PRT582

SOFTWARE ENGINEERING: PROCESS AND TOOLS

Assignment 02

Howard Mai S371832

2 September 2024

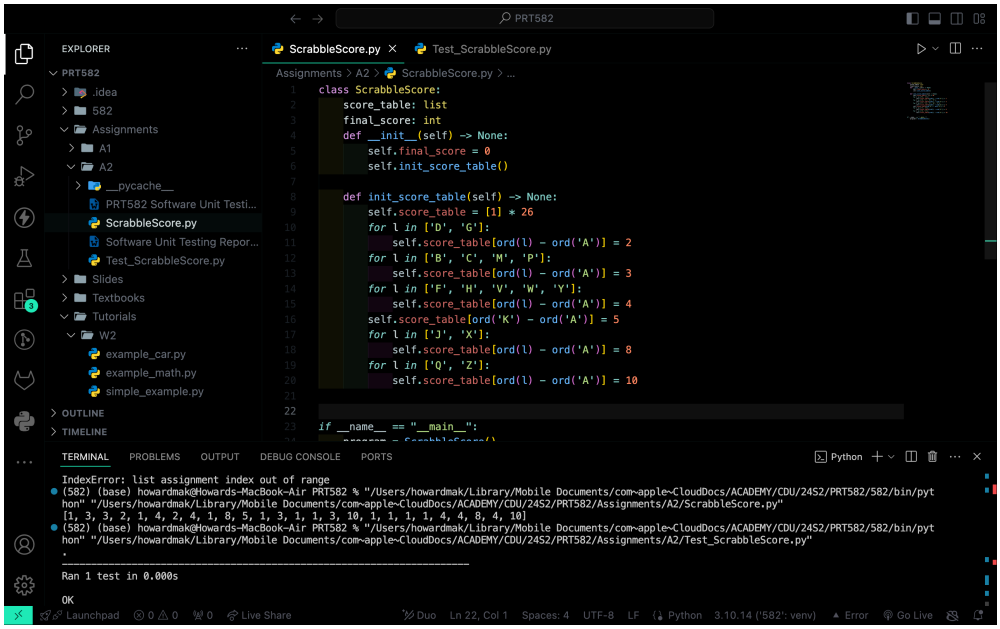
1. INTRODUCTION

To complete the given task and requirements, I use Python to design the game and the TDD method to write the program. Additionally, I also use `unittest` to test all the test cases. Finally, `Flake8` and `PyLint` are utilised to check whether the program is well organised.

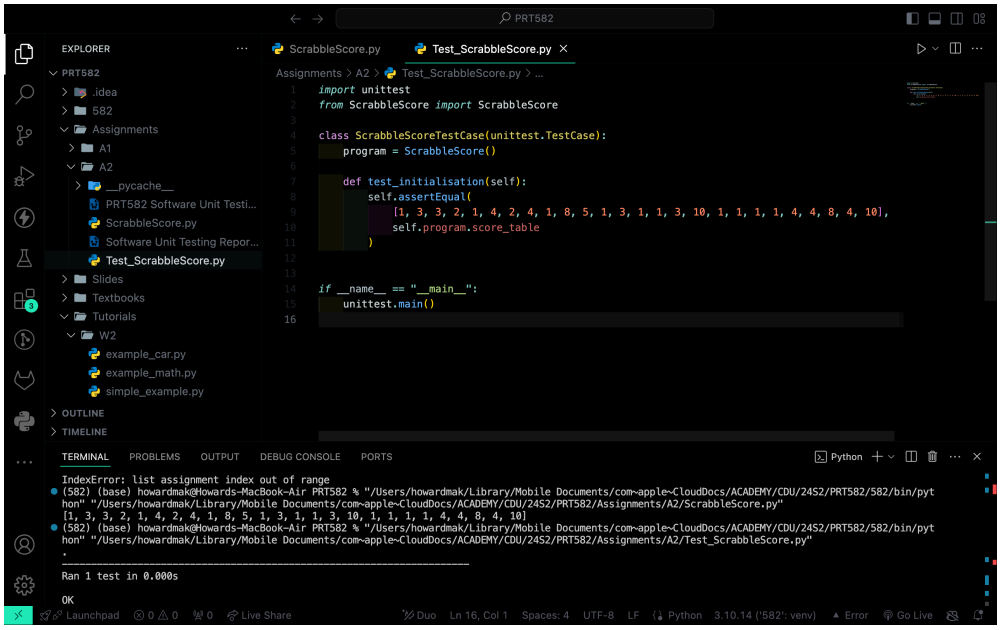
2. PROCESS

2.1. Initialisation

To make sure the numbers can be added up correctly, it is important to initialise a score table corresponding to the score of every letter.

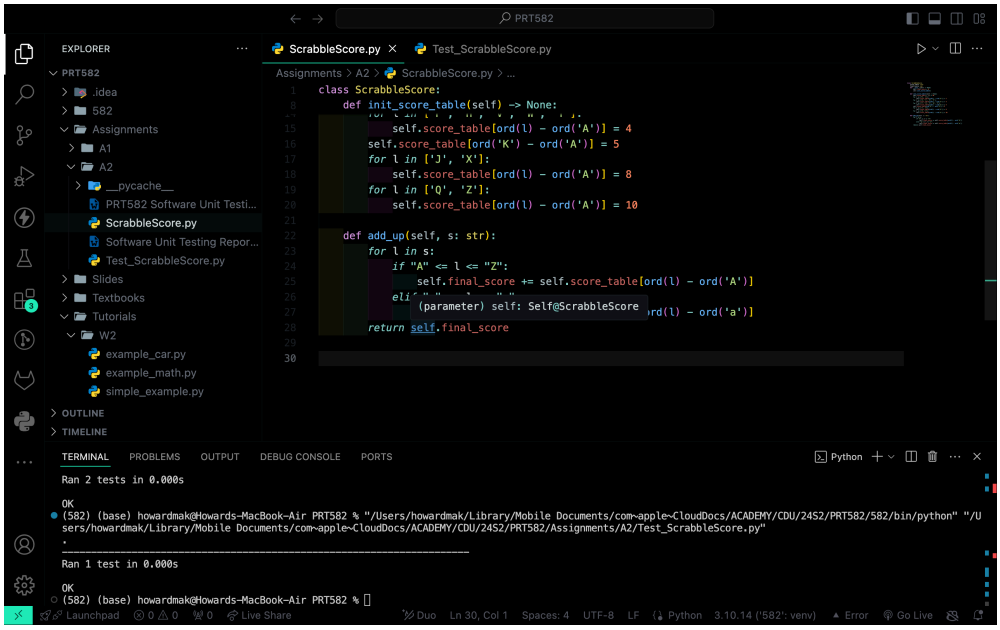


Running through the test case, it can bee seen that the program's score table matches the given condition:

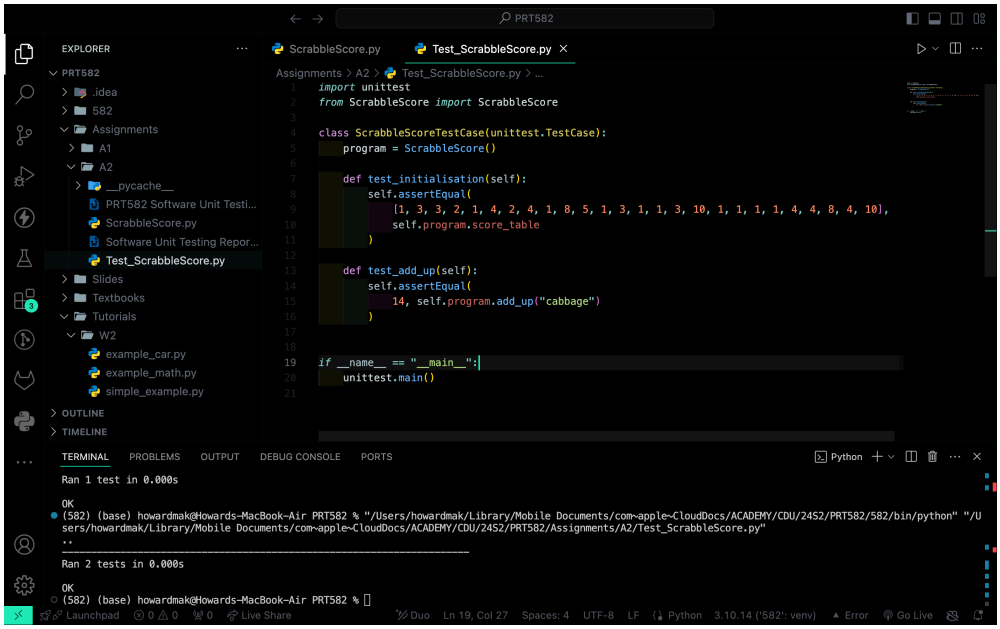


2.2. Adding Validation

In this step, I wrote a function called `add_up` to add up scores of every letter.

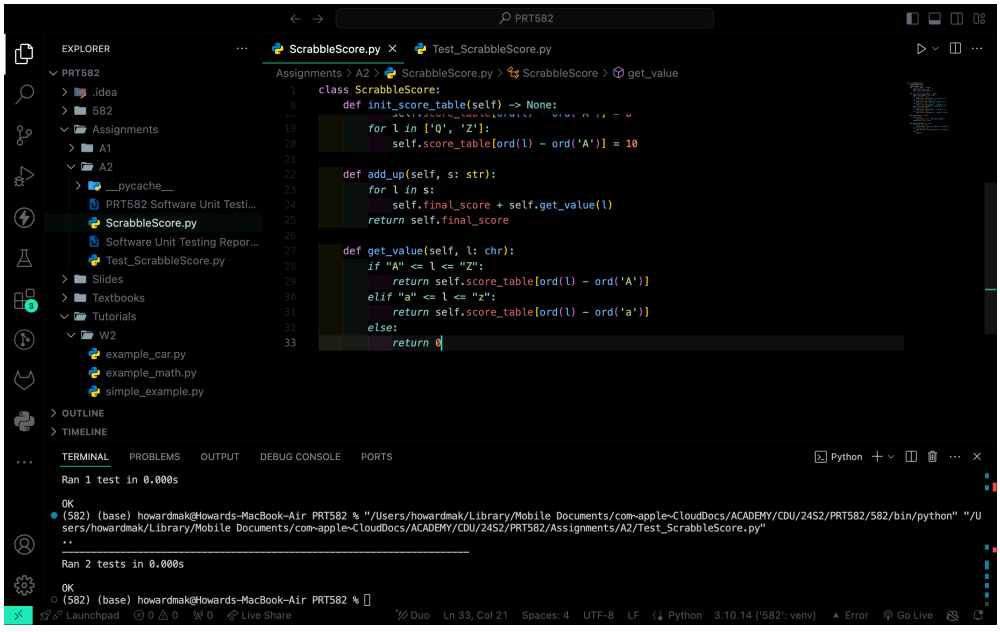


In this test case, the input variable is “cabbage”, and the returned score should be 14. It can be that the second test is passed:

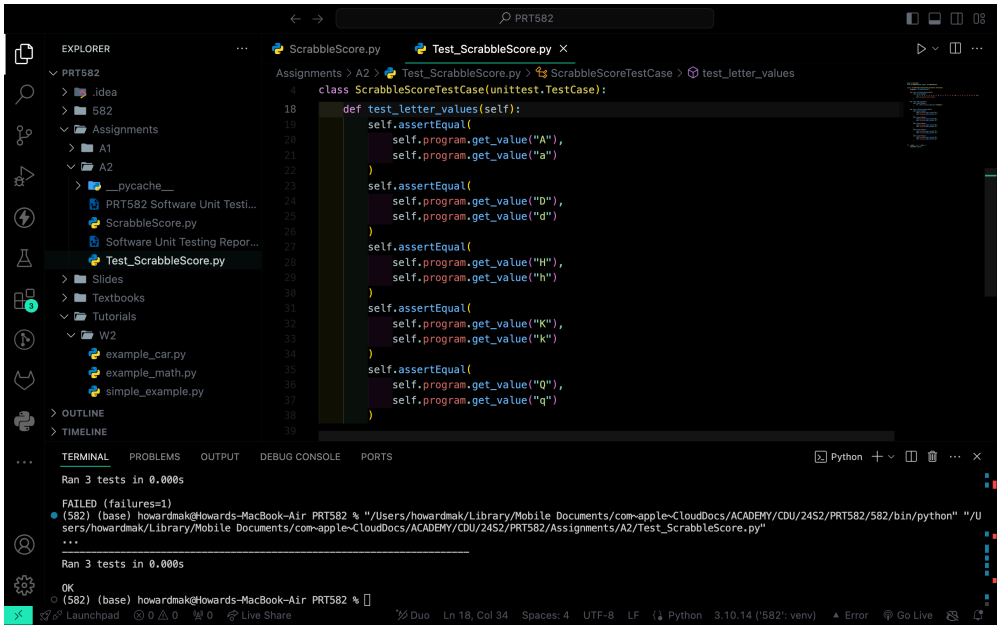


2.3. Letter Score Validation

Given the condition that upper- and lower-case letters should have the same value, I wrote another function called `get_value` to complete this and replace some codes in the function `add_up` with this function:

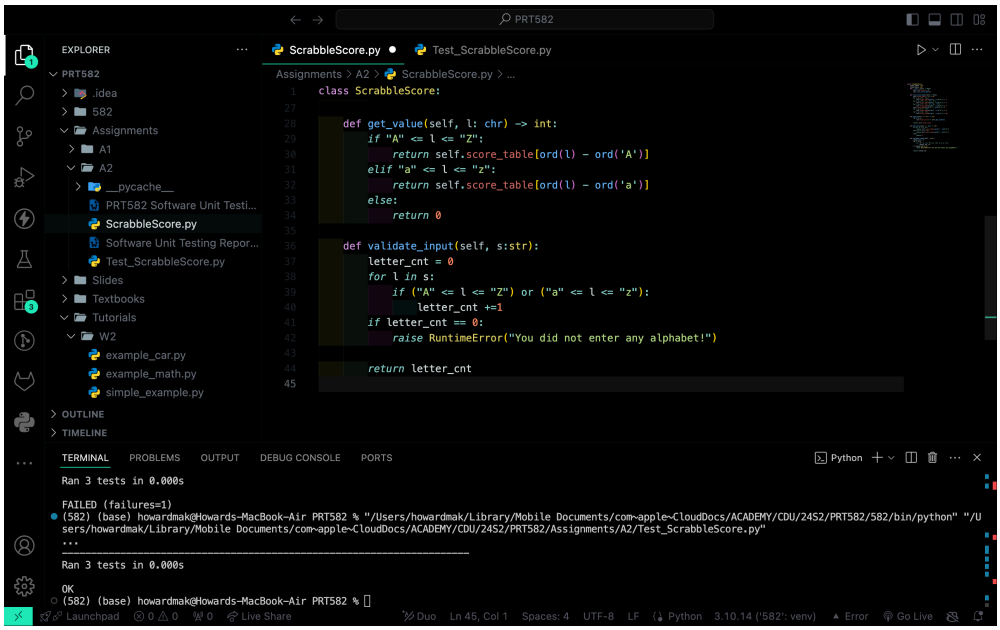


In this test case, the inputs are “A”, “D”, “H”, “K”, “Q”, and they should have the same value with their corresponding lower-case letters. It can be observed that the third test can be also passed:

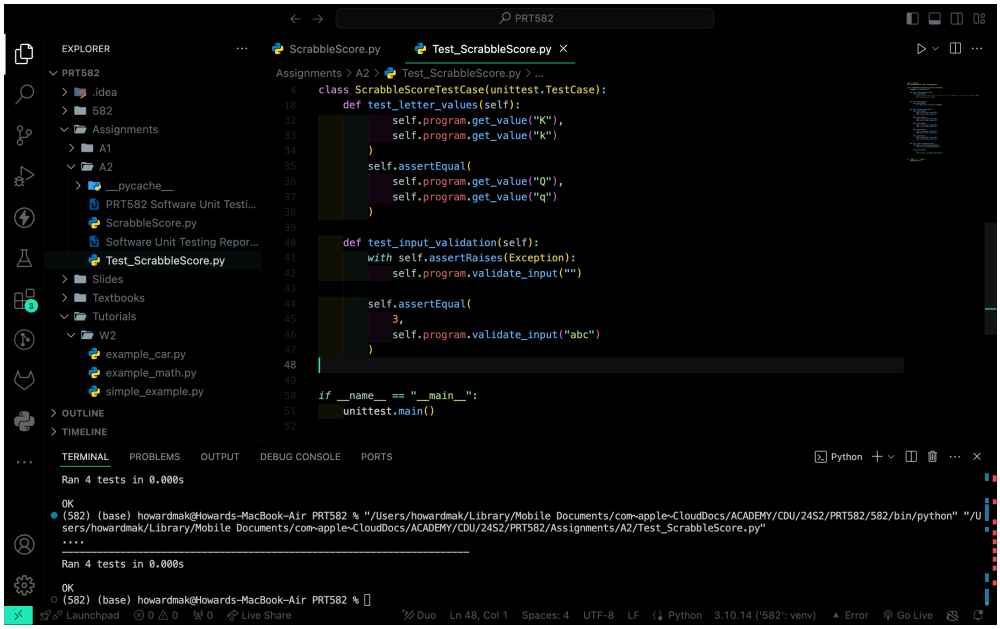


2.4. Input Validation

In this stage, a function called `validate_input` is created to check whether the user has input any alphabets, which will raise a `RuntimeError' when there are no alphabets or return the number of letters.

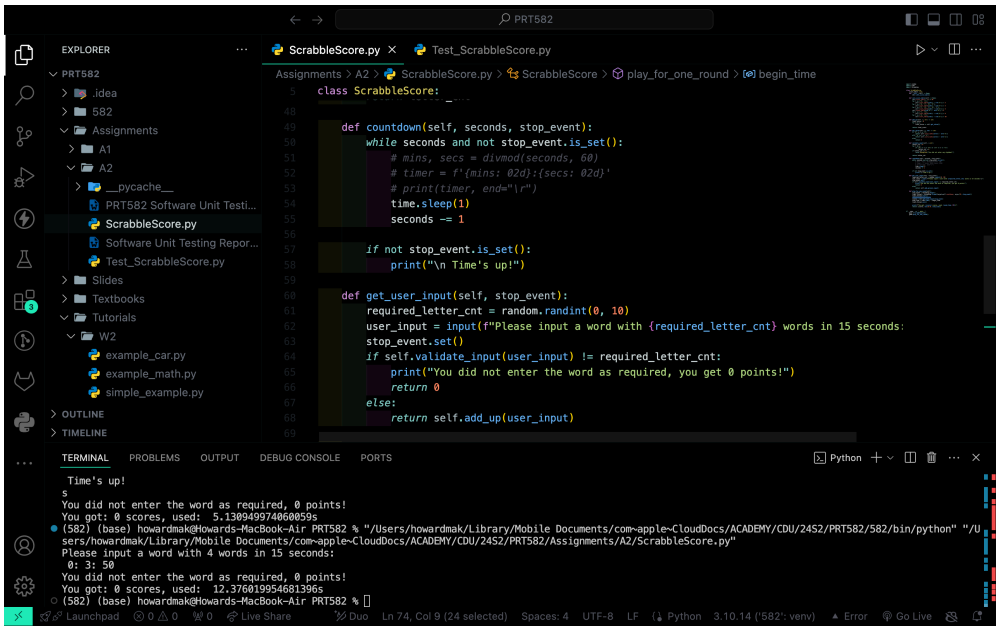


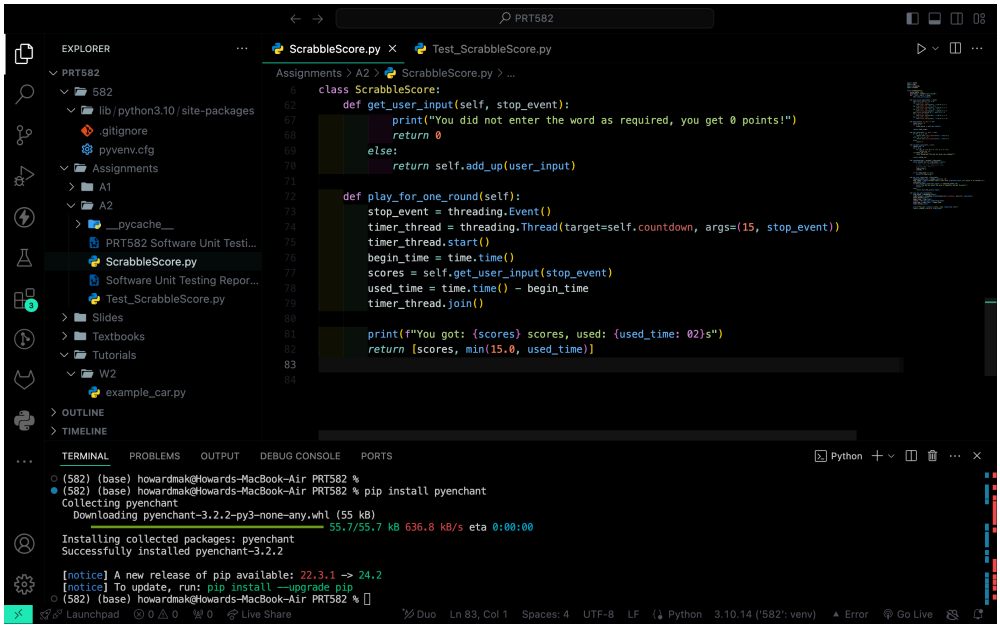
In this test case, there are two inputs, which are null string and “abc”. For the first input, the program should raise a `RuntimeError', while it should return 3 for the second input. It can been seen that this test can be passed:



2.5. Single Round

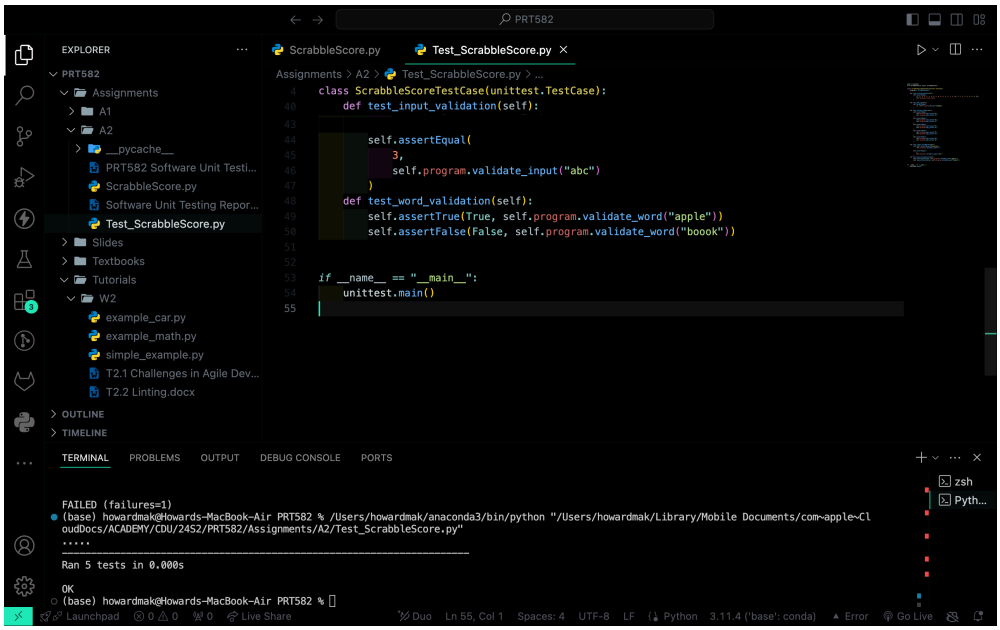
To implement the given condition, three functions are created, which are `countdown`, `get_user_input`, and `play_for_one_round`. Firstly, in the `countdown` function, `time` is imported to set the timer and count down. Secondly, in the `get_user_input`, users will be noticed to input a word, where the number of alphabets is randomly generated. If the word input by the user cannot be validated, it should return 0 scores. Lastly, in the `play_for_one_round`, threading is utilised to implement the timer and user input at the same time, and return the score the user get and the time used, making sure that the score will be higher if less time is used to enter the right length of word.





2.6. Word Validation

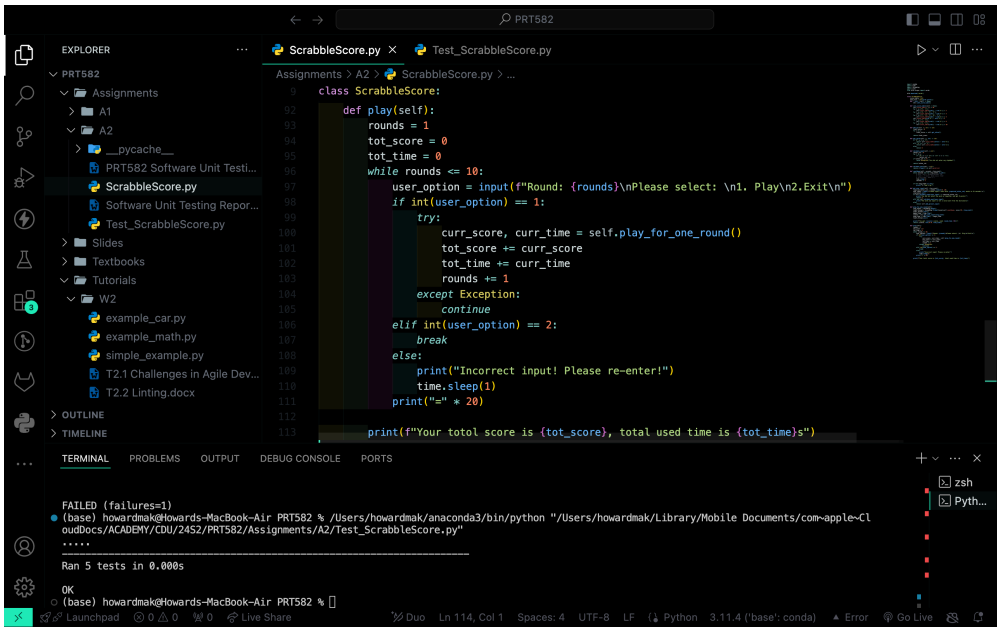
To implement the function that checks whether the word is from a dictionary, a dictionary is initialised when the program is initialised and a function called `validate_word`, which will return true when the word is in the dictionary. In this test case, two inputs, which are “apple” and “boook” are used to process the test. It can be seen that the test case is passed.



2.7. Keep the Game Going

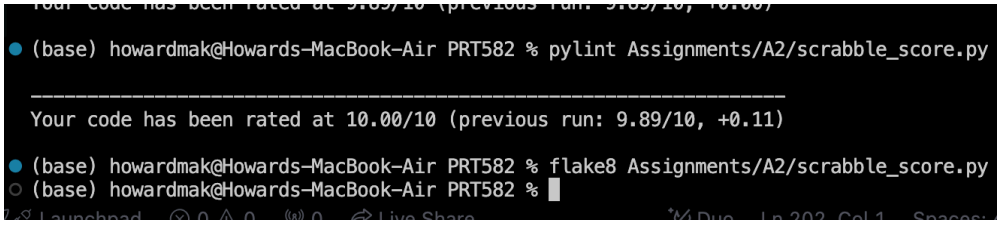
To meet the last requirement, a function `play` is created, where there will be 10 rounds. In each round, the user can choose to continue or exit. If any exception is raised,

the user should re-enter the option or re-enter the word. Finally, after all rounds, the game will count the final total score and used time.



2.8. Check the program with PyLint and Flake8

It can be observed that it passes the PyLint and Flake8 test.



3. CONCLUSION

Through the lesson, I learnt that though TDD development, a complex program can be built and tested step by step, which is very convenient to discover issues in every components.

Github link: https://github.com/HowardMak/PRT582.git, the program file and test cases are in the folder `A2_unit_test`.