Contents

[Introduction: 3](#_Toc143899279)

[Requirements 3](#_Toc143899280)

[Process 3](#_Toc143899281)

[Test Driven Development (TDD) 3](#_Toc143899282)

[Automated Unit Testing 3](#_Toc143899283)

[Test Cases 5](#_Toc143899284)

[Conclusion: 9](#_Toc143899285)

[GitHub link: 9](#_Toc143899286)

Introduction:

Using Python's TDD framework, this project aims to create a "Guess the Number" game. To play, player has to predict a four-digit number created at random. During the play, the system offers crucial hints in the form of symbols denoted by ‘0’ and ‘X’. The symbol '0' denotes the occurrence of a valid digit in the accurate location, whereas the symbol 'X' shows the presence of a valid digit in an incorrect position. The player has the option to keep guessing until they either get it right or decide to stop. The game also records number of attempts made by player to make the correct guess and provides the option to terminate the game at any point.

## Requirements

The requirement for number guessing name are:

1. Generate four digit numbers randomly.
2. Allow player to guess the randomly generated number.
3. Drop hints in the form of symbols denoted by ‘0’ and ‘X’ where The symbol '0' indicates an accurate digit in the correct spot, while 'X' indicates its presence.
4. Show the number of attempts made by player to make the correct guess.
5. Provides option for the player to terminate the game at any point.

# Process

## Test Driven Development (TDD)

Tests are written before the actual code is written in a Test Driven Development cycle. This method is useful for making certain that all code has been thoroughly tested and is performing as expected.

## Automated Unit Testing

I used Python's built-in unittest where Code modules (methods, functions) are tested independently using automated unit tests. To guarantee that the game's features work as intended, i built a suite of tests using the unittest library. Unit testing facilitated the automation of conducting these tests and determining their success or failure.

I also used flake8 to maintain code quality and Python style guidelines. This program automatically examines our codebase for PEP 8 compliance, assuring readability and consistency.

**Output after the player guessed right number with multiple attempts**

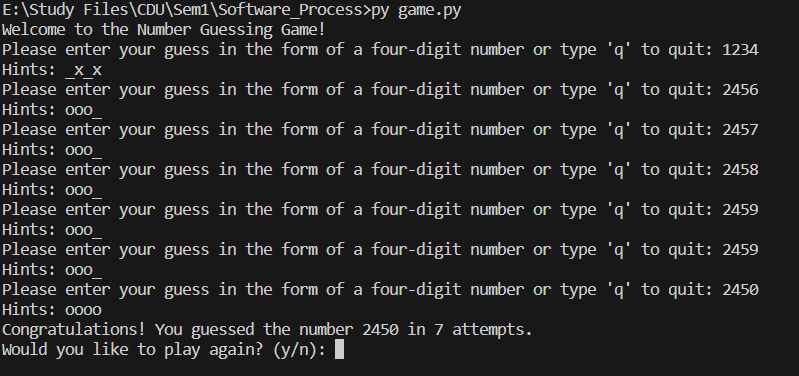


Figure Game output

## Test Cases

1. Test Case for Valid Input
2. Test Case for Invalid Input:
3. Test Case for Exact Match:
4. Test Case for Partial Match:
5. Test Case for No Match:
6. Test Case for Game Win:
7. Test Case for Game Lose:

**Out put of overall test case:**

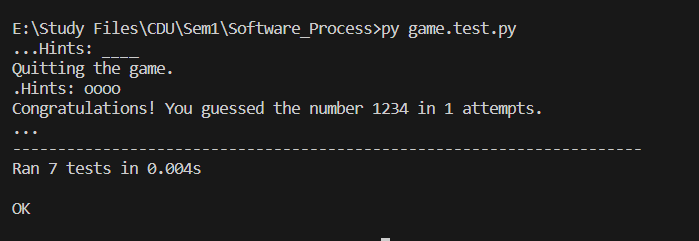
****

Figure overall test case output

Let's explore the usage of TDD and automated testing  
The code starts with the definition of the unittest-derived test class ‘TestNumberGuessingGame; . The many test cases in this class ensure that the game module works as intended.

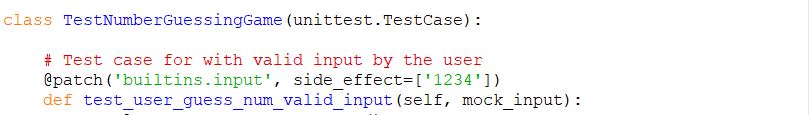


Figure Test class

The @patch decorator is used to simulate user input in the first test case (test\_user\_guess\_num\_valid\_input). The predicted result is what is returned back by user\_guess\_num().

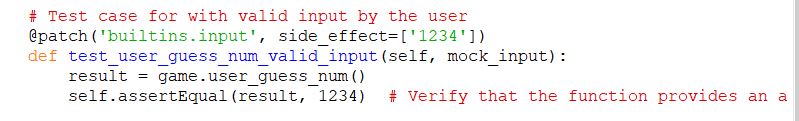


Figure Test case1

Test case for inavalid guess , (test\_user\_guess\_num\_invalid\_input) It simulates several inputs from the user and records the resulting output with the help of patch and mock\_print. After each incorrect input, the test verifies that the error message is printed and that the method returns None.

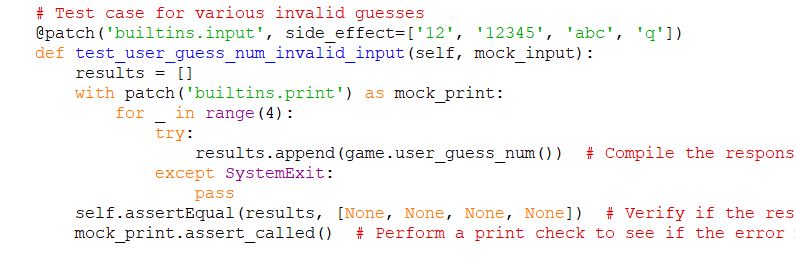


Figure test case 2

(test\_compare\_random\_num\_exact\_match) is a function-checking test case for Exact Match.Compare the random number with the guessed number using the compare\_random\_num() function. It ensures that only accurate responses are sent returned.

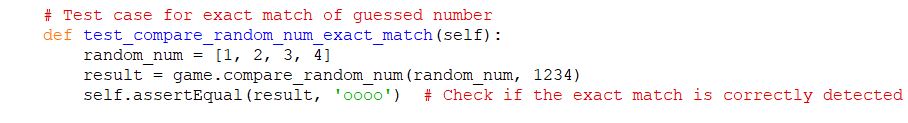


Figure test case 4

Test\_compare\_random\_num\_partial\_match is a case that checks for partial matches between the random number and the guessed number. It ensures that the function provides the intended return.

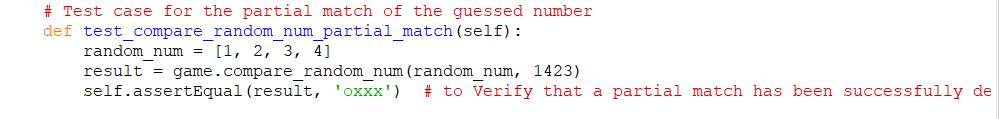


Figure test case 5

The test case test\_compare\_random\_num\_no\_match verifies whether or not the function provides the expected response ('\_\_\_\_') when there is no correlation between the random number and the guess.

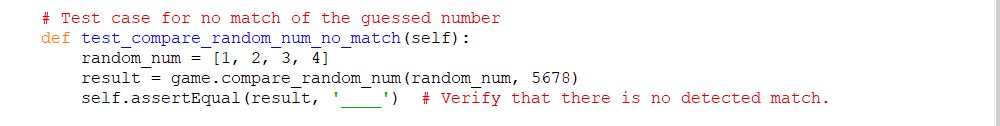


Figure test case 6

The @patch mocks random number generation and user input in the test case (test\_game\_win) for a game win simulation. This test ensures that the game.game() function correctly returns the required number of attempts to win.

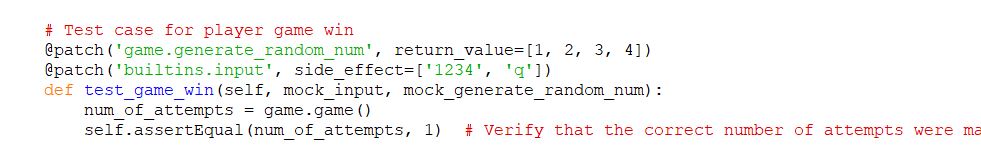


Figure test case 7

The test\_game\_lose case represents a game-ending loss. It checks if the function game exists using @patch,The number of attempts till a defeat is expected is provided by the game() function.

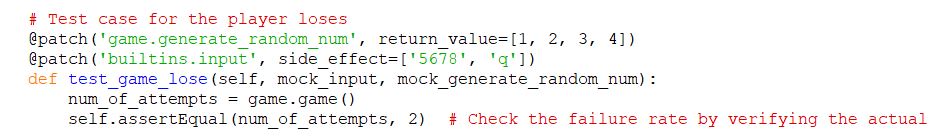


Figure test case 8

The unittest.main() function executes all the tests defined before the if \_\_name\_\_ == '\_\_main\_\_': block. The automated unit testing tool is useful at this stage. All test cases are run and the results are reported.

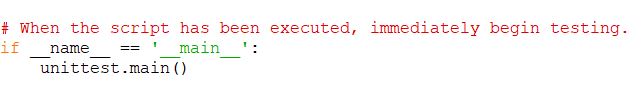


Figure main function

# Conclusion:

This project highlighted how Test Driven Development and automated unit testing work. Using the TDD methodology, I was able to create the "Guess the Number" game in a methodical fashion while staying focused on the requirements. Python's unittest framework, an automated unit testing tool, was important in ensuring the correctness of the implementation and finding bugs.

**Lesson learned**

The project taught me the importance of using TDD to discover flaws early and prevent them from becoming more complex. Writing tests beforehand helped me implement each function by defining its purpose. This method modularized the code, making it easier to manage and debug. TDD also promoted incremental development, which streamlined the process.

**Successes and Challenges:**

TDD effectively identified edge cases and corner scenarios that I may not have considered otherwise. Testing the game's response to invalid input helped me handle such scenarios graciously. Writing tests for the 'evaluate\_guess' function, which gives hints, was difficult due to the intricacy of possible combinations. Breaking the problem into smaller, more manageable components and establishing test cases for multiple circumstances solved this difficulty.

**Areas for Improvement:**

Additional modularization could improve code structure. Separate modules for random numbers, guess evaluation, and the game loop would structure and manage the codebase. Clearer instructions, error messages for invalid inputs, and interface improvements could improve the user experience.To enhance the game, I suggest adding more features to increase engagement. This could include time challenges, multiplayer modes, or difficulty levels. Players may find a graphical user interface (GUI) more intuitive and appealing. Expanding to longer digit sequences could add complexity and depth to the game.

Overall, the project demonstrated the value of TDD in developing stable, trustworthy software. It demonstrated the value of working in smaller chunks, creating thorough tests, and upgrading the code iteratively depending on the results of those tests. Thanks to this opportunity, I now have a deeper understanding of TDD and its significance in the software development process. The ups and downs encountered on the way led to a more complete picture of best practices in software design and testing.

## GitHub link: