

PRT582 Software Unit Testing Report

Min SHI(s366786)

Table of Contents

1. [Introduction](#_bookmark0) 3
2. [Process](#_bookmark1) 5
3. [Conclusion 1](#_bookmark2)0

# Introduction

**Objectives:**

1. Develop a "Guess the Number" game that randomly generates a four-digit number.

2. Implement a feedback mechanism for the player to understand the accuracy of their guesses.

3. Display the number of attempts taken once the game concludes.

4. Allow players the option to either quit or play again once they've guessed correctly.

5. Ensure players have the ability to quit the game at any point.

**Requirements:**

1. Number Generation: The program must be capable of generating a random four-digit number. Each digit should range from 0 to 9. Repeated digits are allowed.

2. Game Loop: The program must continuously prompt the player for a guess until the correct number is guessed or the player decides to quit.

3. Feedback Mechanism: When a guess is made:

- For every digit that is correct and in its correct position, display a ‘circle’.

- For every digit that is correct but in a different position, display an ‘x’.

- Display this feedback immediately after each guess.

4. Attempts Counter: Keep track of how many attempts the player has made. Display this count once the correct number has been guessed.

5. End Game Options:Once the correct number has been guessed:

- Display the message, "You guessed correctly!"

- Ask the player if they want to quit or play again. The game should restart if they choose to play again.

6. Quit Option: The player should have the option to quit at any point. This can be achieved through a specific input (e.g., typing "quit").

**Automated Unit Testing Tool:** pytest

- Description:`pytest` is a mature full-featured Python testing tool that simplifies the testing process, making it easy to write simple and scalable test cases.

- Features:

1.Allows for easy and compact test writing using Python's assert statement.

2.Powerful and flexible fixture model.

3.Can discover and run tests in projects without needing a specific structure or naming convention.

4.Extensible via plugins.

**Use in the Project**: In the "Guess the Number" game, `pytest` is used to:

1.Test the random number generation to ensure it always produces valid four-digit numbers.

2.Test the feedback mechanism to confirm that 'circle' and 'x' hints are generated correctly based on the player's guesses.

3.Test the game loop, ensuring that the correct prompts are given and the game restarts or concludes as expected.

4.Validate edge cases, such as invalid inputs or quitting in the middle of the game.

# 

# Process

I will break down how TDD and automated unit testing tools are used together, step by step during the process of this game testing:

1. **Start with Requirements:**

Before diving into the code, I need a clear understanding of what needs to be achieved – this is usually defined in requirements or user stories.

2. **Write a Test:**

Based on the requirements, I write a test for a small portion of the functionality. At this point, I haven't written the code to provide this functionality, so the test will fail. This is intentional. In TDD, it's important to see a test fail to ensure not getting false positives later on.

3. **Run the Test:**

Using an automated unit testing tool, like `pytest`, run the test. Since the feature isn't implemented yet, the test should fail. This confirms that the test is set up correctly and it's indeed checking the intended functionality.

4. **Write the Code:**

I write the minimum amount of code needed to make the test pass. This ensures that I am only writing code necessary for the requirements and not adding potentially extraneous functionality.

5. **Run the Test Again:**

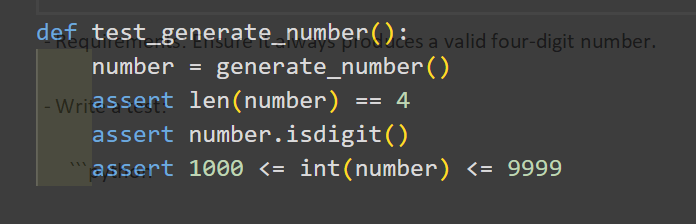
Using the automated testing tool, I run the test again. If it passes, it means the code meets the criteria defined in the test. If not, it's back to the drawing board to adjust and correct.

To use Test Driven Development (TDD) with `pytest`, I'll write tests for the features I mentioned and then ensure my code meets those test conditions. Below is a basic outline on how I could set up my tests:

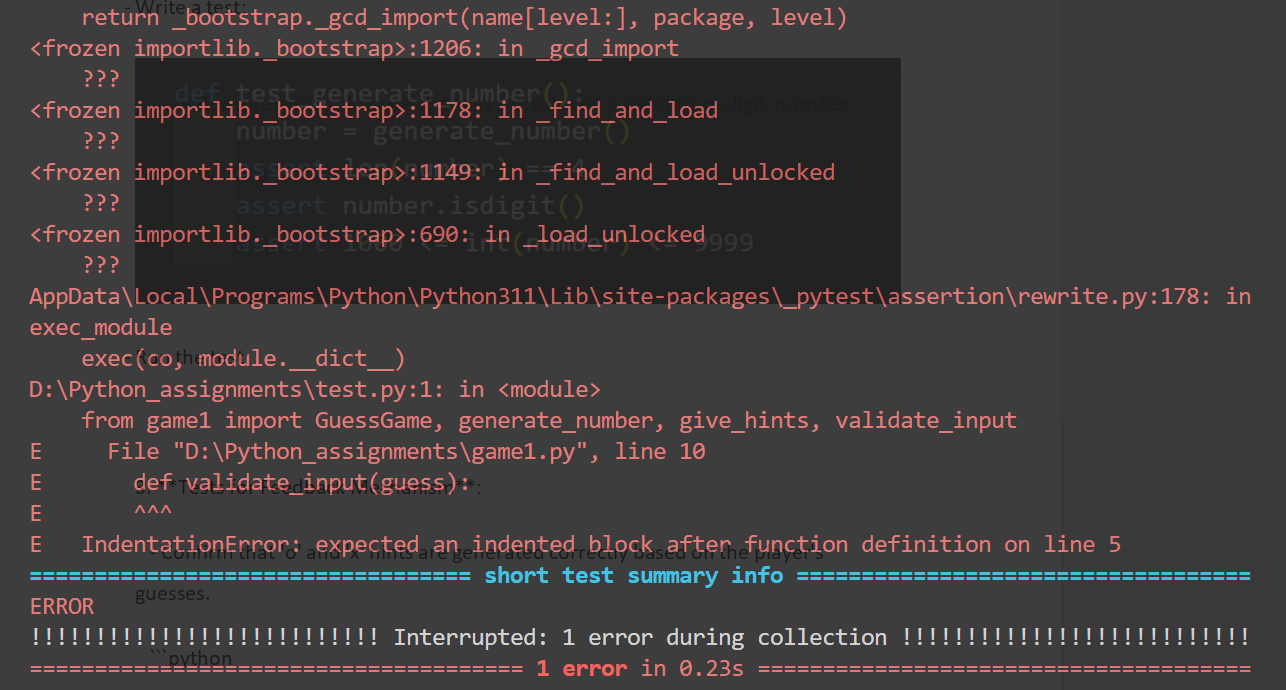
**1. Tests for Random Number Generation:**

- Requirements: Ensure it always produces a valid four-digit number.

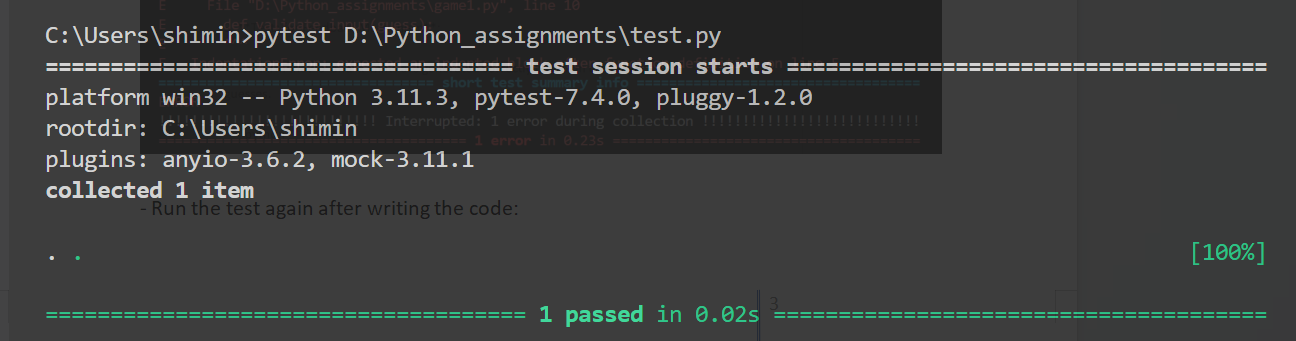
- Write a test:



- Run the test:



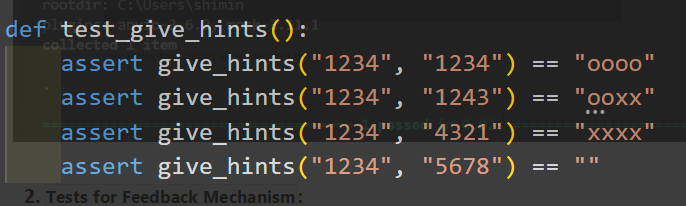
- Run the test again after writing the code:



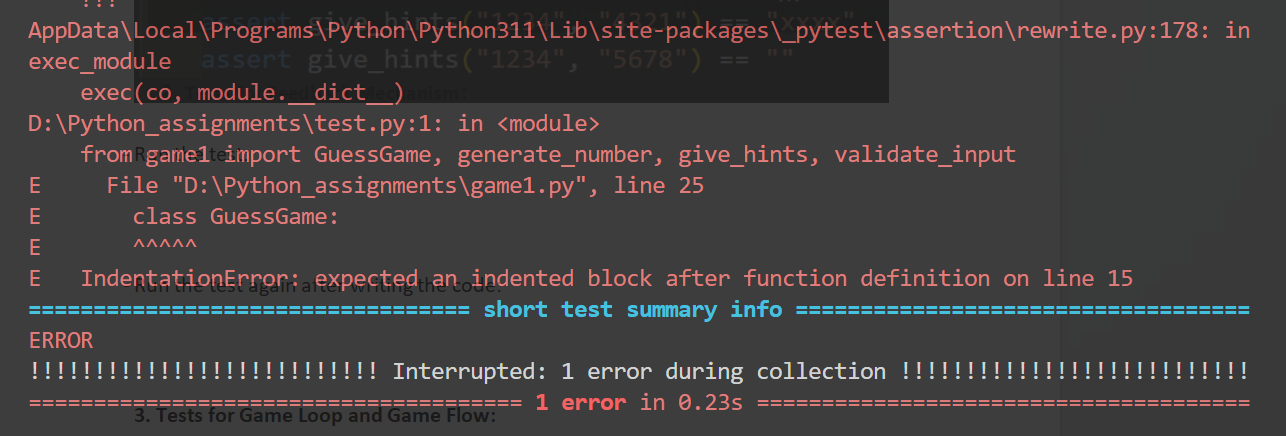
**2. Tests for Feedback Mechanism:**

- Requirements: Confirm that 'o' and 'x' hints are generated correctly based on the player's guesses.

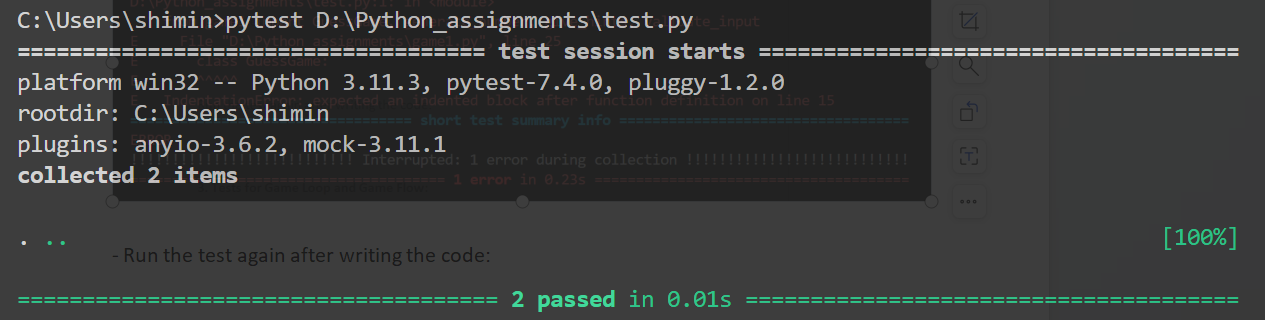
- Write a test:



- Run the test:



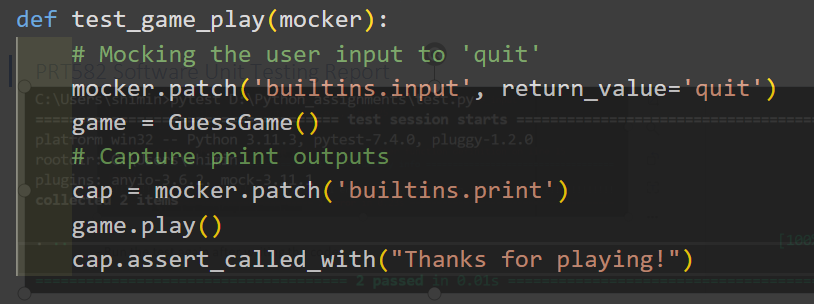
- Run the test again after writing the code:



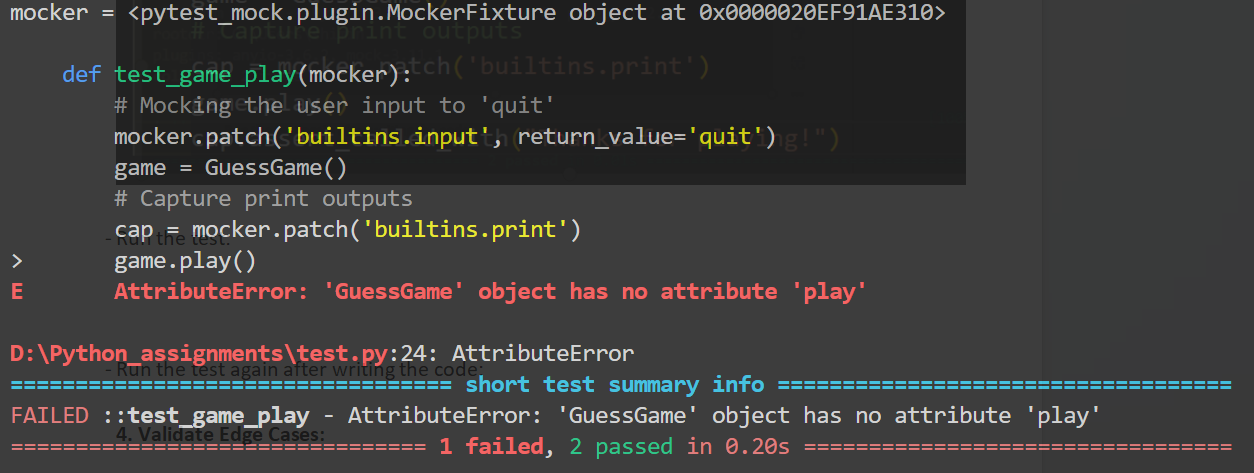
**3. Tests for Game Loop and Game Flow:**

- Requirements: Mocking function and testing the game loop.input.

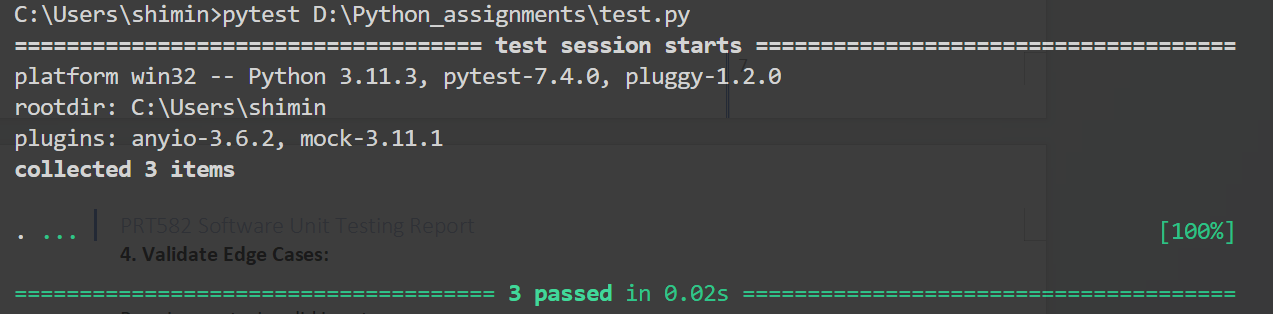
- Write a test:



- Run the test:



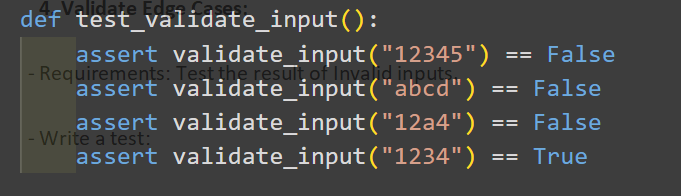
- Run the test again after writing the code:



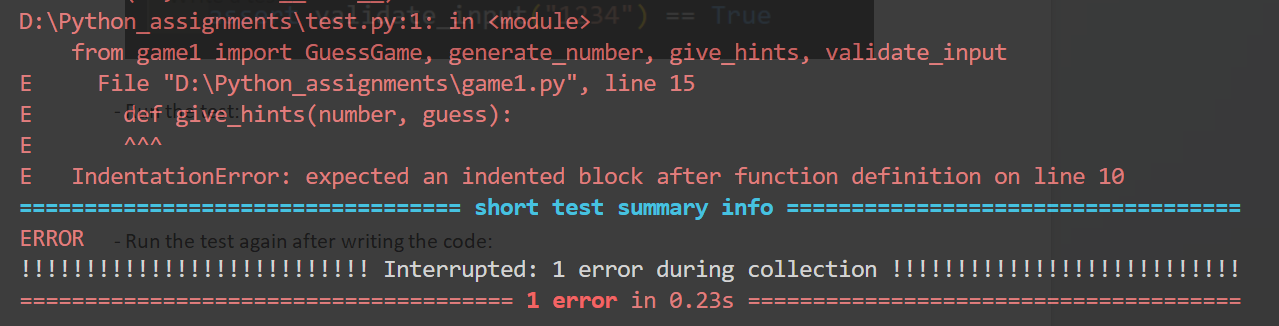
**4. Validate Edge Cases:**

- Requirements: Test the result of Invalid inputs.

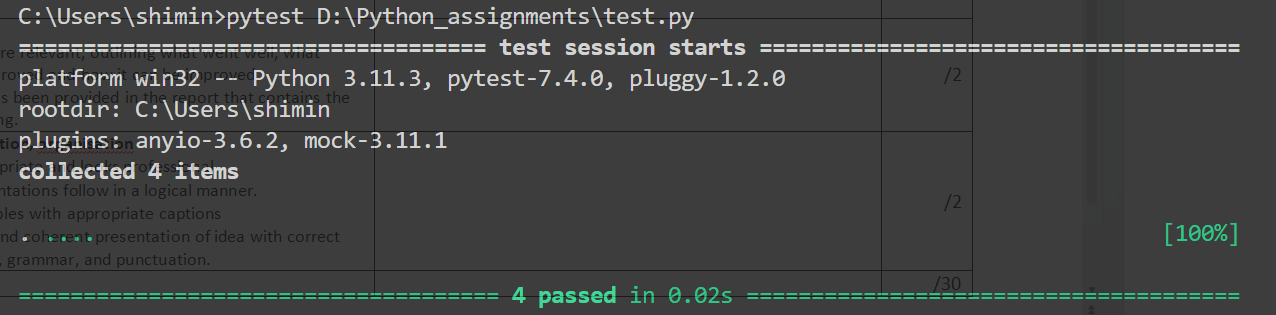
- Write a test:



- Run the test:



- Run the test again after writing the code:



# Conclusion

**What Went Well:**

1. Test-First Approach: TDD ensured that the code was written with testing in mind. Before even starting the actual coding, the conditions for success were clearly defined in the tests. This not only set clear expectations but also provided a roadmap for development.

2. Modularity and Function Decomposition: The game had various functions like `generate\_number`, `validate\_input`, and `give\_hints`. This modularity was beneficial for TDD because it allowed for testing specific functionalities in isolation.

3. Automated Testing with pytest: Using `pytest` made it straightforward to run tests frequently. Its concise syntax and clear output made it easy to interpret test results and adjust the code accordingly.

4. Error Handling: TDD brought forward scenarios that might've been overlooked during traditional development. For instance, the test for validating inputs ensured that only valid numbers were accepted.

**What Needs to Be Improved:**

1. Mocking Interactions: Testing interactive components like the game loop was a bit tricky. Mocking user inputs and outputs is a necessary but complex step in the testing process.

2. Test Coverage: There might still be some edge cases not covered. For instance, if there's a feature added later on (like a hint feature after N attempts), new tests need to be written for it.

3. Refactoring and Code Structure: The game code was written in a single script. While it might work for smaller games, as the game grows, it would be more maintainable to split functionalities into separate modules. Additionally, while TDD encourages refactoring after writing tests, more emphasis should be placed on regularly revisiting code to ensure optimal structure and performance.

**How It Can Be Improved:**

1. Enhanced Mocking: Familiarity with advanced mocking libraries or techniques can help simulate user interactions better. This is crucial for games or applications that rely heavily on user input/output.

2. Continuous Integration (CI): Incorporating CI tools can automatically run tests whenever changes are made, ensuring that any new code does not break existing functionality.

3. Regular Code Reviews: Even with TDD, peer code reviews can help identify areas of improvement, potential pitfalls, and optimizations. They also provide different perspectives which can be invaluable for improving code quality.

4. Expand Test Scenarios: Always consider more diverse test scenarios. This might include testing under different system conditions, adding more complex user interactions, or even simulating rare edge cases.

5. Documentation: While TDD provides a sort of "living documentation", it's essential to have clear, user-friendly documentation. This includes not only how the game works from a user's perspective but also developer documentation on the game's architecture, dependencies, and more.

In conclusion, while TDD and `pytest` provided a structured and methodological approach to developing the "Guess the Number" game, continuous evaluation and iteration are essential for improving both the game and the development process.

Github: https://github.com/shidami/software\_unit\_testing.git