**Aim:** Execute test cases for Structural testing of the software and prepare code coverage report.

**Description:** Structural testing, also known as white-box testing, focuses on verifying the internal structure, logic, and implementation of the code. The goal is to ensure that all possible execution paths are tested, edge cases are handled, and no untested code remains. Code coverage reports provide a quantitative measure of how much of the code has been tested, which helps identify untested areas.

**Approach:**

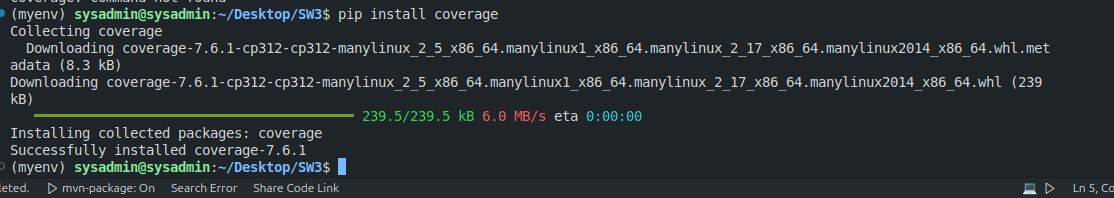
1. *Set up the environment:*
   * Install the required testing frameworks and code coverage tools (e.g., JUnit, PyTest
   * , Jest, or JaCoCo).
   * Ensure the software codebase is updated and compiled without errors.
2. *Identify test cases:*
   * Analyze the code structure to identify control flows, decision points, and loops.
   * Write or modify existing test cases to cover all branches, statements, and paths.
3. *Execute test cases:*
   * Run the prepared test cases using a testing framework.
   * Log the results to verify if the software behaves as expected.
4. *Generate a code coverage report:*
   * Use code coverage tools integrated with the testing framework to generate reports.
   * Examine coverage metrics such as statement, branch, and method coverage.
5. *Analyze and refine:*
   * Review the report to identify untested code sections.
   * Add new test cases or modify existing ones to achieve higher coverage.
6. *Documentation:*
   * Document the test execution process, coverage results, and any additional observations.
   * Save the coverage report for future reference.

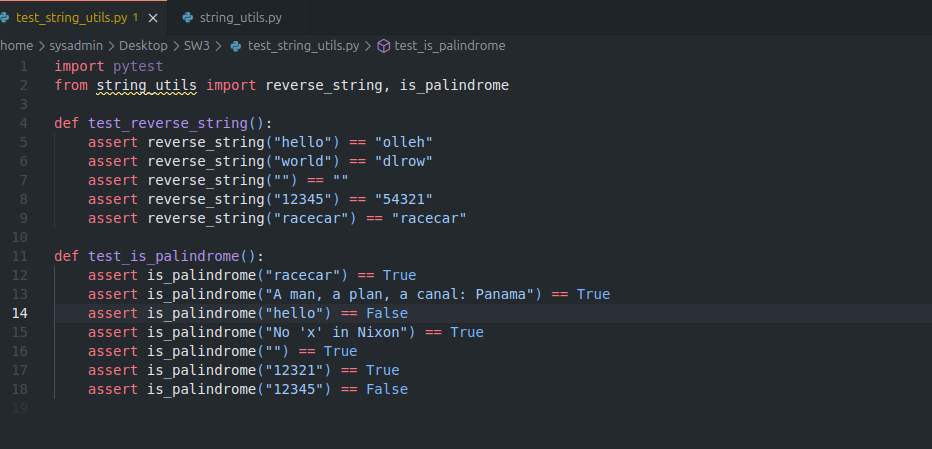
**Steps:**

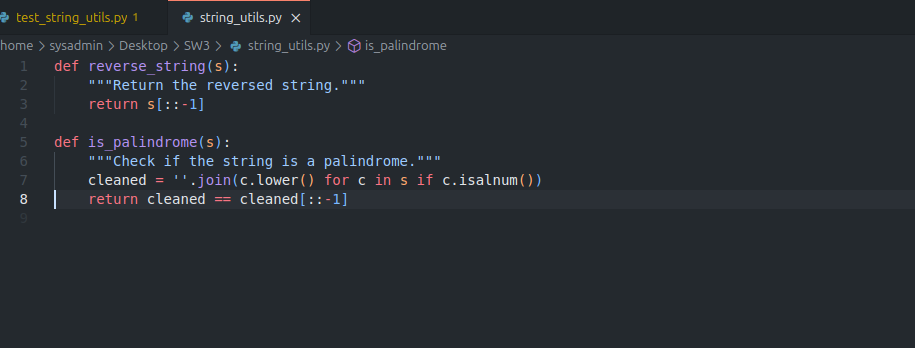
Install pytest and coverage modules, if not already installed:

*sudo apt update*

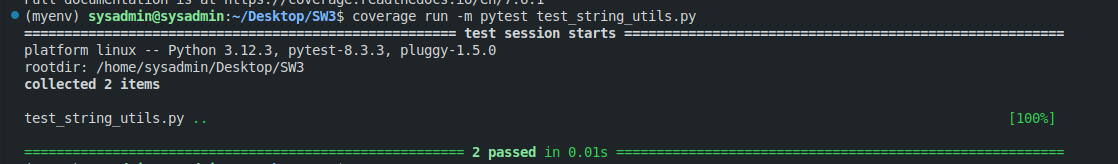
*pip install coverage*

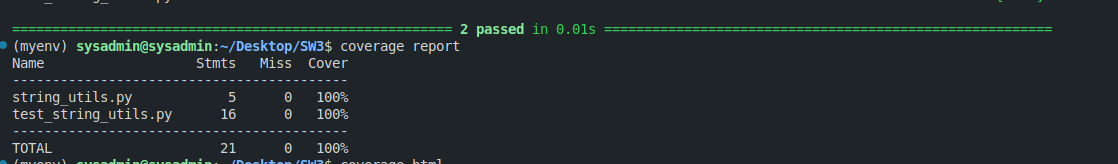
****

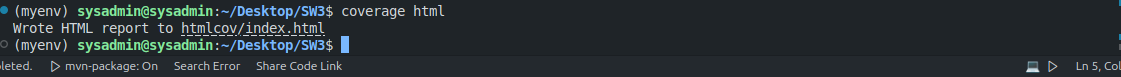
****

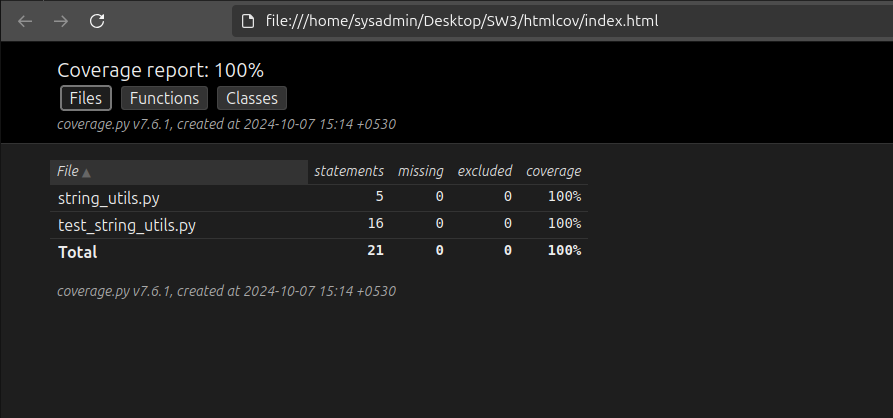
****

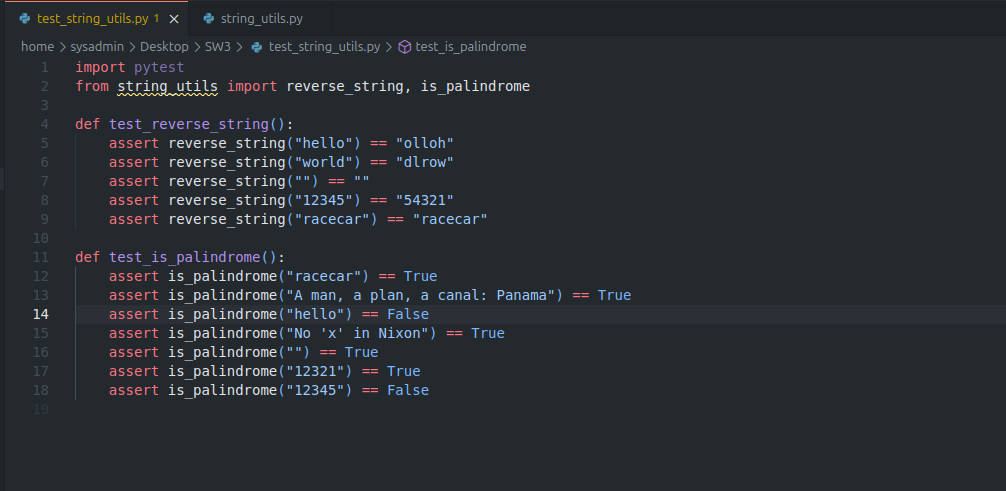
*To run: coverage run -m pytest filename.py*

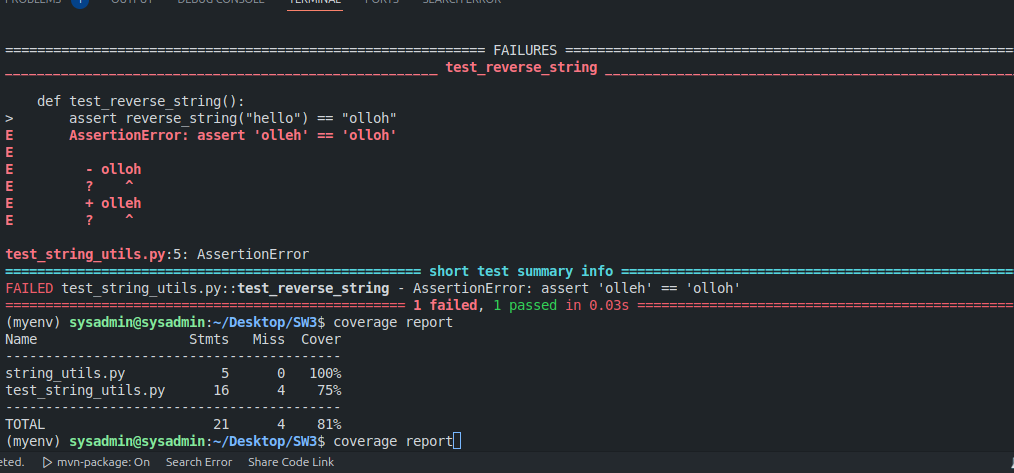
****

****

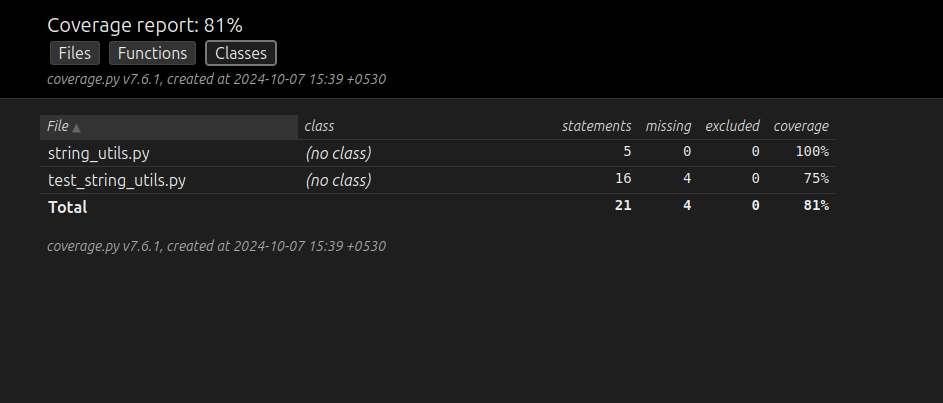
****

****

****

****

****

****

**Code:***string\_utils.py*def reverse\_string(s):

"""Return the reversed string."""

return s[::-1]

def is\_palindrome(s):

"""Check if the string is a palindrome."""

cleaned = ''.join(c.lower() for c in s if c.isalnum())

return cleaned == cleaned[::-1]

import pytest

from string\_utils import reverse\_string, is\_palindrome

*test\_string\_utils.py*

def test\_reverse\_string():

assert reverse\_string("hello") == "olloh"

assert reverse\_string("world") == "dlrow"

assert reverse\_string("") == ""

assert reverse\_string("12345") == "54321"

assert reverse\_string("racecar") == "racecar"

def test\_is\_palindrome():

assert is\_palindrome("racecar") == True

assert is\_palindrome("A man, a plan, a canal: Panama") == True

assert is\_palindrome("hello") == False

assert is\_palindrome("No 'x' in Nixon") == True

assert is\_palindrome("") == True

assert is\_palindrome("12321") == True

assert is\_palindrome("12345") == False

**Conclusion:**

* Structural testing ensures the internal logic of the software is reliable and robust.
* A detailed code coverage report provides insights into the quality of testing and highlights areas requiring further attention.
* Iterative improvement of test cases based on the report enhances overall software reliability.