**Software Unit Testing Report**

**Introduction**

This project developed a Python-based Scabble Score Calculator to ascertain a player's word score. The application automates unit testing and game logic using Test-Driven Development (TDD). Participants in this program enter random-length words (generated each round), have them checked against a dictionary, and get Scrabble points. Participants on a 15-second timer must react fast and get extra points for speedier responses.

This project's development aims at various objectives:

1. Standard Scrabble letter values provide correct scores.
2. Treating lowercase and upper-case letters consistently.
3. Noting for users erroneous or non-alphabetic inputs.
4. consulting a dictionary phrase before scoring.
5. After ten rounds or until the player quits, shows the score.

Python was used for its simplicity and testing and validation easiness. Python's unit test framework was used to test our program as an automated bug and error detector. Automated testing is fast and reliable for code functionality verification without human tests.

**Process**

We used Test-Driven Development (TDD) in our project, which entails building test cases before writing code. Before going on to the next feature, we defined and tested each piece of functionality. Every stage of this approach provided us assurance that our code was working properly.

**1. Following the TDD Approach**

TDD was used all through the development process to guarantee the functionalities of the software operated as they should. We used TDD as follows:

**1. Writing the Tests First:**

Test cases were developed to specify function behavior prior to any code writing. We created a test case, for instance, to make sure the software would properly handle unusual characters like {\*} and calculate the Scrabble value for words like "cat" ,”xylophone” and "quiz."

Example Test Case:

    def test\_word\_scoring(self):

        """Test the score calculation for different words."""

        self.assertEqual(compute\_score('cat'), 5)

        self.assertEqual(compute\_score('quiz'), 22)

        self.assertEqual(compute\_score('xylophone'), 24)

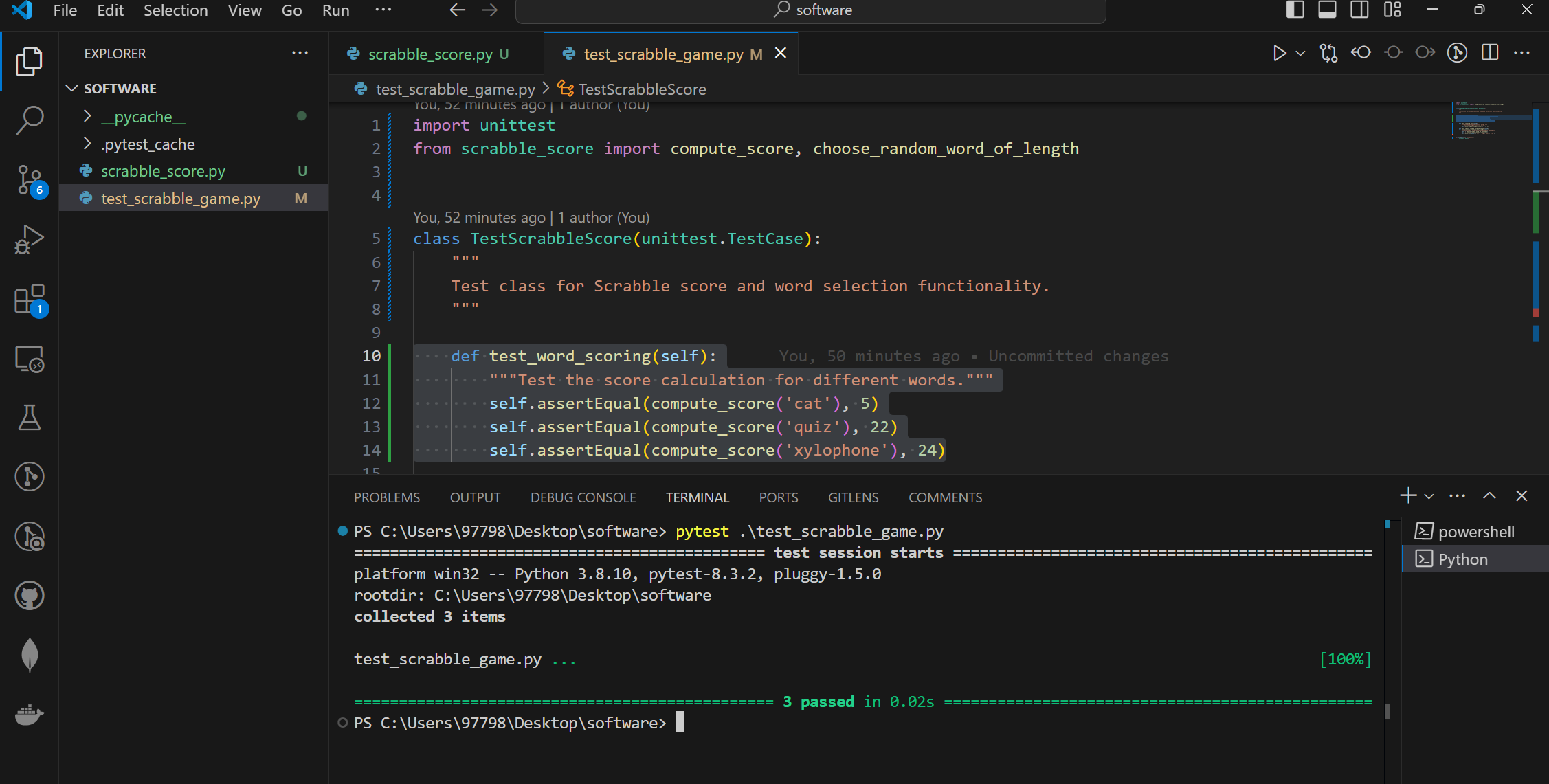
**2. Developing the Program:**

We developed the program once test cases were created. We developed `compute\_score()` method to apply pre-defined Scrabble letter values to word score computation. Depending on the given length, the function `choose\_random\_word\_of\_length()` chooses a random word from a dictionary.

**3. Running the Tests:**

We tested the functions to guarantee correct implementation once they were developed. Debugging and fixing of test failures in pertinent functions We made sure our software was totally operational using this iterative approach.

**Screenshot of successful test results:**



**2. Using the `unittest` Framework**

Python's "unittest" framework appealed to us because of its simplicity and automation. Using "unittest," word score and word choice calculations were rapidly verified. Before they became major problems, we tested to find and fix flaws.

**The key tests included:**

* Word Scoring: Making sure the computer uses the permitted letter values of Scrabble to precisely compute scores for different phrases.
* Handling invalid inputs: Making sure the program could manage non-alphabetic letters and erroneous words would help with edge situations.
* Selecting random words: choosing dictionaries terms with a certain length.

**Key Features Developed:**

**1. Random Word Choice:**

Each game begins with the computer choosing a word length (three to 10 letters) and asking the player to enter a word that fits. The new function "choose\_random\_word\_of\_length()" was built for this.

def choose\_random\_word\_of\_length(word\_length):

    """

    Select a random word of a given length from the dictionary.

    :param word\_length: int, the required length of the word.

    :return: str, the selected random word of the given length.

    """

    valid\_words = [word for word in dictionary if len(word) == word\_length]

    if valid\_words:

        return random.choice(valid\_words)

    return None

**2. 15-Second Timer**:

The game uses a 15-second timer to help one maintain time. It's an interesting challenge as fast word entering awards players extra points.

        # Check if the guess is correct and calculate the score

        score = compute\_score(guess)

        time\_bonus = max(0, 15 - int(time\_taken))

        # Time bonus: max of 15 points if entered quickly

        total\_score += score + time\_bonus

        print(f"Correct! Your score for this round: {score} + {time\_bonus} "

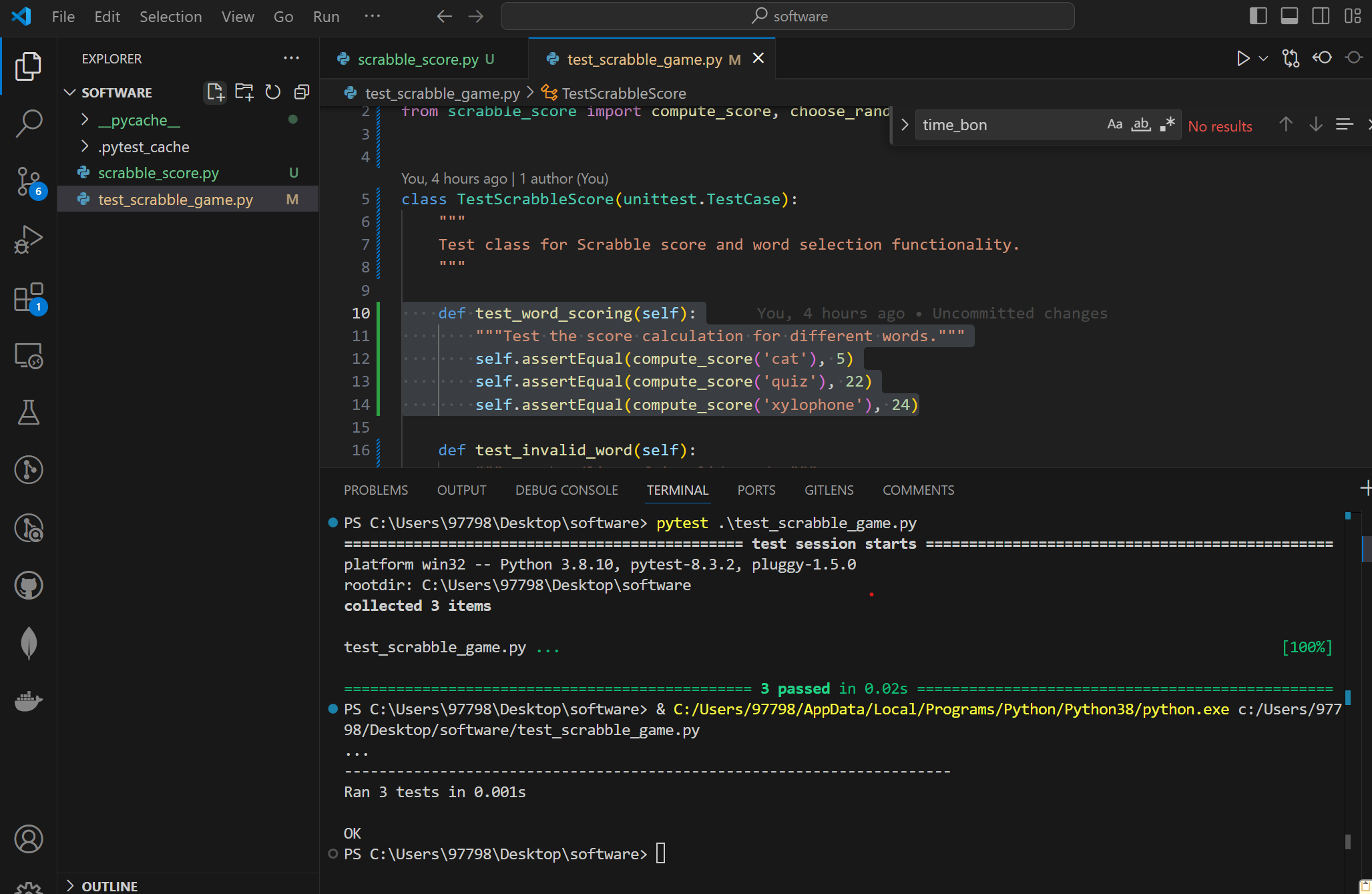
              f"(time bonus) = {score + time\_bonus}")

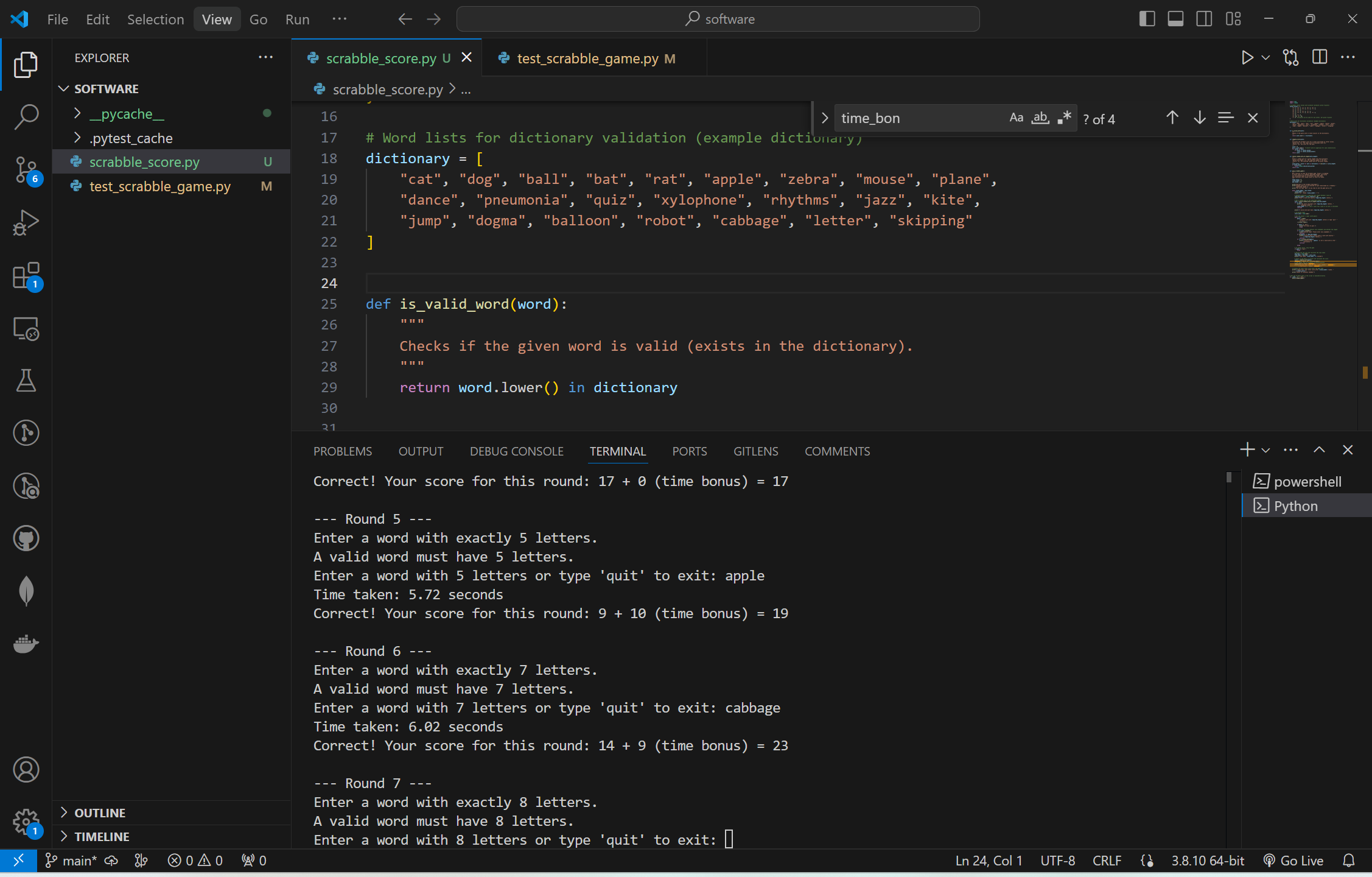
**3. Game Continuation:**

The player may play ten rounds before hitting "quit" to empty the board. Show the gamer their overall score after every game to introduce friendly rivalry into the app.

**Screenshots:**

**1. Test Case for Word Scoring:**





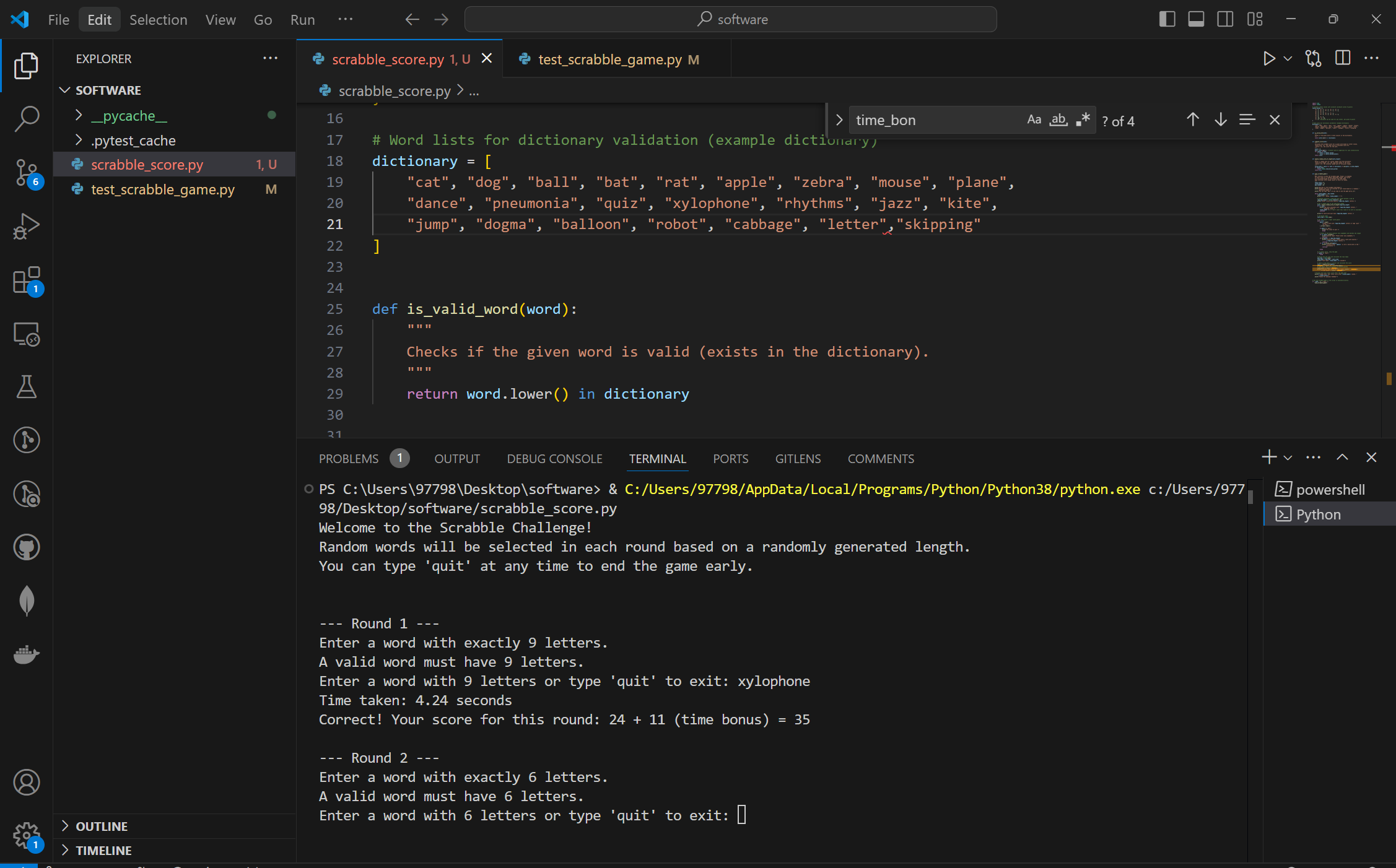
**2. Test Case for Word Length Selection:**

    def test\_choose\_random\_word\_of\_length(self):

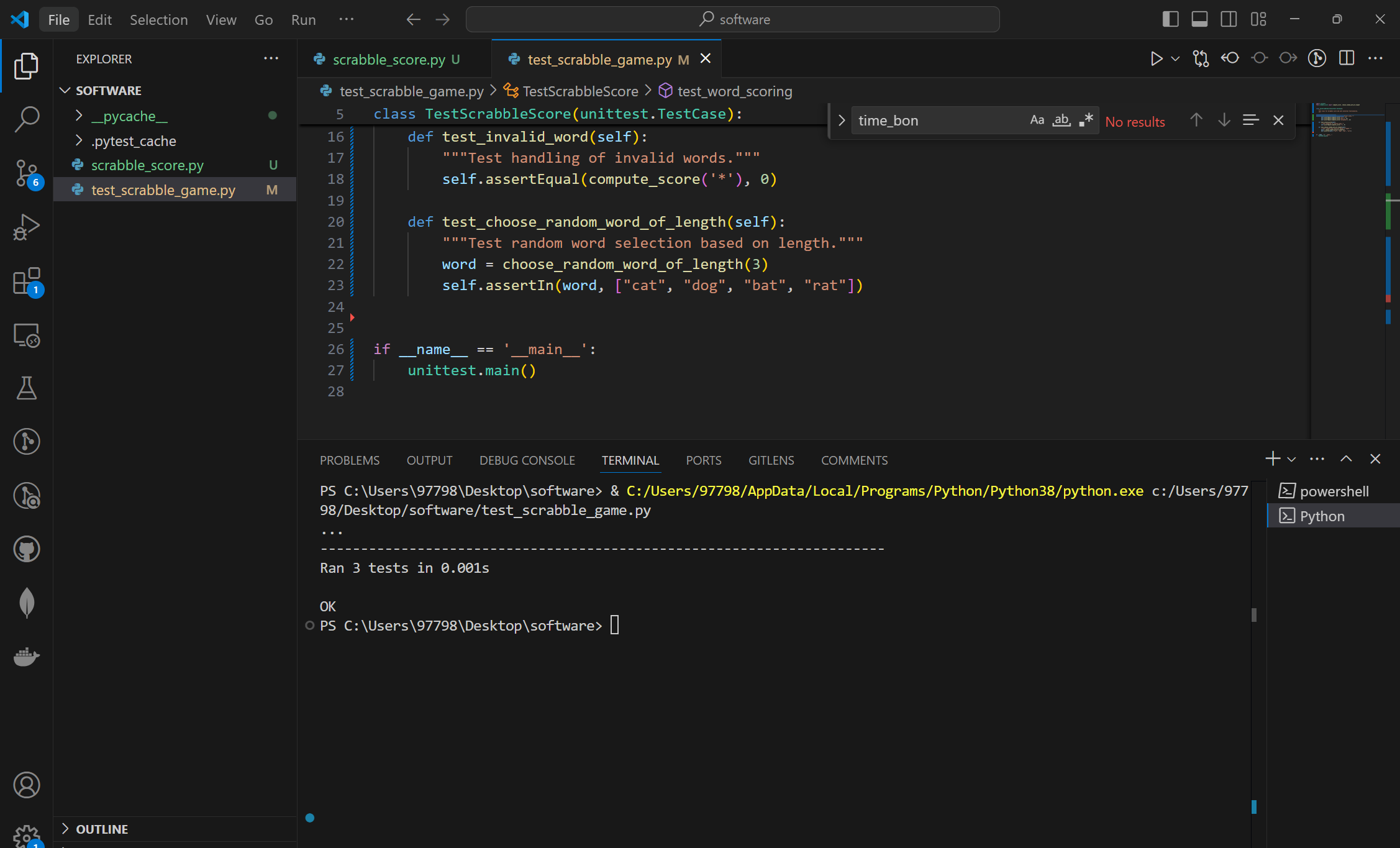
        """Test random word selection based on length."""

        word = choose\_random\_word\_of\_length(3)

        self.assertIn(word, ["cat", "dog", "bat", "rat"])



**3. Test Results:**



Conclusion

This project taught me how to write reliable, maintainable code using Test-Driven Development (TDD) and the 'unittest' framework. By initially writing tests, we could carefully describe each function's functionality and ensure the code met all requirements. This made the application more stable and reduced faults.

**Lessons Learned:**

**What went well:**

* Using TDD allowed us to ensure that each feature was fully developed before moving on to the next one. This not only made work simpler but also less likely to have problems.
* Automated testing utilising "unittest" validated consistently and quickly the functionality of our application, therefore ensuring that every operation performed as intended.

**What could be improved:**

* In order to make the game more realistic and difficult, we may add additional terms to the program's dictionary in the future.
* Making the game more feature-rich, such with a graphical user interface, might increase its appeal and accessibility.

Along with all of the code and test cases, the project is available entirely on GitHub. This repository contains the report, every test case, and the source files for the software.

**GitHub Link: https://github.com/anjanshrestha622/Scrabble\_Score\_Game.git**