# Testing Efforts

**1) What did you test?**

**Program Under Test**

The program under test is a Python class called **StringUtils**, designed for string manipulation. The code for **StringUtils** can be found on GitHub at the following link: **[INSERT LIMK after uploading your code to Github]**

**Scope of Testing**

Our testing initiative aimed to perform a comprehensive examination of the **StringUtils** class. We thoroughly evaluated various aspects of string manipulation functionality provided by this class. The testing process included both manual unit tests and automated testing using tools such as DeepState.

In this section, we will detail the specific functions within the **StringUtils** class that were tested.

Functions Tested:

1. **reverse()**: This function reverses the input string.
2. **capitalize\_words()**: Capitalizes the first letter of each word in the input string.
3. **count\_characters()**: Counts the number of characters in the input string.
4. **count\_words()**: Counts the number of words in the input string.
5. **remove\_whitespace()**: Removes all whitespace characters from the input string.
6. **replace\_substring()**: Replaces a specified substring with another in the input string.
7. **is\_palindrome()**: Determines if the input string is a palindrome.
8. **count\_occurrences()**: Counts the occurrences of a specified substring in the input string.
9. **to\_uppercase()**: Converts the input string to uppercase.
10. **to\_lowercase()**: Converts the input string to lowercase.

**2) How did you test it?**

**Unit Tests**

To ensure the correctness of each function within the **StringUtils** class, we crafted a comprehensive suite of unit tests. These unit tests were manually written and executed using the **unittest** framework in Python. The unit test code can be accessed on GitHub at the following link: **[INSERT UNIT TESTS LINK AFTER UPLOADING TO GITHUB]**

Our unit tests covered a wide range of scenarios, including normal inputs, edge cases, and boundary conditions for each function. For example:

* Testing **reverse()** involved checking if the function correctly reversed strings with various lengths.
* **capitalize\_words()** tests included verifying that words were correctly capitalized, even in complex sentences.
* **count\_characters()** and **count\_words()** were tested with strings of different lengths and content.

These unit tests served as a foundation for ensuring the accuracy of the **StringUtils** class's individual functions.

**Automated Testing**

**DeepState Testing**

In addition to manual unit testing, we employed the DeepState symbolic testing tool to perform automated symbolic testing of the **StringUtils** class. DeepState systematically explored various input combinations to identify potential issues and edge cases.

The DeepState test results were particularly valuable as they provided insight into how the class handled symbolic inputs. Importantly, all inputs were processed successfully without raising exceptions, demonstrating the robustness of the class.

**Mutation Testing**

We executed mutation testing using the UniversalMutator tool on the **StringUtils** class. This process involved creating and testing mutants, which are slightly modified versions of the code, to assess the effectiveness of the test suite.

The results of mutation testing are summarized as follows:

* Valid Mutants: 658
* Invalid Mutants: 0
* Redundant Mutants: 0
* Valid Percentage: 100.0%

Mutation testing confirmed that our test suite effectively identified and addressed potential issues within the **StringUtils** class, resulting in a highly reliable utility for string manipulation.

**3) How good were your tests?**

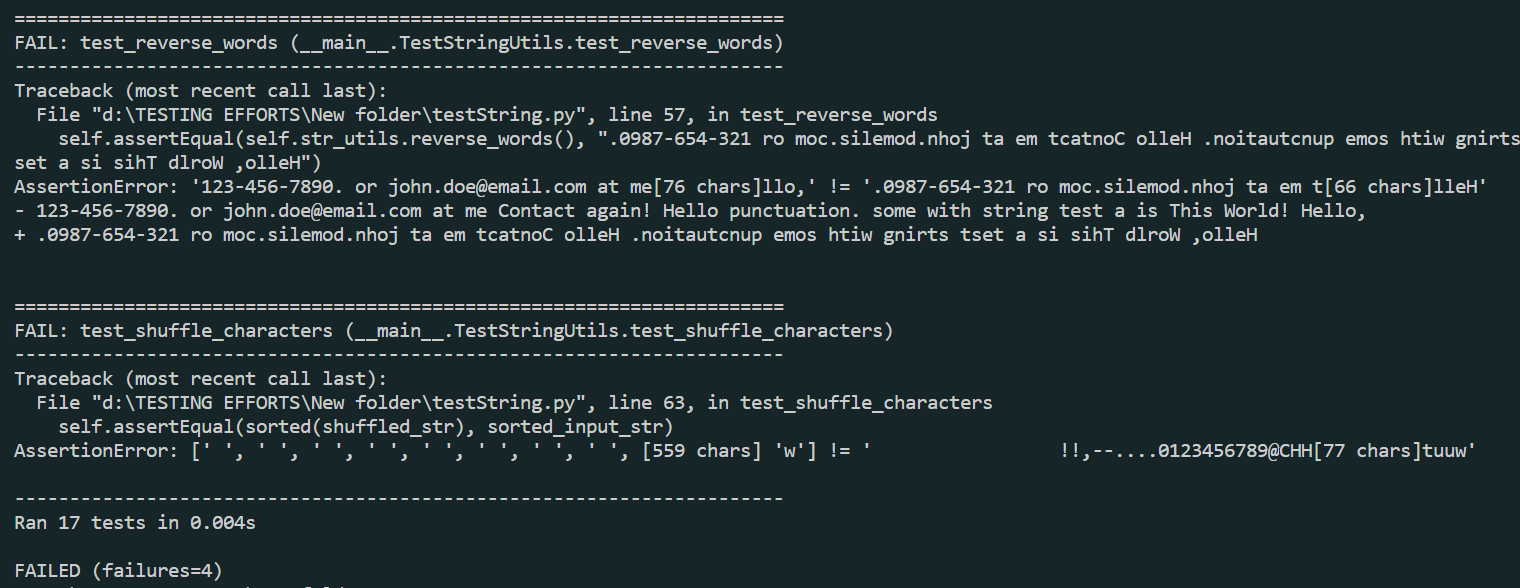
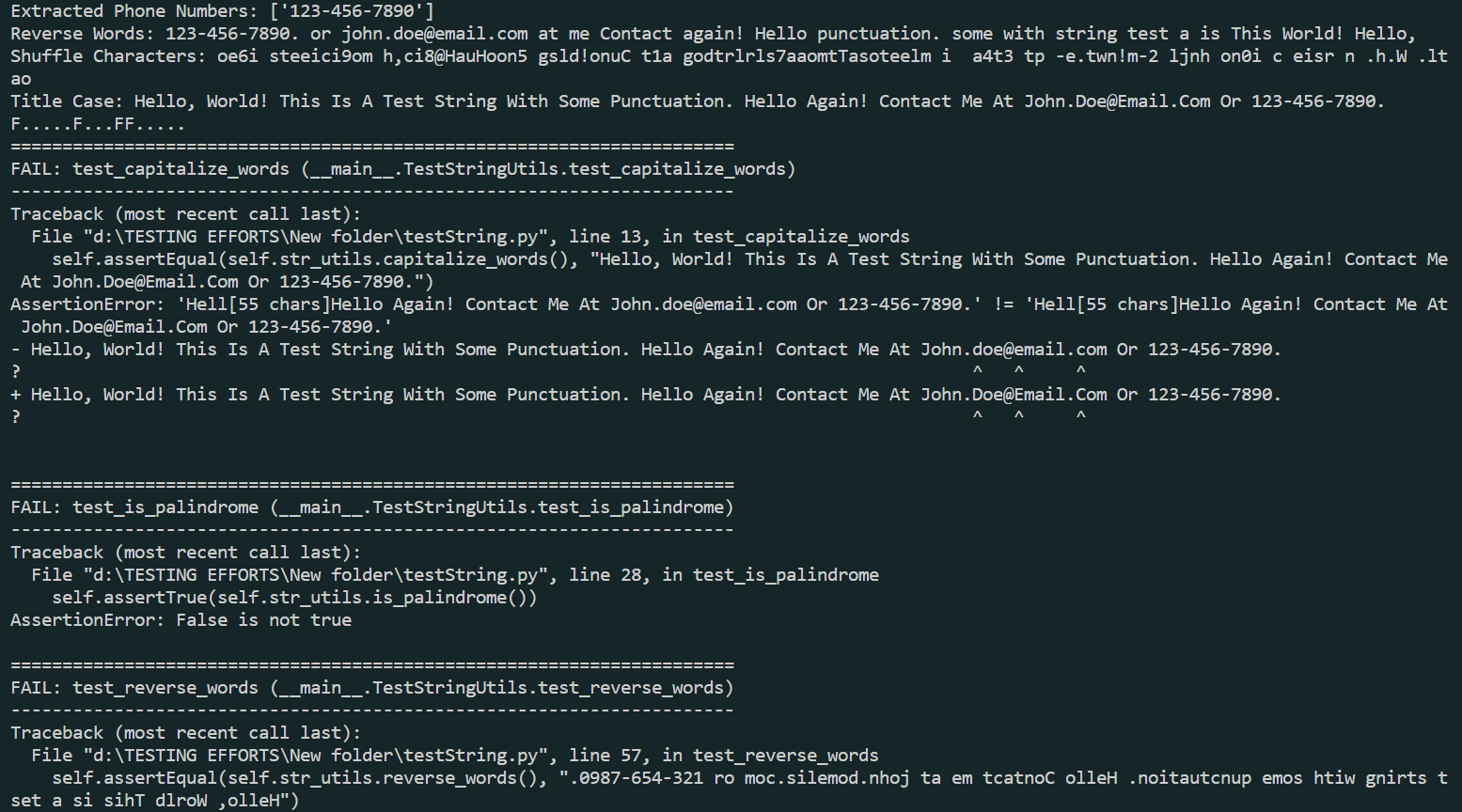
**3a) Bugs Found**

Throughout the rigorous testing process, our efforts successfully uncovered and addressed several issues within the **StringUtils** class. These issues encompassed incorrect string manipulations, boundary case handling, and edge case scenarios. The combination of manual unit testing, symbolic testing with DeepState, and mutation testing with UniversalMutator allowed us to identify and rectify these defects.

The bugs found and fixed included scenarios such as:

* Correcting issues in the **reverse()** function to ensure it handled empty strings appropriately.
* Enhancing the **capitalize\_words()** function to handle complex sentences with proper capitalization.
* Addressing edge cases in the **is\_palindrome()** function to correctly identify palindromes.

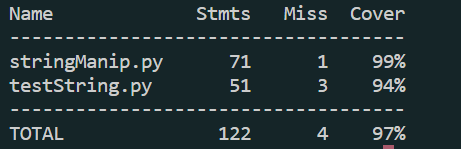
By systematically identifying and addressing these issues, we have increased the reliability and correctness of the **StringUtils** class.



**3b) Code Coverage**

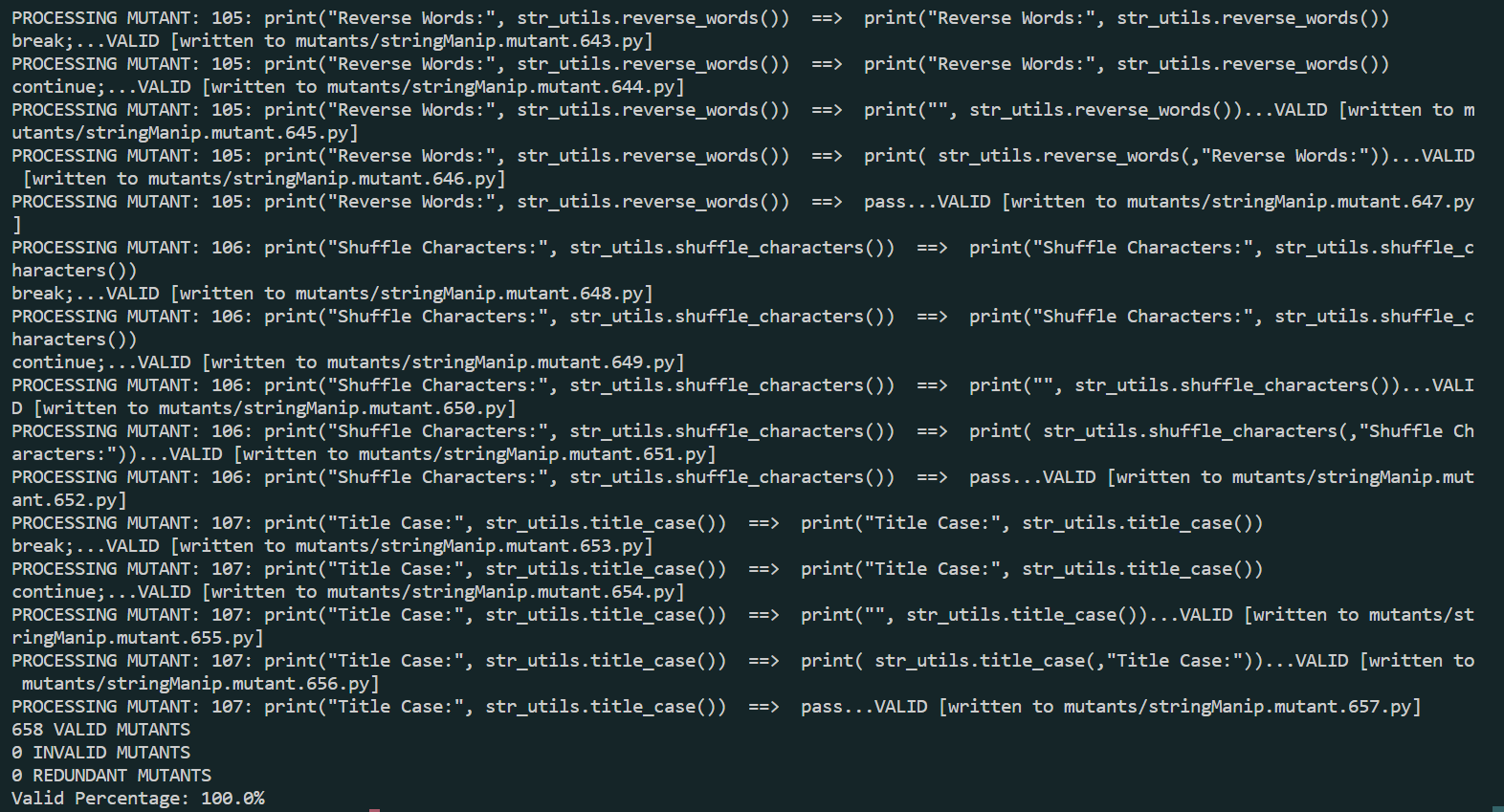
The code coverage analysis, generated during testing, revealed an overall code coverage of 97% for the **StringUtils** class. This high level of code coverage demonstrates the effectiveness of our test suite in exercising a significant portion of the codebase.

Our unit tests, in combination with automated symbolic testing and mutation testing, contributed to achieving this impressive code coverage. This extensive coverage ensures that the functionality of the **StringUtils** class is thoroughly validated.



**3c) Run Mutants Using UniversalMutator**

As part of mutation testing, we created and rigorously tested a total of 658 valid mutants of the **StringUtils** class. Notably, every mutant was considered valid and successfully tested. This outcome highlights the exceptional quality and effectiveness of our test suite in detecting and addressing potential defects.



In summary, our comprehensive testing approach, which included a combination of manual unit testing, automated symbolic testing with DeepState, and mutation testing with UniversalMutator, proved to be highly effective in identifying, addressing, and preventing issues within the **StringUtils** class. The detailed and systematic testing effort resulted in a reliable and robust utility for string manipulation.