (TDD) methodology, which entails creating test cases prior to implementing the real code, was applied in this project. This guarantees that the functionality is accurately and progressively implemented.

Step 1: Writing the Test Case :

Initially, we created a test case in `scrabble_test.py` to verify that the software accurately determines the word "cabbage"'s Scrabble score, which is supposed to be 14 points.

```
scrabble_test.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help

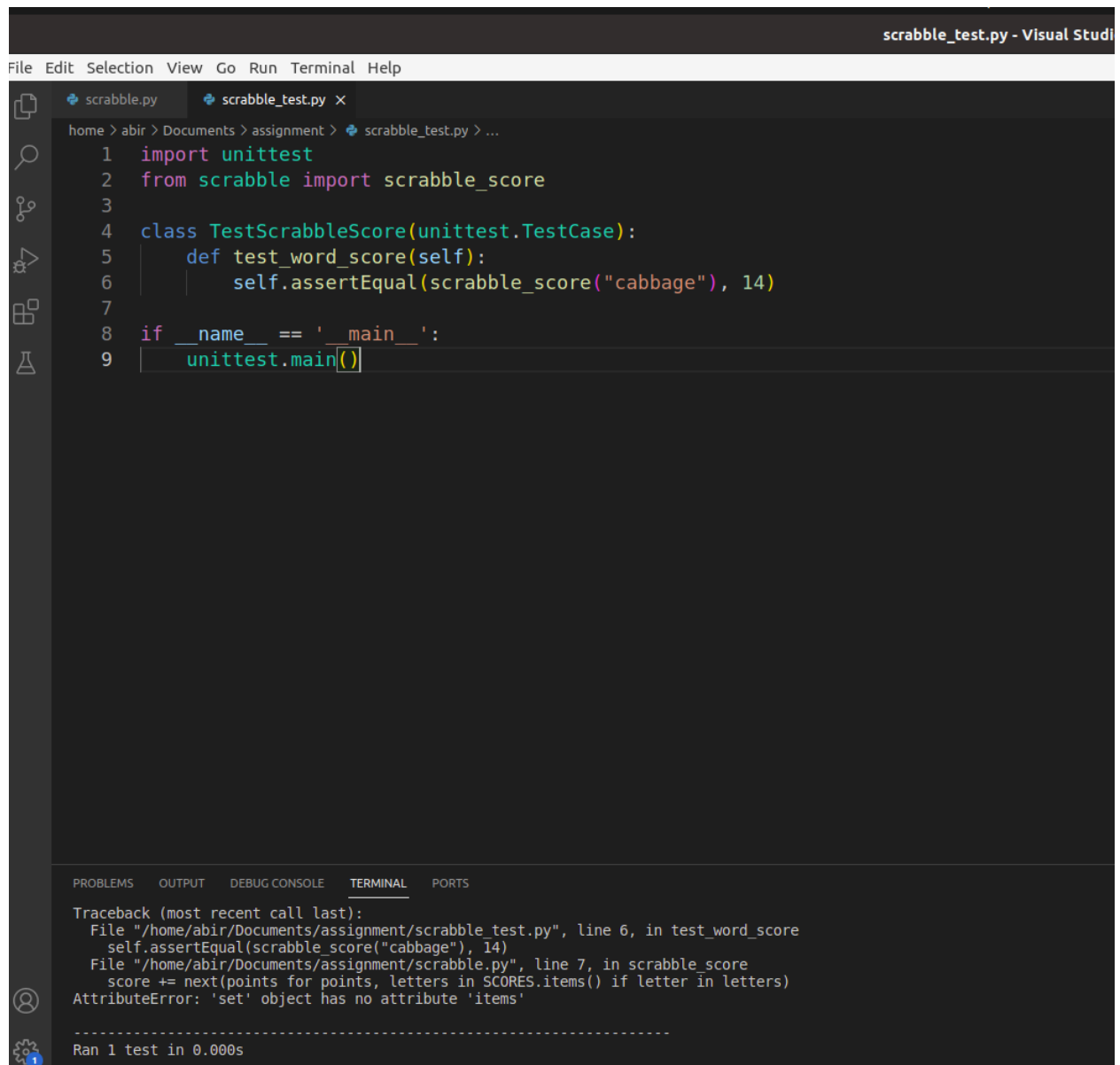
home > abir > Documents > assignment > scrabble_test.py > ...
1 import unittest
2 from scrabble import scrabble_score
3
4 class TestScrabbleScore(unittest.TestCase):
5     def test_word_score(self):
6         self.assertEqual(scrabble_score("cabbage"), 14)
7
8 if __name__ == '__main__':
9     unittest.main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

File "/home/abir/Documents/assignment/scrabble_test.py", line 6, in test_word_score
self.assertEqual(scrabble_score("cabbage"), 14)

Step 2: Running the Test (Fail)

Due to the lack of implementation of the `scrabble_score()` method, the unittest run of the test failed. In TDD, this failure is anticipated.



The screenshot shows the Visual Studio Code interface with a file named `scrabble_test.py` open. The code in the editor is as follows:

```
1 import unittest
2 from scrabble import scrabble_score
3
4 class TestScrabbleScore(unittest.TestCase):
5     def test_word_score(self):
6         self.assertEqual(scrabble_score("cabbage"), 14)
7
8 if __name__ == '__main__':
9     unittest.main()
```

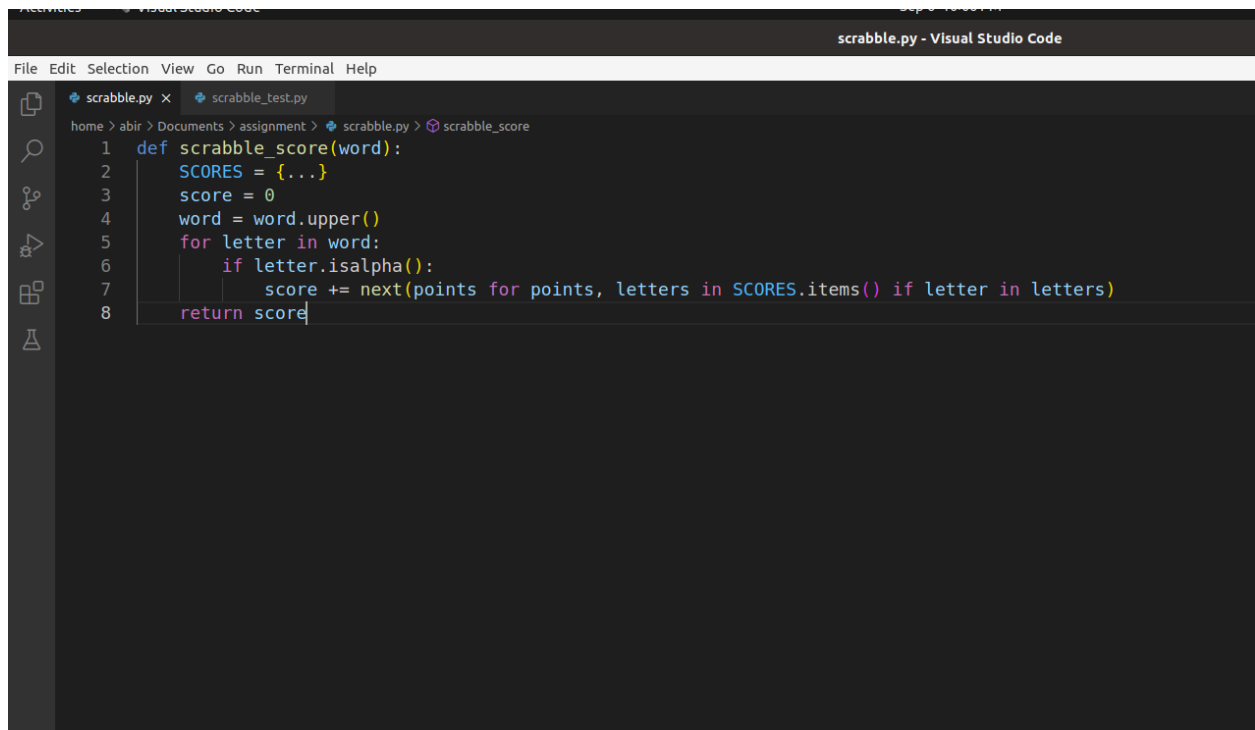
Below the code editor, the **TERMINAL** tab is active, displaying the following output:

```
Traceback (most recent call last):
  File "/home/abir/Documents/assignment/scrabble_test.py", line 6, in test_word_score
    self.assertEqual(scrabble_score("cabbage"), 14)
  File "/home/abir/Documents/assignment/scrabble.py", line 7, in scrabble_score
    score += next(points for points, letters in SCORES.items() if letter in letters)
AttributeError: 'set' object has no attribute 'items'
```

At the bottom of the terminal, it shows: `Ran 1 test in 0.000s`

Step 3: Implementing the Code

Next, in `scrabble.py`, we created the `scrabble_score()` method to calculate the score based on letter values:

A screenshot of the Visual Studio Code editor interface. The title bar at the top reads "scrabble.py - Visual Studio Code". The menu bar includes "File", "Edit", "Selection", "View", "Go", "Run", "Terminal", and "Help". The Explorer sidebar on the left shows two files: "scrabble.py" and "scrabble_test.py". The main editor window displays the code for "scrabble.py" with the following content:

```
1 def scrabble_score(word):
2     SCORES = {...}
3     score = 0
4     word = word.upper()
5     for letter in word:
6         if letter.isalpha():
7             score += next(points for points, letters in SCORES.items() if letter in letters)
8     return score
```

Step 4: Running the Test (Pass)

We reran the test after applying the code, and this time it passed, verifying that the score calculation functions as intended. For other features, this cycle of creating a test, watching it fail, putting the code into practice, and watching the test pass continued. The unittest framework that comes with Python was utilised to run and automate the tests.

Conclusion:

Using Test-Driven Development (TDD) to create a Scrabble scoring programme has been an insightful and useful project. The following are the main takeaways from the procedure. Here are the key lessons learned from the process:

Importance of TDD: TDD is important since it makes sure that we understand the needs of the code we are building. We can ensure that every component of the code functions as intended right from the beginning by writing tests before developing functionality. This method makes sure that the programme satisfies all requirements and aids in the early detection of faults.

Effective Use of Automated Testing: The unittest framework for Python turned out to be a very useful tool for testing process automation. It enhanced the speed and

dependability of code development by enabling us to promptly confirm that the code operates as intended following every modification.

Incremental Development: The incremental development method's significance is emphasised by the TDD approach. We can make sure every component functions properly before going on to the next by creating tests for discrete functionalities and putting them into practice one step at a time. This approach lowers the possibility of introducing problems and facilitates code debugging and maintenance.

Handling Edge Cases: Creating tests made it easier to find edge cases and make sure the software could adapt gracefully to a range of input conditions. This entails managing erroneous inputs and guaranteeing accurate output in various scenarios.

Code Quality Confidence: Testing the code on a regular basis gave assurance that it functions as intended. Testing new features frequently made sure that new code did not interfere with already-existing functionality.

All things considered, the project proved that TDD is a useful technique for creating reliable software. In addition to producing a Scrabble scoring programme that worked flawlessly, the methodical approach to generating and testing code offered insightful information about software development procedures.

[Github](#)