**PRT 582 - SOFTWARE ENGINEERING PROCESS AND TOOLS**

**Software Unit Testing Report**

**Guess the Number game using Test Driven Development in Python**

**Report Created By:**

**Ravneet Singh  
(S370164)**

Table Of Contents

[**Introduction** 3](#_Toc143900306)

[Objectives and Requirements 3](#_Toc143900307)

[Automated Unit Testing Tool 3](#_Toc143900308)

[**Process** 4](#_Toc143900309)

[Test-Driven Development (TDD) 4](#_Toc143900310)

[Functionality 1 ( Generating Random Secret Number): 5](#_Toc143900311)

[Functionality 2 ( Test for all possible conditions): 6](#_Toc143900312)

[Automated Unit Testing: 7](#_Toc143900313)

[Pylint: 7](#_Toc143900314)

[Code Implementation: 7](#_Toc143900315)

[Conclusion 8](#_Toc143900316)

[Lessons Learned 8](#_Toc143900317)

# **Introduction**

This report outlines the objectives and requirements of the Number Guessing Game, along with the utilization of Test-Driven Development (TDD) and an automated unit testing tool in its development.

This report explains the software engineering process and tools utilized in the development of the Number Guessing Game. Further, it outlines the game's objectives and requirements, shedding light on the workings of Test-Driven Development (TDD) and the strategic deployment of an automated unit testing tool. We will also encapsulate the process with valid screenshots and draw conclusions while reflecting on valuable lessons learned.

## Objectives and Requirements

The main objective of the Number Guessing Game is to provide an engaging experience for players by challenging them to guess a secret 4-digit number with no repeating digits. The game must meet the following requirements:

1. **Random Number Generation**: The game should generate a random 4-digit secret number.
2. **User Interaction**: The game should accept user guesses and provide hints for each guess. Users can choose to quit the game at any time.
3. **Hint System**: The game should provide hints in the form of 'O' for a correct digit in the correct position, 'X' for a correct digit in the wrong position, and '$' for an incorrect digit.
4. **Multiple Rounds**: Players should have the option to play the game multiple times.

### Automated Unit Testing Tool

During the development of the Number Guessing Game, we have employed an automated unit testing tool to ensure the reliability and correctness of the code. Unit tests are written to verify the functionality of various game components, such as number generation and hint calculation. In practice, we have implemented automated unit tests using a framework such as 'unittest' to systematically validate the functionality of the 'NumGuessGame' class.

In place of manual testing, this automated testing method can be accomplished using predefined test cases to systematically validate functions. Testing is carried out to verify the accuracy of randomly generated numbers, the structure of clues, and input validation procedures. Each test case has predefined conditions that are anticipated to be met or violated, and the results show whether the game's features are reliable.

The code file provided guess\_game.py contains the implementation of the game and test\_guess\_game.py contains automated testing code.

# **Process**

In this section, we will detail how Test-Driven Development (TDD) and the automated unit testing tool have been utilized to create the Number Guessing Game. We will also provide relevant screenshots to illustrate the process.

# Test-Driven Development (TDD)

Test-Driven Development (TDD) is a crucial methodology in software development, which involves the following steps:

1. **Write Tests**: Initially, tests are written to define the expected behaviour of the code.
2. **Write Code**: The code is then written to fulfill the requirements set by the tests.
3. **Run Tests**: The tests are executed to check if the code passes or fails based on the expected behavior.

For the Number Guessing Game, the requirements of the game to be followed are listed as follows:

1. The game must generate a random four-digit number and save it as a secret number.

2. The game should continuously prompt the user to guess the number until they guess it correctly or choose to quit.

3. At each guess, the game provides clues to indicate the accuracy of the guess

1. 'O' – denotes that the digit is correct and in the correct position.
2. 'X' – denotes that the digit is correct but in incorrect position.
3. '$' – denotes that the digit is incorrect.
4. After completion of game:
5. Total attempts made are displayed.
6. The user can choose the option to quit the game or play again.

The requirements are divided into functionalities, each of which performs a specific function, as part of the development process. The ‘guess\_game.py' file implements these functions. A collection of unit tests is included in the 'test\_guess\_game.py', file which is automated.

## Functionality 1 ( Generating Random Secret Number):

It checks several aspects of the generated secret number, including its range, type, length, uniqueness of digits, and whether all characters are digits.

This test case is for a class NumGuessGame and tests the **generate\_secret\_number** method. A breakdown of what each part of the test case does is as follows:

1. **Valid Range Check**: **self.assertTrue(1000 <= int(secret\_number) <= 9999):**

It ensures that the generated number is within the range of 1000 to 9999, inclusive.

1. **Type Check**: **self.assertTrue(isinstance(secret\_number, str)):**

It checks if the generated number is a string.

1. **Length Check**: **self.assertEqual(len(secret\_number), 4)**:

It verifies that the length of the generated number is exactly 4 characters.

1. **Uniqueness Check**: **self.assertEqual(len(set(secret\_number)), 4)**:

It checks if all digits in the generated number are unique.

1. **Digits Check**: **self.assertTrue(secret\_number.isdigit()):**

It checks if all characters in the generated string are digits (0-9).

Additionally, each of these checks prints a message indicating whether the test was completed successfully or not.

A screen shot of a computer

Description automatically generated

Figure 1: Code\_Implementation

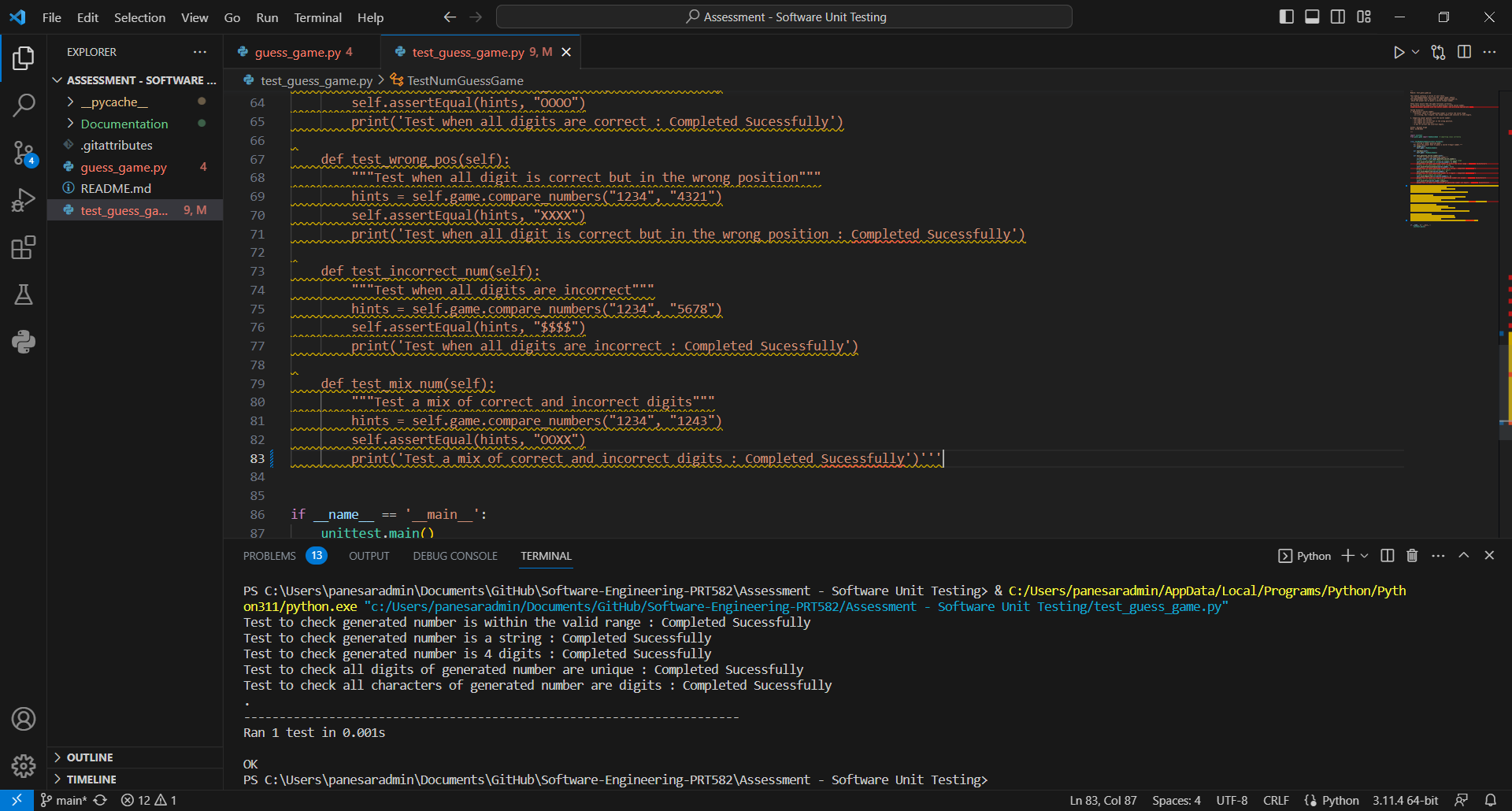


Figure 2: Output\_Functionality\_1

## Functionality 2 ( Test for all possible conditions):

The code you've provided includes test cases for a function named **compare\_numbers**. These test cases seem to be testing the functionality of this function, Here's an explanation of each test case and a suggestion for a test case:

1. **Test when all digits are correct** (**test\_all\_correct**):

This test case checks if the function correctly returns "OOOO" when all digits are in the correct positions. It uses "1234" as the secret number and "1234" as the guessed number.

1. **Test when all digits are correct but in the wrong position** (**test\_wrong\_pos**):

This test case checks if the function correctly returns "XXXX" when all digits are correct but not in the correct positions. It uses "1234" as the secret number and "4321" as the guessed number.

1. **Test when all digits are incorrect** (**test\_incorrect\_num**):

This test case checks if the function correctly returns "$$$$" when all digits are incorrect. It uses "1234" as the secret number and "5678" as the guessed number.

1. **Test a mix of correct and incorrect digits** (**test\_mix\_num**):

This test case checks if the function correctly returns "OOXX" when there's a mix of correct and incorrect digits in the guessed number. It uses "1234" as the secret number and "1243" as the guessed number.

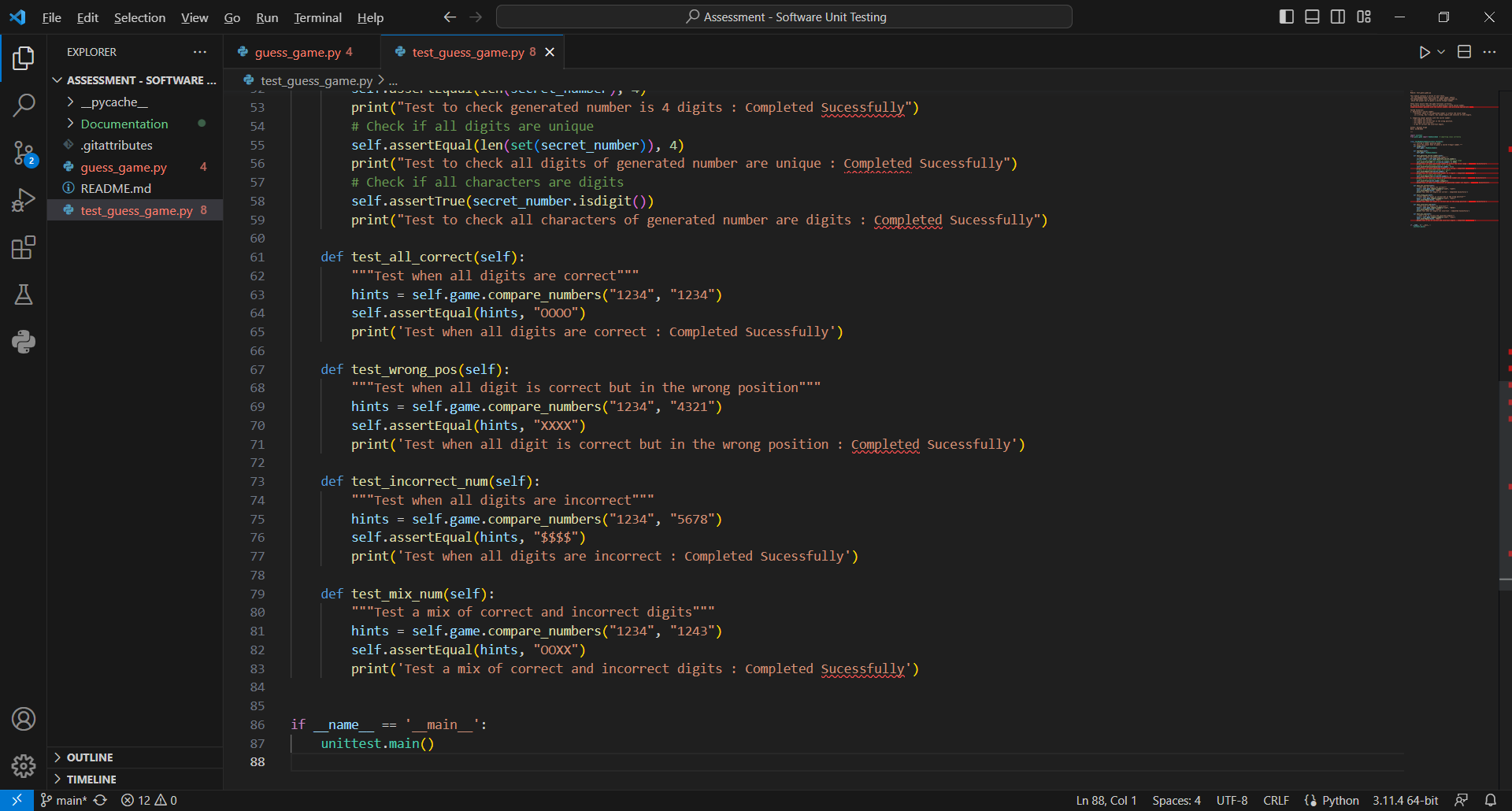


Figure 3:Test\_For\_All\_Possible\_Condition

A screenshot of a computer program

Description automatically generated

Figure 4:Output

# Automated Unit Testing:

## Pylint:

The code for guess\_game.py and test\_guess\_game.py has been rated 10.00/10 which represents the coding has been done in respect after adhering all of the comments of pylint.

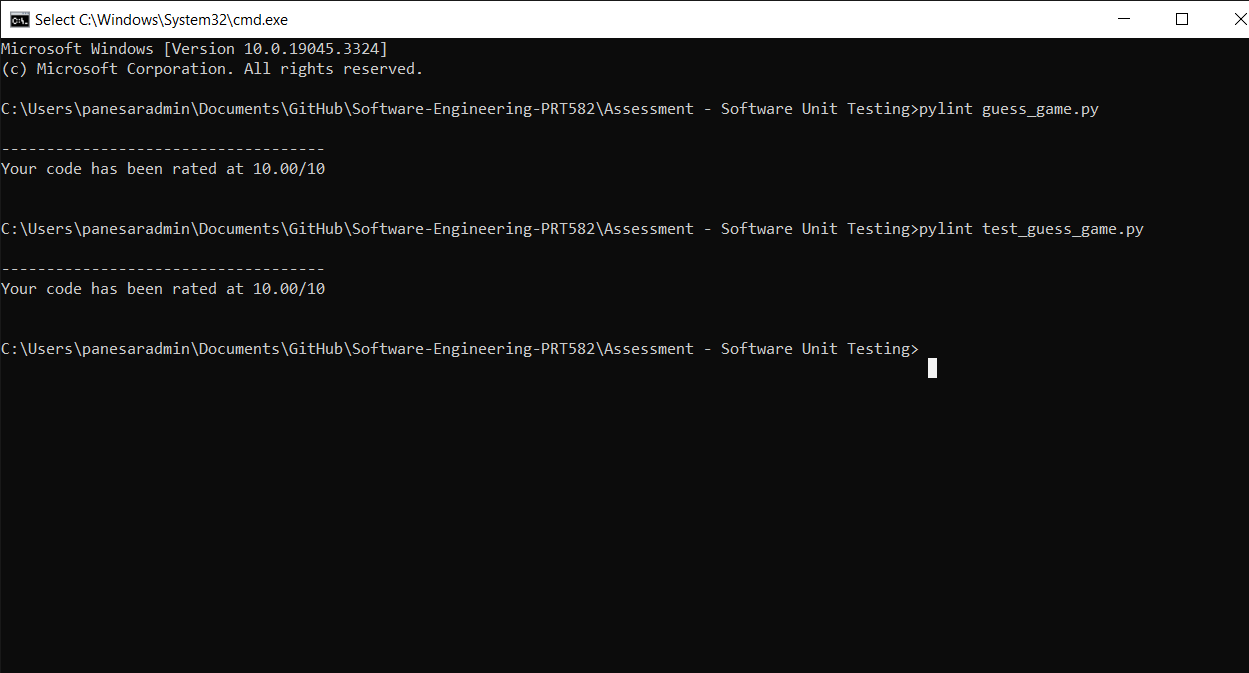
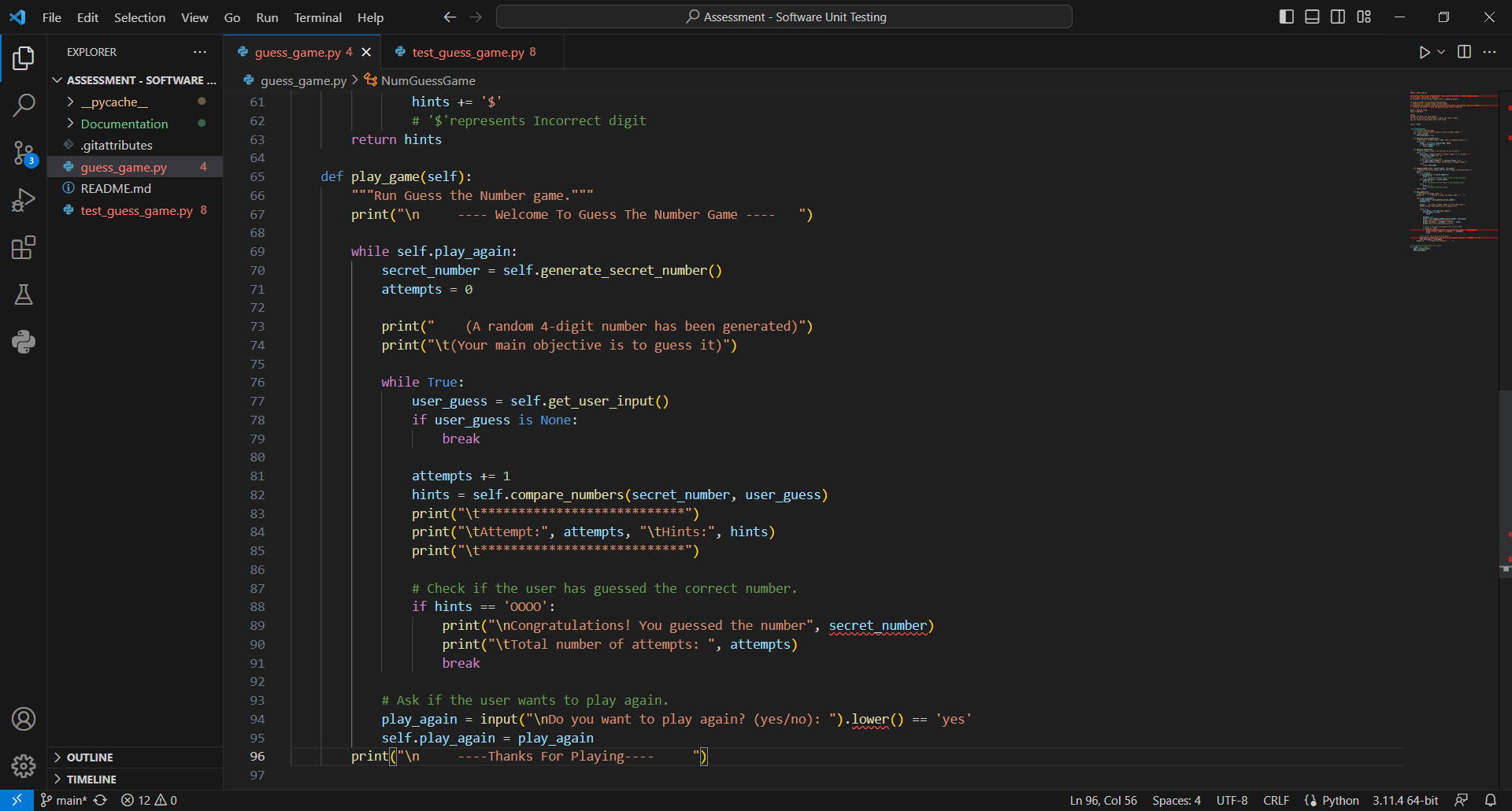


Figure 5: Pylint\_Score

# Code Implementation:

The code for the Number Guessing Game has been implemented following the TDD approach. It includes the NumGuessGame class, which encapsulates the game logic. The code is structured and modular, making it easy to test and maintain. Below is snippet for the play\_game() function to demonstrate the code implementation done.



# Conclusion

In conclusion, the development of the Number Guessing Game has been successful, thanks to the adoption of Test-Driven Development and the utilization of an automated unit testing tool. This approach has ensured the reliability of the code, and all requirements have been met.

## Lessons Learned

Throughout the development process, several lessons have been learned:

* **TDD Benefits**: Test-Driven Development greatly improves code quality and helps catch issues early in the development cycle.
* **Code Modularity**: Structuring code in a modular fashion enhances testability and maintainability.
* **Automated Testing**: Automated unit testing tools streamline the testing process and provide quick feedback.

Moving forward, it is important to continue practicing TDD and automated testing to maintain code quality and reliability in future software projects.