# Software Unit Testing Report

## Introduction

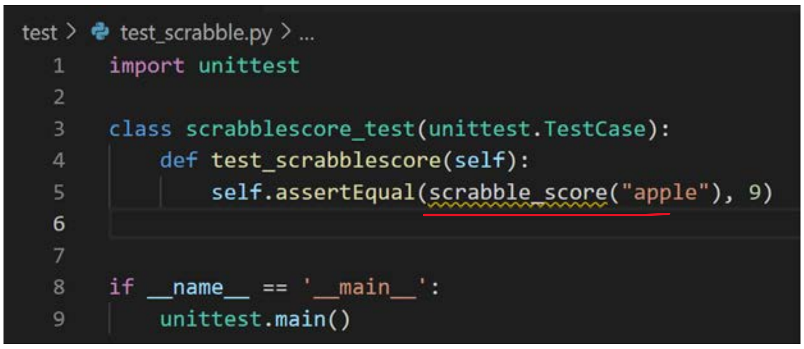
The task for this assignment was to write a program that can calculate the scrabble score of a given word. The main objective is to demonstrate Test Driven Development (TDD) process by developing test cases to state and validate the functionality of the code. For this assignment, TDD approach will be used to write failing tests before we write the main code. Test cases for each functional requirement will be created and tested first. As the test fails, new code will be added to pass the tests resulting in clean, clear, and simple code.

For this assignment, Python will be used as the programming language and PyUnit and PyTest will be used as the automated testing tools. Python is an interpreted high-level general-purpose programming language and is well known for software development and automation. Python has an object-oriented development approach which helps developers write clear and logical programs.

Unit testing is done early in the software development in order to identify bugs earlier and this results in clear code which less expensive to fix. PyUnit is a standard unit test framework for python which supports test cases for automated testing of the code. It helps us in detecting bugs sooner and helps in writing better programs.

## Process

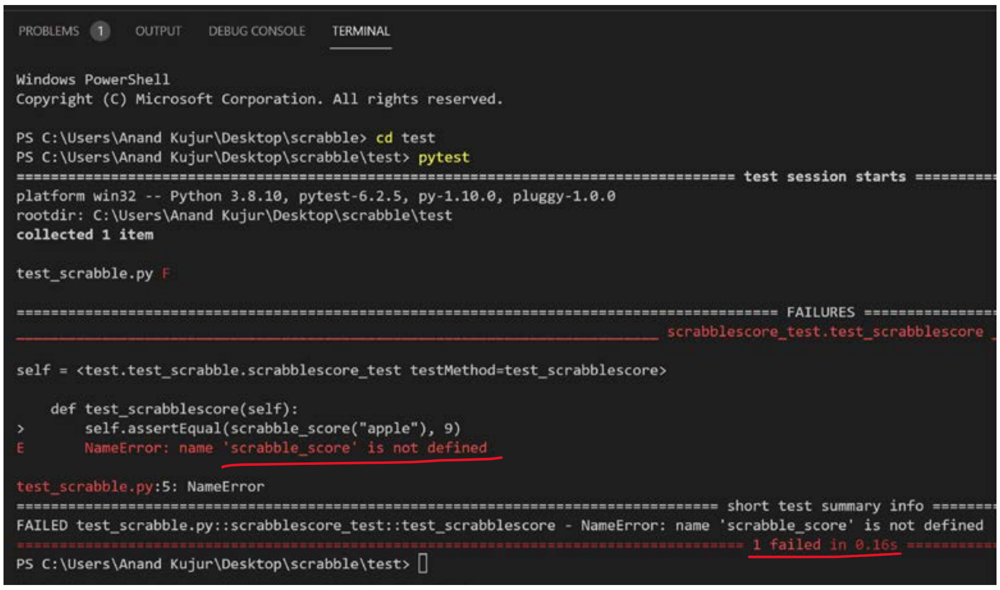
Since TDD approach will be used in this assignment, we need to first identify and understand each of the software requirements. The basic function of the program is to ask the user to input a word and will then calculate the scrabble score by adding the scores of each alphabet in the user input. In order to achieve this, a scrabble scoring function needs to be developed first, which will calculate the score of the user input by first accessing the alphabet scores stored in a variable and then adding the scores of the alphabets in the user input. The correct scrabble score should then be printed.


test > test_scrabble.py > ...

```python
import unittest

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self):
        self.assertEqual(scrabble_score("apple"), 9)


if __name__ == '__main__':
    unittest.main()
```

Figure 1 (writing the first test case)

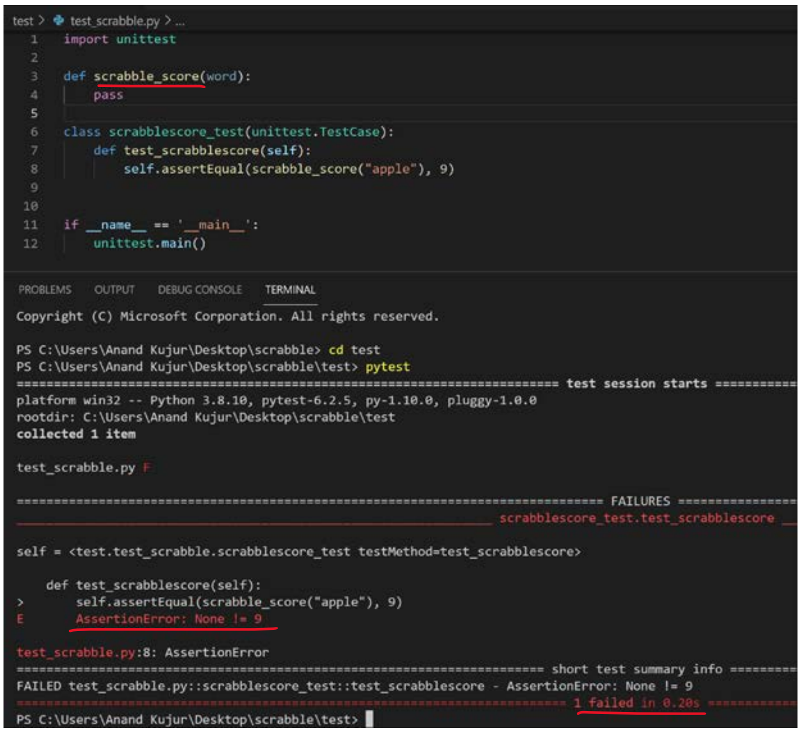
Figure 1 shows the first test case written assuming that there will be a function called “scrabble_score” which will give is the scrabble score of the user input. We have also assumed that the user input is the word “apple” which has a scrabble score 9. Since the test is based on assumptions, it will fail when it runs.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Anand Kujur\Desktop\scrabble> cd test
PS C:\Users\Anand Kujur\Desktop\scrabble\test> pytest
=================================================================================== test session starts =========
platform win32 -- Python 3.8.10, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\Users\Anand Kujur\Desktop\scrabble\test
collected 1 item

test_scrabble.py F

========================================================================================= FAILURES ================
___________________________________________________________________________ scrabblescore_test.test_scrabblescore _

self = <test.test_scrabble.scrabblescore_test testMethod=test_scrabblescore>

    def test_scrabblescore(self):
>       self.assertEqual(scrabble_score("apple"), 9)
E       NameError: name 'scrabble_score' is not defined

test_scrabble.py:5: NameError
================================================================================= short test summary info =======
FAILED test_scrabble.py::scrabblescore_test::test_scrabblescore - NameError: name 'scrabble_score' is not defined
==================================================================================== 1 failed in 0.16s ==========
PS C:\Users\Anand Kujur\Desktop\scrabble\test> 
```

Figure 2 (first test case failed)

PyTest was used for the first test and according to figure 2, the test failed because the name “scrabble_score” is not defined. This means we need to now create a function called “scrabble_score”.

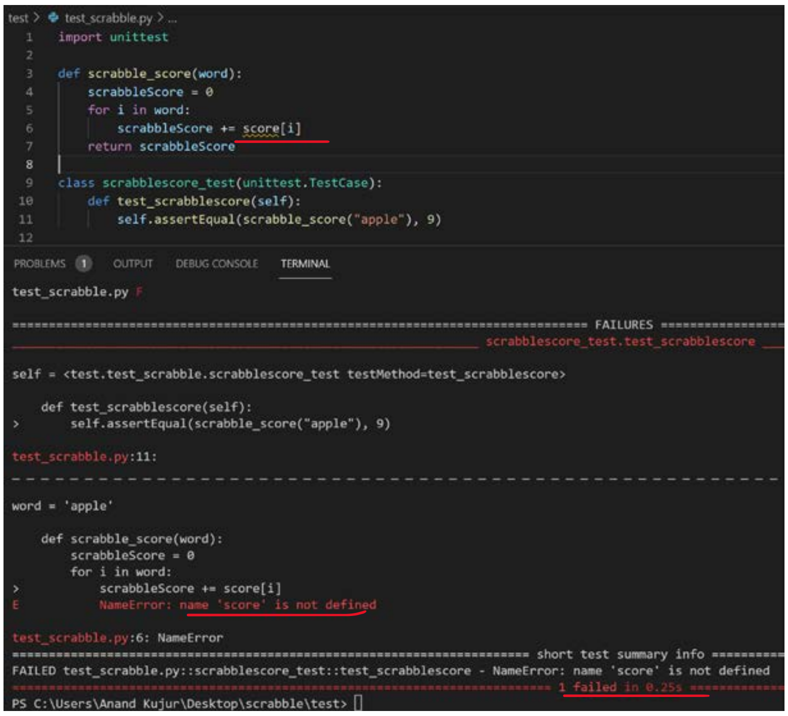


```python
import unittest

def scrabble_score(word):
    pass

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self):
        self.assertEqual(scrabble_score("apple"), 9)


if __name__ == '__main__':
    unittest.main()
```

```
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Anand Kujur\Desktop\scrabble> cd test
PS C:\Users\Anand Kujur\Desktop\scrabble\test> pytest
========================================================================= test session starts ===========
platform win32 -- Python 3.8.10, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\Users\Anand Kujur\Desktop\scrabble\test
collected 1 item

test_scrabble.py F

============================================================================== FAILURES ================
_________________________________________________________________ scrabblescore_test.test_scrabblescore __

self = <test.test_scrabble.scrabblescore_test testMethod=test_scrabblescore>

    def test_scrabblescore(self):
>       self.assertEqual(scrabble_score("apple"), 9)
E       AssertionError: None != 9

test_scrabble.py:8: AssertionError
======================================================================= short test summary info =========
FAILED test_scrabble.py::scrabblescore_test::test_scrabblescore - AssertionError: None != 9
========================================================================== 1 failed in 0.20s ===========
PS C:\Users\Anand Kujur\Desktop\scrabble\test> 
```

Figure 3 (define function “scrabble_score”)

As shown in figure 3, even when a new function called “scrabble_score” was defined, the test still failed because the function can not calculate the scrabble score yet. Here, an assertion error was displayed as the function “scrabble_score” cannot calculate the score of the word “apple” as 9. So, the next step would be to write code that will check each alphabet in the word “apple” and add the score of each alphabet to get the scrabble score.



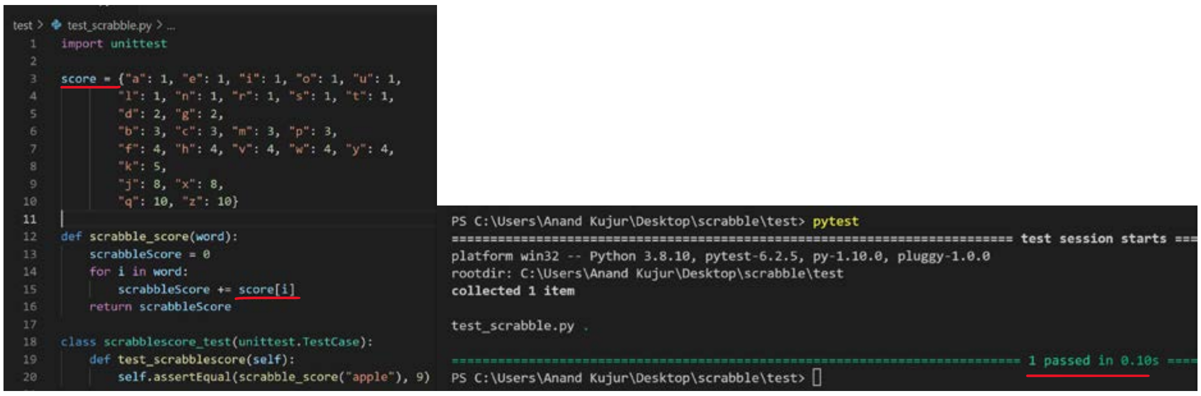
test > test_scrabble.py > ...

```
import unittest

def scrabble_score(word):
    scrabbleScore = 0
    for i in word:
        scrabbleScore += score[i]
    return scrabbleScore

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self):
        self.assertEqual(scrabble_score("apple"), 9)
```

```
test_scrabble.py F

============================================================================= FAILURES ================
_______________________________________________________________ scrabblescore_test.test_scrabblescore ___

self = <test.test_scrabble.scrabblescore_test testMethod=test_scrabblescore>

    def test_scrabblescore(self):
>       self.assertEqual(scrabble_score("apple"), 9)

test_scrabble.py:11:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

word = 'apple'

    def scrabble_score(word):
        scrabbleScore = 0
        for i in word:
>           scrabbleScore += score[i]
E           NameError: name 'score' is not defined

test_scrabble.py:6: NameError
======================================================================== short test summary info ==========
FAILED test_scrabble.py::scrabblescore_test::test_scrabblescore - NameError: name 'score' is not defined
========================================================================== 1 failed in 0.25s ============
PS C:\Users\Anand Kujur\Desktop\scrabble\test> 
```

Figure 4 (refactor)

In figure 4, code has been added to “scrabble_score” that will calculate the scrabble score for the word “apple”. But the test failed again as name “score” is not defined.



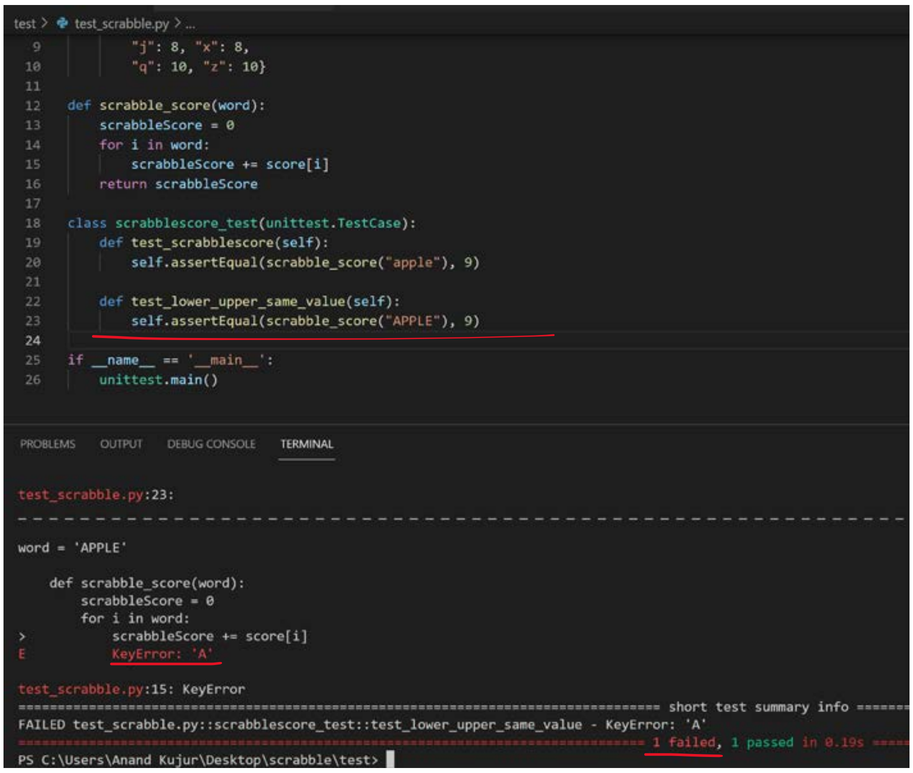
test > test_scrabble.py > ...

```
import unittest

score = {"a": 1, "e": 1, "i": 1, "o": 1, "u": 1,
        "l": 1, "n": 1, "r": 1, "s": 1, "t": 1,
        "d": 2, "g": 2,
        "b": 3, "c": 3, "m": 3, "p": 3,
        "f": 4, "h": 4, "v": 4, "w": 4, "y": 4,
        "k": 5,
        "j": 8, "x": 8,
        "q": 10, "z": 10}

def scrabble_score(word):
    scrabbleScore = 0
    for i in word:
        scrabbleScore += score[i]
    return scrabbleScore

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self):
        self.assertEqual(scrabble_score("apple"), 9)
```

```
PS C:\Users\Anand Kujur\Desktop\scrabble\test> pytest
=============================================================================== test session starts ===
platform win32 -- Python 3.8.10, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\Users\Anand Kujur\Desktop\scrabble\test
collected 1 item

test_scrabble.py .

================================================================================ 1 passed in 0.10s ===
PS C:\Users\Anand Kujur\Desktop\scrabble\test> 
```

Figure 5 (define variable “score”)

As seen in figure 5, a variable “score” is created which has the scores for all the alphabets. Now the test passed as the function “scrabble_score” was now able to check each alphabet in the word “apple” and add the scores [a(1) + p(3) + p(3) + l(1) + e(1) = apple(9)].

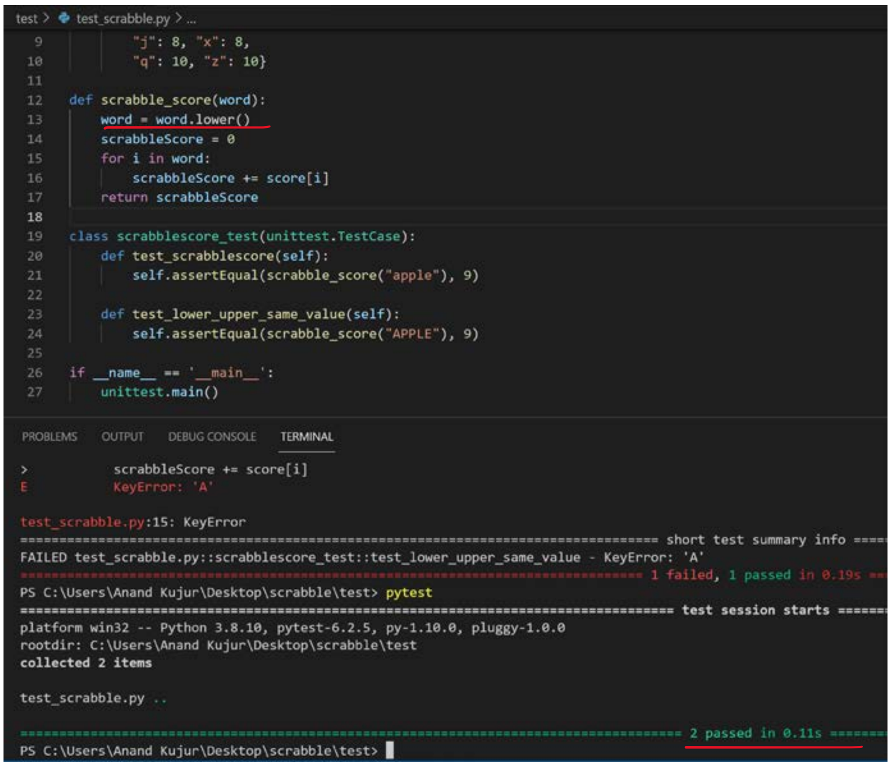


```
        "j": 8, "x": 8,
        "q": 10, "z": 10}

def scrabble_score(word):
    scrabbleScore = 0
    for i in word:
        scrabbleScore += score[i]
    return scrabbleScore

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self):
        self.assertEqual(scrabble_score("apple"), 9)

    def test_lower_upper_same_value(self):
        self.assertEqual(scrabble_score("APPLE"), 9)

if __name__ == '__main__':
    unittest.main()
```

```
test_scrabble.py:23:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

word = 'APPLE'

    def scrabble_score(word):
        scrabbleScore = 0
        for i in word:
>           scrabbleScore += score[i]
E           KeyError: 'A'

test_scrabble.py:15: KeyError
=================================================================================== short test summary info ======
FAILED test_scrabble.py::scrabblescore_test::test_lower_upper_same_value - KeyError: 'A'
================================================================================== 1 failed, 1 passed in 0.19s ====
PS C:\Users\Anand Kujur\Desktop\scrabble\test>
```

Figure 6 (second test case)

The next requirement was to have the same values for both upper- and lower-case alphabets. Figure 6 shows that the second test case has been added and the user input "APPLE" is in capital letters. We can also see that the second test failed because the variable "score" has scores for only small letters.

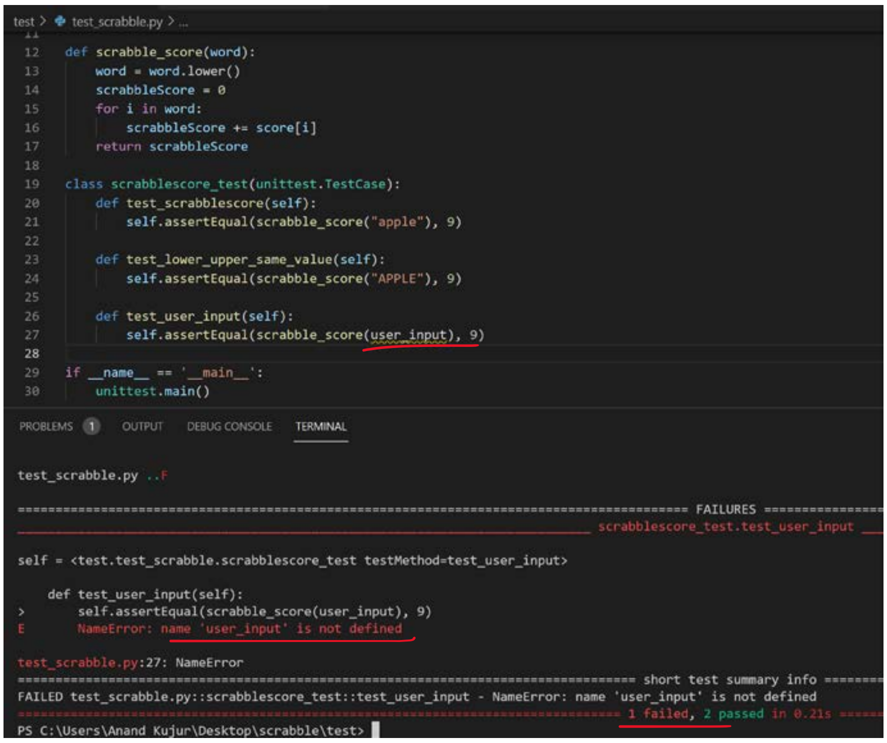


```
        "j": 8, "x": 8,
        "q": 10, "z": 10}

def scrabble_score(word):
    word = word.lower()
    scrabbleScore = 0
    for i in word:
        scrabbleScore += score[i]
    return scrabbleScore

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self):
        self.assertEqual(scrabble_score("apple"), 9)

    def test_lower_upper_same_value(self):
        self.assertEqual(scrabble_score("APPLE"), 9)

if __name__ == '__main__':
    unittest.main()
```

```
>           scrabbleScore += score[i]
E           KeyError: 'A'

test_scrabble.py:15: KeyError
=================================================================================== short test summary info ====
FAILED test_scrabble.py::scrabblescore_test::test_lower_upper_same_value - KeyError: 'A'
==================================================================================== 1 failed, 1 passed in 0.19s ==
PS C:\Users\Anand Kujur\Desktop\scrabble\test> pytest
===================================================================================== test session starts ======
platform win32 -- Python 3.8.10, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\Users\Anand Kujur\Desktop\scrabble\test
collected 2 items

test_scrabble.py ..

====================================================================================== 2 passed in 0.11s =======
PS C:\Users\Anand Kujur\Desktop\scrabble\test>
```

Figure 7 (refactor)

By adding a new line of code, as seen in figure 7, the user input will be seen as small letters when calculating the scrabble score. As a result, the second test was successful and now the upper- and lower-case alphabets will have the same value.

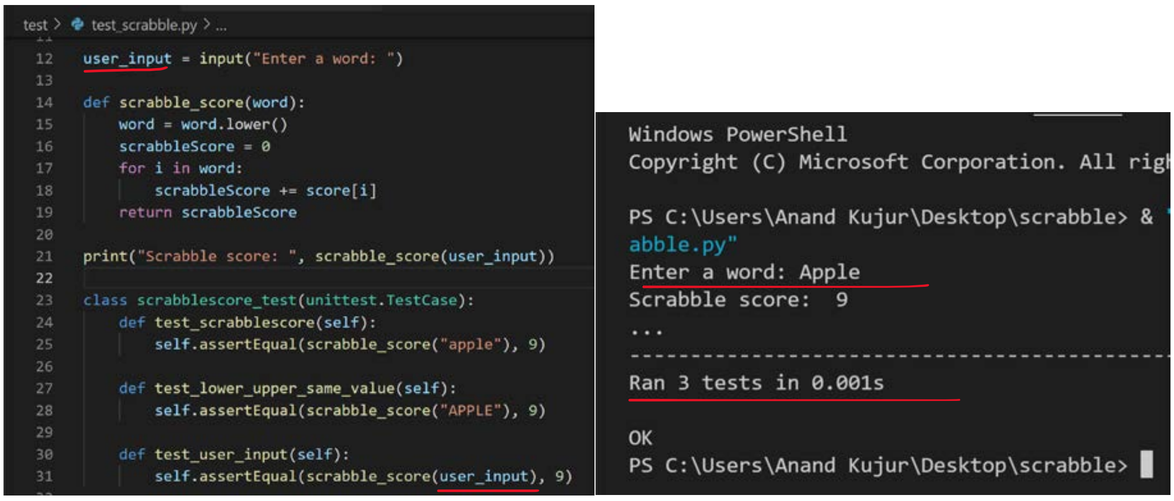


```
test > test_scrabble.py > ...
def scrabble_score(word):
    word = word.lower()
    scrabbleScore = 0
    for i in word:
        scrabbleScore += score[i]
    return scrabbleScore

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self):
        self.assertEqual(scrabble_score("apple"), 9)

    def test_lower_upper_same_value(self):
        self.assertEqual(scrabble_score("APPLE"), 9)

    def test_user_input(self):
        self.assertEqual(scrabble_score(user_input), 9)

if __name__ == '__main__':
    unittest.main()
```

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL

test_scrabble.py ..F

========================================================================================= FAILURES ===============
__________________________________________________________________________________ scrabblescore_test.test_user_input ___

self = <test.test_scrabble.scrabblescore_test testMethod=test_user_input>

    def test_user_input(self):
>       self.assertEqual(scrabble_score(user_input), 9)
E       NameError: name 'user_input' is not defined

test_scrabble.py:27: NameError
================================================================================= short test summary info =======
FAILED test_scrabble.py::scrabblescore_test::test_user_input - NameError: name 'user_input' is not defined
====================================================================================== 1 failed, 2 passed in 0.21s =====
PS C:\Users\Anand Kujur\Desktop\scrabble\test>
```

Figure 8 (third test case)

In figure 8, the third test case has been added to test if the user can enter a word and get the scrabble score. As we can see, the test failed as the name "user_input" is not defined.

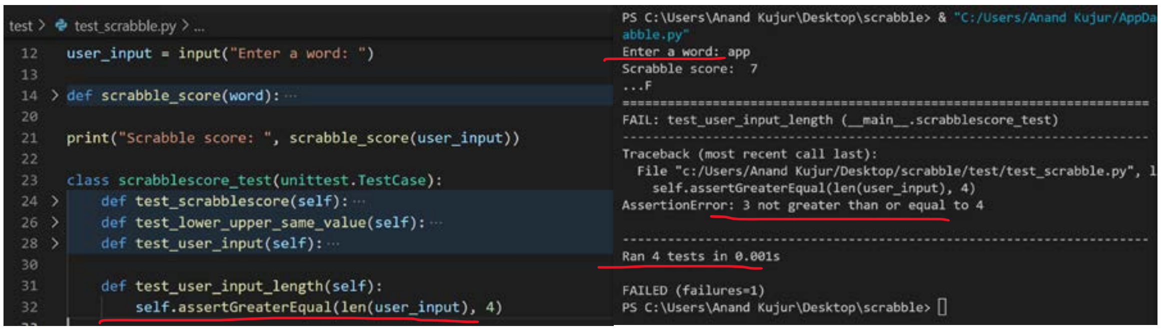


```
test > test_scrabble.py > ...
user_input = input("Enter a word: ")

def scrabble_score(word):
    word = word.lower()
    scrabbleScore = 0
    for i in word:
        scrabbleScore += score[i]
    return scrabbleScore

print("Scrabble score: ", scrabble_score(user_input))

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self):
        self.assertEqual(scrabble_score("apple"), 9)

    def test_lower_upper_same_value(self):
        self.assertEqual(scrabble_score("APPLE"), 9)

    def test_user_input(self):
        self.assertEqual(scrabble_score(user_input), 9)
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rig

PS C:\Users\Anand Kujur\Desktop\scrabble> &
abble.py"
Enter a word: Apple
Scrabble score:  9
...
--------------------------------------------
Ran 3 tests in 0.001s

OK
PS C:\Users\Anand Kujur\Desktop\scrabble>
```

Figure 9 (user input)

Figure 9 shows that the user input "user_input" has been defined. A print command has also been added to print the scrabble score for the user input. So, when the test was run, this time using PyUnit, the user was asked to enter a word. When the word "Apple" was entered, the scrabble score was printed, and the third test was completed.

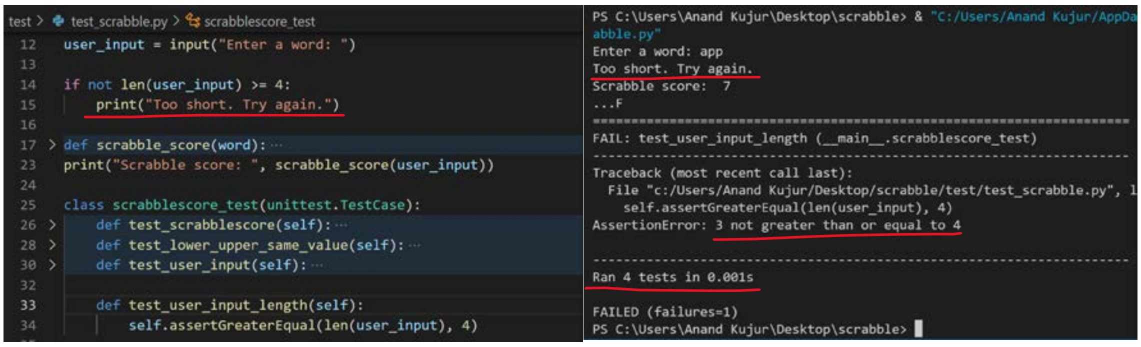


```
test > test_scrabble.py > ...
user_input = input("Enter a word: ")

def scrabble_score(word): ···

print("Scrabble score: ", scrabble_score(user_input))

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self): ···
    def test_lower_upper_same_value(self): ···
    def test_user_input(self): ···

    def test_user_input_length(self):
        self.assertGreaterEqual(len(user_input), 4)
```

```
PS C:\Users\Anand Kujur\Desktop\scrabble> & "C:/Users/Anand Kujur/AppDa
abble.py"
Enter a word: app
Scrabble score:  7
...F
======================================================================
FAIL: test_user_input_length (__main__.scrabblescore_test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:/Users/Anand Kujur/Desktop/scrabble/test/test_scrabble.py", l
    self.assertGreaterEqual(len(user_input), 4)
AssertionError: 3 not greater than or equal to 4

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (failures=1)
PS C:\Users\Anand Kujur\Desktop\scrabble>
```

Figure 10 (fourth test case)

In figure 10, the fourth test case has been added which will test if the user input is of certain length. For this assignment, the user input should be greater than equal to 4. As we can see, the test will fail when we enter a word “app” which is less than 4 characters long.

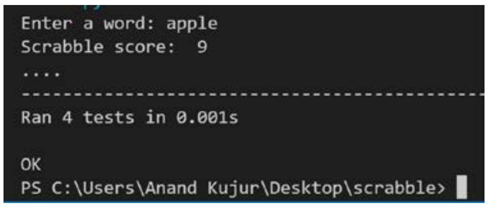


```
test > test_scrabble.py > scrabblescore_test
user_input = input("Enter a word: ")

if not len(user_input) >= 4:
    print("Too short. Try again.")

def scrabble_score(word): ···
print("Scrabble score: ", scrabble_score(user_input))

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self): ···
    def test_lower_upper_same_value(self): ···
    def test_user_input(self): ···

    def test_user_input_length(self):
        self.assertGreaterEqual(len(user_input), 4)
```

```
PS C:\Users\Anand Kujur\Desktop\scrabble> & "C:/Users/Anand Kujur/AppDa
abble.py"
Enter a word: app
Too short. Try again.
Scrabble score:  7
...F
======================================================================
FAIL: test_user_input_length (__main__.scrabblescore_test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:/Users/Anand Kujur/Desktop/scrabble/test/test_scrabble.py", l
    self.assertGreaterEqual(len(user_input), 4)
AssertionError: 3 not greater than or equal to 4

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (failures=1)
PS C:\Users\Anand Kujur\Desktop\scrabble>
```

Figure 11 (refactor)

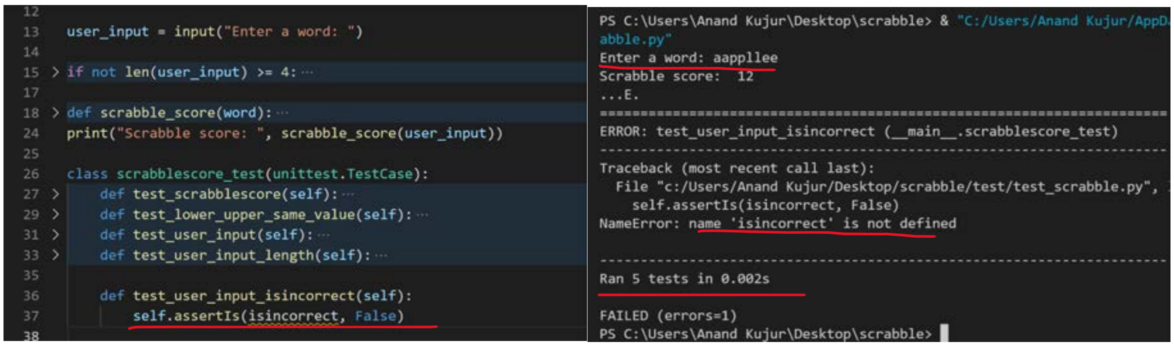
As seen in figure 11, new code has been added which will prompt user when the user input is less than 4 characters. So, when we entered a 3-letter word, the prompt was displayed that the word is too short, and we need to try again resulting in test failure.



```
Enter a word: apple
Scrabble score:  9
....
----------------------------------------------
Ran 4 tests in 0.001s

OK
PS C:\Users\Anand Kujur\Desktop\scrabble>
```

Figure 12

Figure 12 shows that when the test was run again and a 5-letter word “apple” was entered, the scrabble score was calculated, and the test passed successfully.

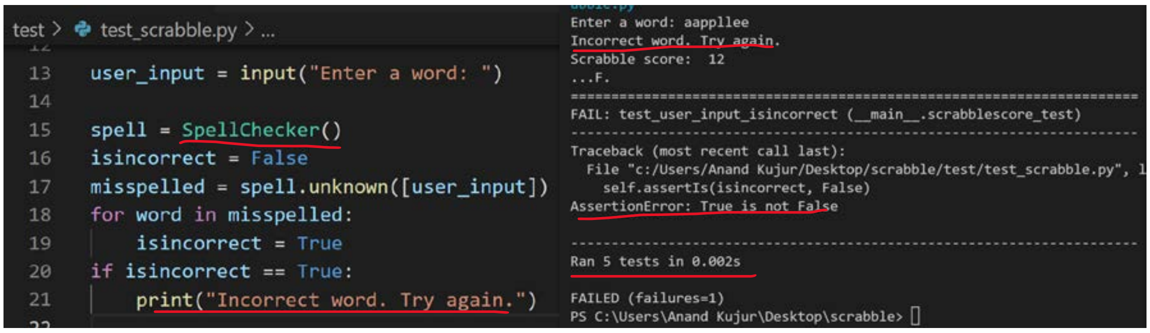


```python
user_input = input("Enter a word: ")

if not len(user_input) >= 4: ···

def scrabble_score(word): ···
print("Scrabble score: ", scrabble_score(user_input))

class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self): ···
    def test_lower_upper_same_value(self): ···
    def test_user_input(self): ···
    def test_user_input_length(self): ···

    def test_user_input_isincorrect(self):
        self.assertIs(isincorrect, False)
```

```
PS C:\Users\Anand Kujur\Desktop\scrabble> & "C:/Users/Anand Kujur/AppD
abble.py"
Enter a word: aappllee
Scrabble score:  12
...E.
======================================================================
ERROR: test_user_input_isincorrect (__main__.scrabblescore_test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:/Users/Anand Kujur/Desktop/scrabble/test/test_scrabble.py",
    self.assertIs(isincorrect, False)
NameError: name 'isincorrect' is not defined

----------------------------------------------------------------------
Ran 5 tests in 0.002s

FAILED (errors=1)
PS C:\Users\Anand Kujur\Desktop\scrabble>
```

Figure 13 (fifth test case)

As seen in figure 13, a new test case has been added that determines if the user input is an English dictionary word or not. As we can see, the test failed because of a name error.



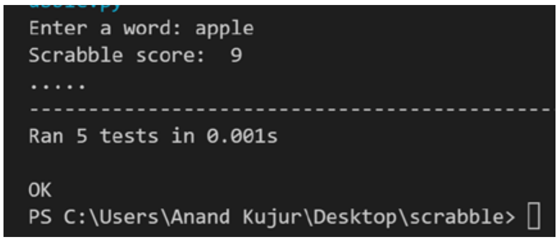
test > test_scrabble.py > ...

```python
user_input = input("Enter a word: ")

spell = SpellChecker()
isincorrect = False
misspelled = spell.unknown([user_input])
for word in misspelled:
    isincorrect = True
if isincorrect == True:
    print("Incorrect word. Try again.")
```

```
Enter a word: aappllee
Incorrect word. Try again.
Scrabble score:  12
...F.
======================================================================
FAIL: test_user_input_isincorrect (__main__.scrabblescore_test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:/Users/Anand Kujur/Desktop/scrabble/test/test_scrabble.py", l
    self.assertIs(isincorrect, False)
AssertionError: True is not False

----------------------------------------------------------------------
Ran 5 tests in 0.002s

FAILED (failures=1)
PS C:\Users\Anand Kujur\Desktop\scrabble>
```

Figure 14 (refactor)

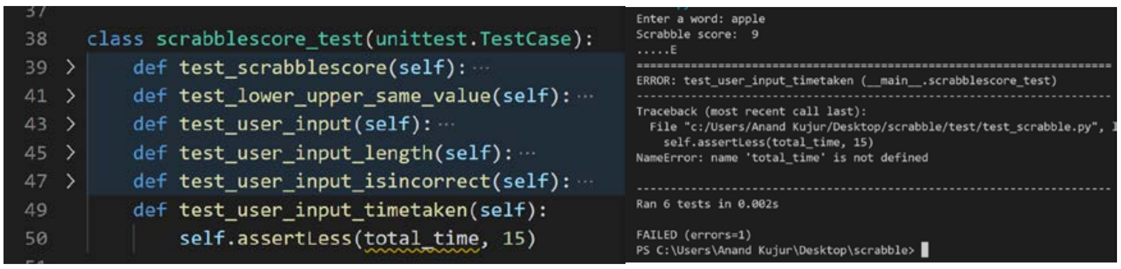
In figure 14, a new module "SpellChecker" has been used to check if the user input is an English dictionary word or not. This module uses an English dictionary by default and checks for spelling mistakes as well. So, when a misspelled word "aappllee" was entered, the test failed and the "Try again." Prompt was displayed.



```
Enter a word: apple
Scrabble score:  9
.....
----------------------------------------------
Ran 5 tests in 0.001s

OK
PS C:\Users\Anand Kujur\Desktop\scrabble>
```

Figure 15

As seen in figure 15, when a correct English word "apple" was entered, the scrabble score was calculated, and the test passed successfully.

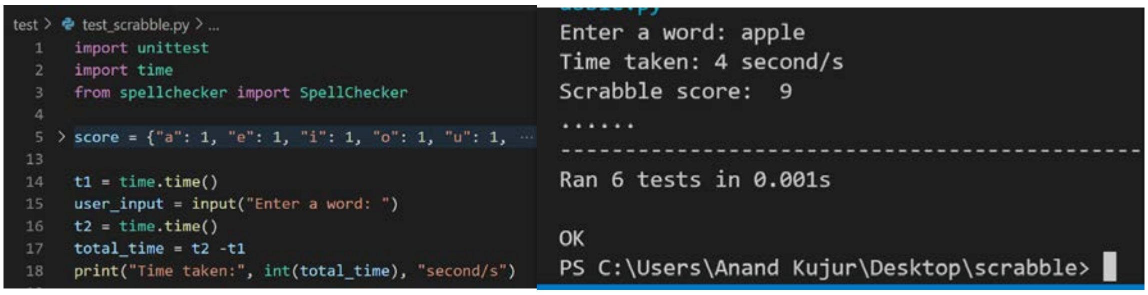


```python
class scrabblescore_test(unittest.TestCase):
    def test_scrabblescore(self): ...
    def test_lower_upper_same_value(self): ...
    def test_user_input(self): ...
    def test_user_input_length(self): ...
    def test_user_input_isincorrect(self): ...
    def test_user_input_timetaken(self):
        self.assertLess(total_time, 15)
```

```
Enter a word: apple
Scrabble score:  9
.....E
======================================================================
ERROR: test_user_input_timetaken (__main__.scrabblescore_test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:/Users/Anand Kujur/Desktop/scrabble/test/test_scrabble.py", 
    self.assertLess(total_time, 15)
NameError: name 'total_time' is not defined


----------------------------------------------------------------------
Ran 6 tests in 0.002s

FAILED (errors=1)
PS C:\Users\Anand Kujur\Desktop\scrabble> 
```

Figure 16 (calculating time taken)

To check the time taken to enter the user input, a new test case was added, as seen in figure 16. Just like previous tests the test failed because of a name error. In this case, the name “total_time” is not defined.



```python
test > test_scrabble.py > ...
import unittest
import time
from spellchecker import SpellChecker

score = {"a": 1, "e": 1, "i": 1, "o": 1, "u": 1, ...

t1 = time.time()
user_input = input("Enter a word: ")
t2 = time.time()
total_time = t2 -t1
print("Time taken:", int(total_time), "second/s")
```

```
Enter a word: apple
Time taken: 4 second/s
Scrabble score:  9
......
----------------------------------------------
Ran 6 tests in 0.001s

OK
PS C:\Users\Anand Kujur\Desktop\scrabble> 
```

Figure 17 (refactor)

In figure 17, “time” module has been used and new code has been added. This will record the time “t1” when the user is asked to enter a word and the time “t2” when the user is done. Then the time taken to enter user input which is “total_time” will be calculated by subtracting time “t1” from time “t2”. The time taken will then be printed along with the scrabble score. So, when the word “apple” is entered, the time taken, and the scrabble score is printed. With this a total of six tests have passed successfully.

In addition, more tests were performed while assessing other software requirements. Once all the remaining tests and the code was completed, the python script file (.py) was then converted into an executable file (.exe) using PyInstaller. This file can now be distributed and run-on other computers.

**Conclusion**

In this report, we have seen how TDD approach is so crucial for writing better codes and meeting all the software requirements. This process is as simple as writing failing test cases and then writing the required code to pass the test. This has helped in learning how code and test are iteratively built together one requirement at a time. This approach enables faster and better results as the code is robust. Using automated unit testing tools like PyUnit and PyTest speeds up the whole process and allows the developers write reliable and bug-free code.