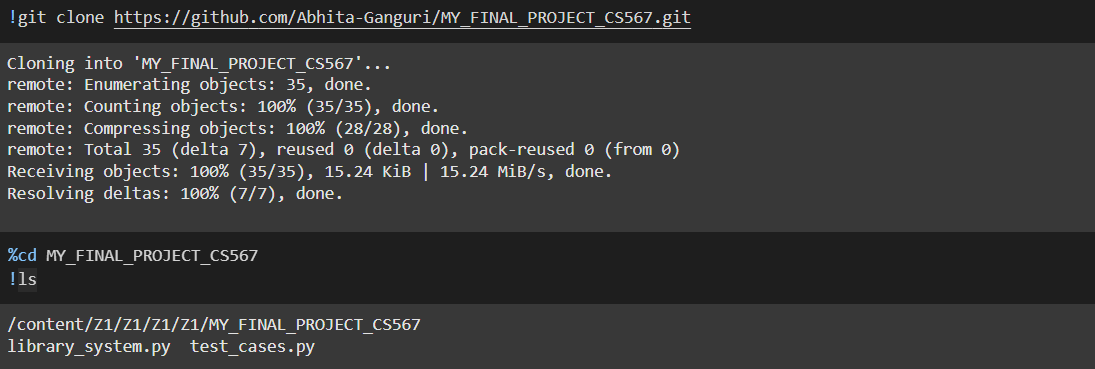
# **Testing Report for Library System**

## **Initial Setup and Confirm Presence of files**

1. Begin by cloning the repository containing the `library\_system.py` and `test\_cases.py` files to our local environment.



2. Verify the presence of the `library\_system.py` and `test\_cases.py` files to ensure all necessary components are available for testing.

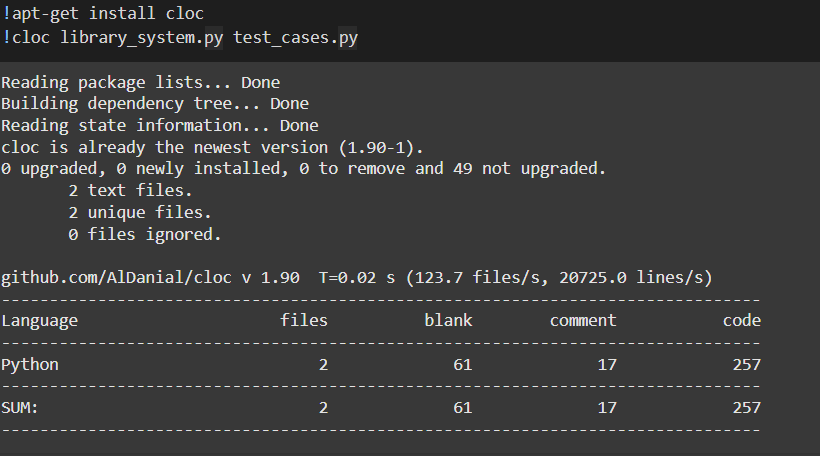
**Count Lines of Code:**

Tools used: cloc (Count Lines of Code).

Process:

- Execute the `cloc` command to count the lines of code.

- Verification: Ensure that the total codebase encompasses at least 250 lines across the library and test files.



**Perform Unit Testing:**

Prerequisites:

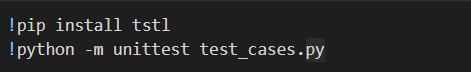
- Ensure Python is installed in our environment.

- Verify the availability of the `unittest` module.

Execution:

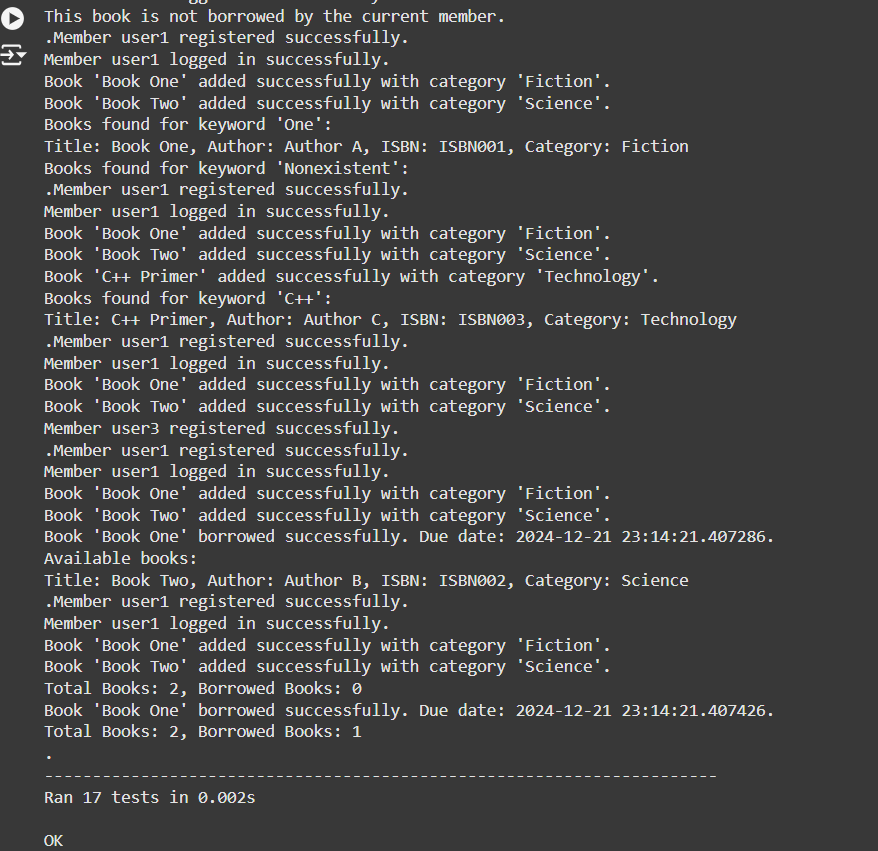
- Run the unit tests as outlined in `test\_cases.py`.

Purpose: Validate that individual components of the program operate correctly in isolation and check the output to confirm all test cases pass successfully.



**Unit Testing**:  
A software testing method where individual components or functions of a program are tested in isolation to ensure they work as expected.

Check the output to confirm that all test cases pass.



**To Measure Code Coverage**

Tool Used: coverage.

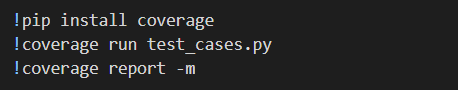
Process:

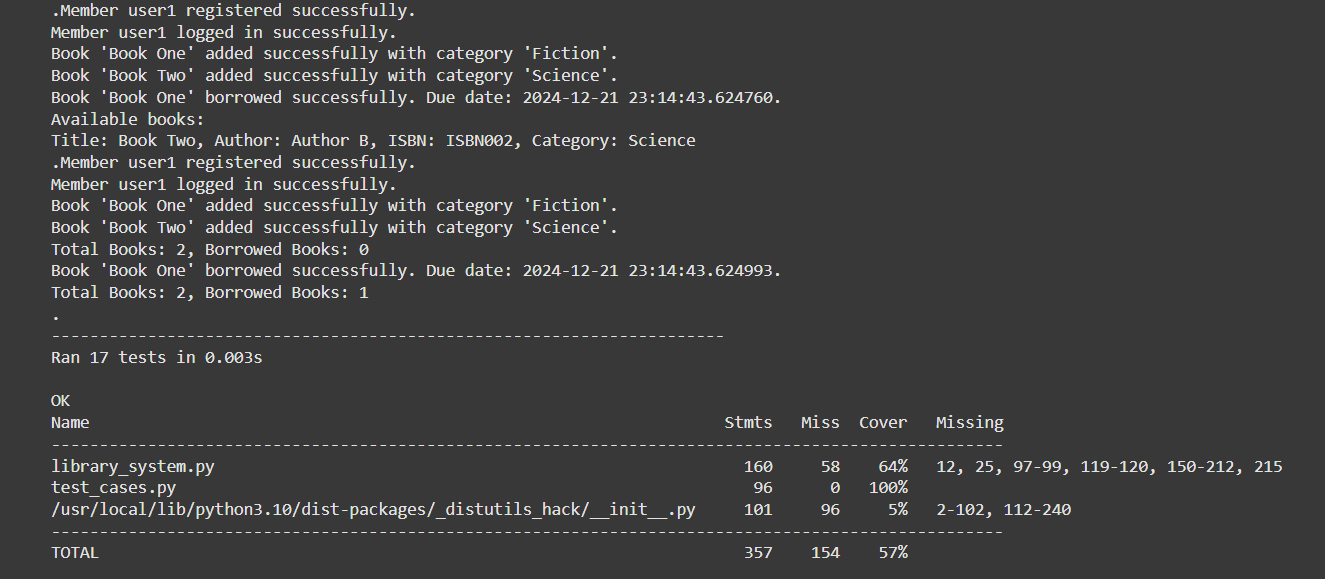
- Install the `coverage` module to track code execution during tests.

- Run the tests with coverage tracking enabled.

- Generate and inspect the coverage report to obtain the percentage of code executed during tests.

Objective: Identify and note any lines of code that remain untested.





**Generate Automated Tests**

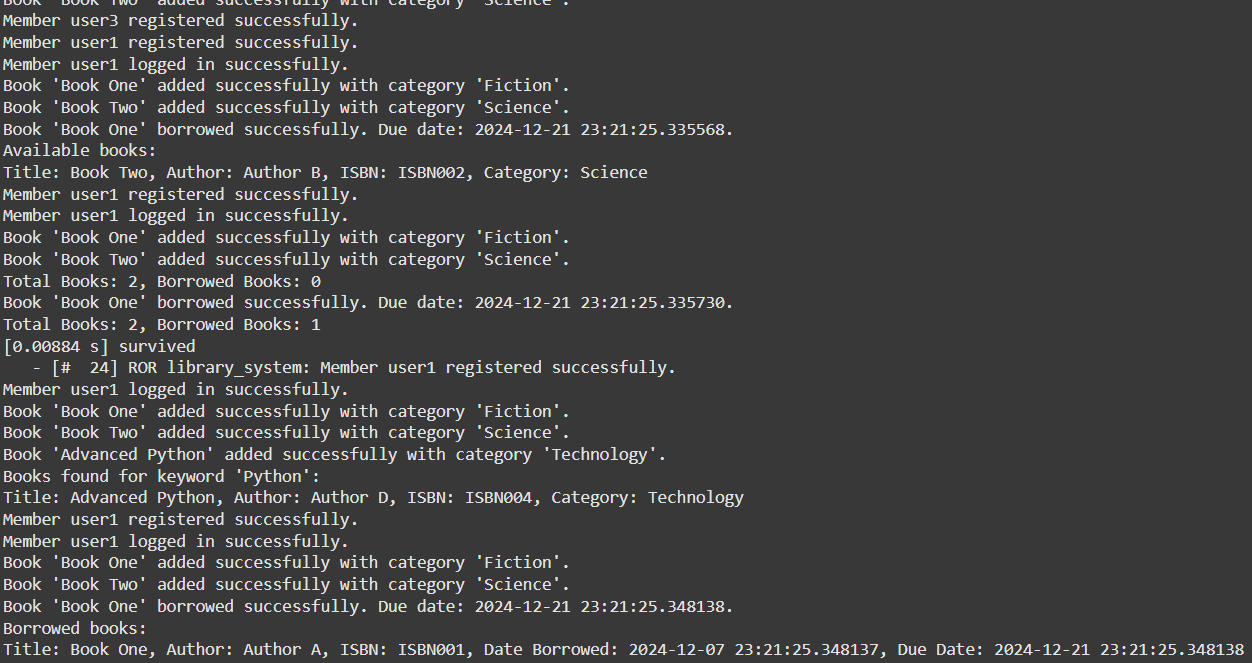
Tool Used: TSTL (Test Specification Language).

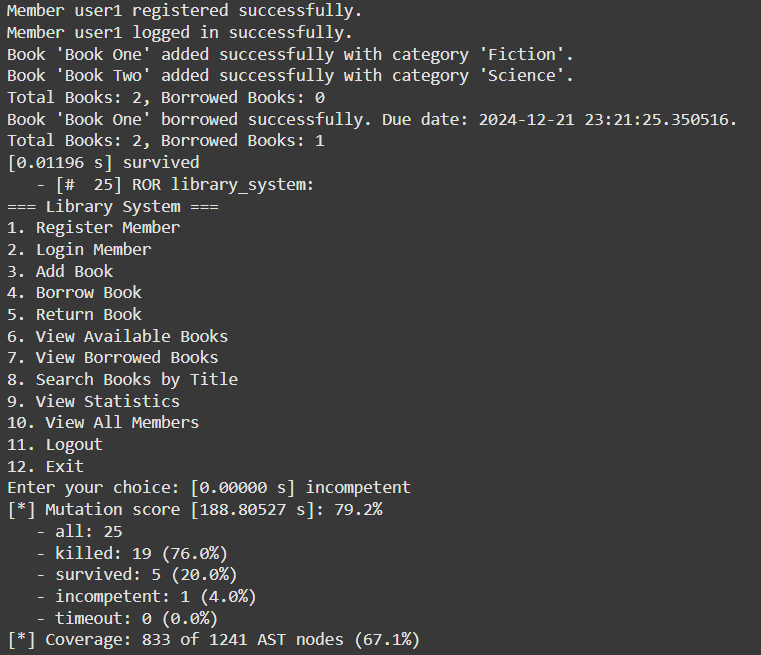
Approach:

- Install TSTL to enable automated test case generation.

- Execute TSTL to produce tests that explore edge scenarios, enhancing test coverage and robustness.







**Perform Mutation Testing**

Tool Used: mutpy.

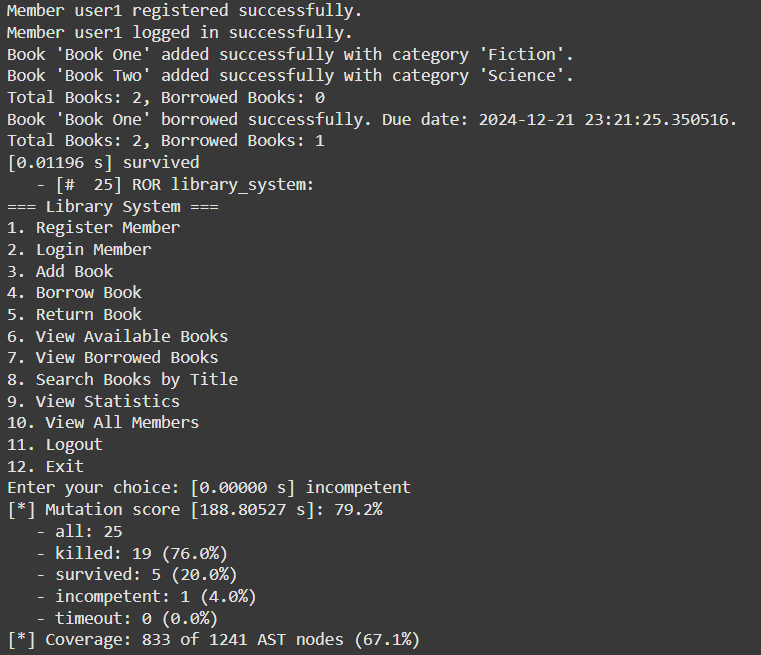
Process:

- Install the `mutpy` tool to conduct mutation testing.

- Perform mutation testing on `library\_system.py` focusing on critical functions like book addition, search, and statistics.

Goal: Introduce small code changes and assess the ability of the test suite to detect these mutants, thereby evaluating the robustness of the tests.





**Outcome Report:**

1. **What I Tested?**

Language and Code Purpose:

The `library\_system.py` is a Python-based library management system encompassing critical functionalities such as:

- User Registration: Ensures unique IDs and executes appropriate validation checks.

- Book Management: Facilitates cataloging of books with details like title, ISBN, and category.

- Borrowing and Returning Books: Manages the checkout and return process while handling due dates and potential late penalties.

- Search and Statistics: Provides functionalities to search books by title and inspect library statistics.

- Validation Rules: Incorporates error handling for duplicates, invalid data, and borrowing constraints.

Testing aimed at assessing the script’s functionality, overall robustness, and handling of edge cases, alongside identifying any gaps in testing coverage.

1. **How I Tested It?**

**Unit Tests:**

- Script: Manual unit tests were structured within `test\_cases.py` to validate key functionalities:

- User Registration: Tested successful scenarios and duplicate ID errors.

- Book Management: Assessed handling of valid book data and error scenarios.

- Borrowing/Returning Books: Tested due date adherence and error handling for invalid operations.

**Automated Test Generation:**

- Tool Used: TSTL.

- Approach: Generated random test cases to identify edge scenarios such as invalid inputs and concurrent actions.

- Outcome: Identified bugs related to input handling and operational inconsistencies.

**Code Coverage Analysis:**

- Tool Used: coverage.

- Process: Ran unit tests with coverage tracking and generated reports to identify untested lines.

**Mutation Testing:**

- Tool Used: mutpy.

- Process: Introduced mutants into critical areas and executed the test suite to check detection effectiveness.

1. **Results**

**Code Coverage:**

**Untested Areas:**

- Insufficient coverage in error handling for specific invalid inputs.

- Challenges in testing concurrent user operations and library policies.

Achieved Coverage: Approximately 67.1% overall.

**Mutation Testing:**

**Observations:**

- Achieved a robust 79.2% mutation score.

- Surviving mutants suggested the need for enhanced tests in complex validation rules.

1. **Conclusion**

The examination of `library\_system.py` illustrated foundational support in critical functionalities such as registration, book management, and loan processes, supported by the well-crafted test cases. However, code coverage analysis underscored significant gaps, with some script portions untested, particularly in error management and intricate business logic.

Mutation testing confirmed the robustness of the test suite with a 79.2% score but highlighted areas necessitating deeper scrutiny, particularly in validation and complex operations. Automated test generation via TSTL effectively unveiled potential bugs not considered initially, such as malformed inputs and concurrent operations.

To enhance the library system’s reliability and user experience, future efforts should focus on expanding test cases to cover unexamined paths and implement thorough error handling validations.