PES UNIVERSITY

Department of Computer Science & Engineering

Software Engineering

UE23CS341A

Assignment 7 Submission

Name of the Student	Pranav Hemanth
SRN	PES1UG23CS433
Section	G
Department	CSE
Campus	RR

Code and Branch Coverage Analysis

UE23CS341A

Lab 7: Code and Branch Coverage Analysis

Objective:

To explore fuzz testing as a security and quality assurance technique by identifying and fixing bugs in input processing functions using Python and the Hypothesis fuzzing library.

Objective

By the end of this lab, we will

- Understand the difference between code coverage and branch coverage.
- Use coverage to measure, analyse, and improve test coverage.
- Interpret coverage reports and identify untested paths in code.
- Practice writing test cases that cover edge cases and decision branches.

Learning Outcomes

- Understand Code Coverage Concepts: You will differentiate between line coverage, branch coverage, path coverage, and function coverage to measure test completeness.
- Apply Coverage Analysis Tools: You will use Python's coverage library to generate reports, interpret coverage metrics, and identify untested code paths.
- Design Comprehensive Test Cases: You will write targeted test cases to achieve minimum 90% coverage by addressing edge cases and decision branches.

Introduction

What is Code Coverage?
Code coverage measures how much of your source code is executed during testing.

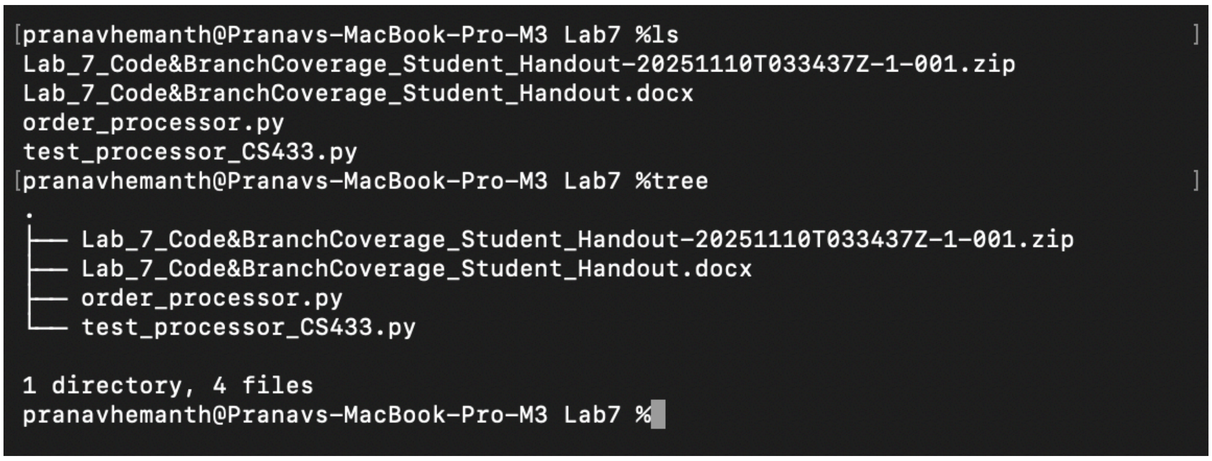
Types of coverage:

Type	Meaning
Line Coverage	How many lines of code were executed
Branch Coverage	How many branches (e.g., if, else) were followed
Path Coverage	How many unique paths through code were tested
Function Coverage	How many functions were invoked during tests

Why It Matters:

- High coverage improves confidence in correctness.
- Helps catch untested paths and dead code.
- Encourages test completeness, especially in critical systems

Folder Structure



```
[pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %ls                                    ]
 Lab_7_Code&BranchCoverage_Student_Handout-20251110T033437Z-1-001.zip
 Lab_7_Code&BranchCoverage_Student_Handout.docx
 order_processor.py
 test_processor_CS433.py
[pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %tree                                  ]
 .
 ├── Lab_7_Code&BranchCoverage_Student_Handout-20251110T033437Z-1-001.zip
 ├── Lab_7_Code&BranchCoverage_Student_Handout.docx
 ├── order_processor.py
 └── test_processor_CS433.py

 1 directory, 4 files
 pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %
```

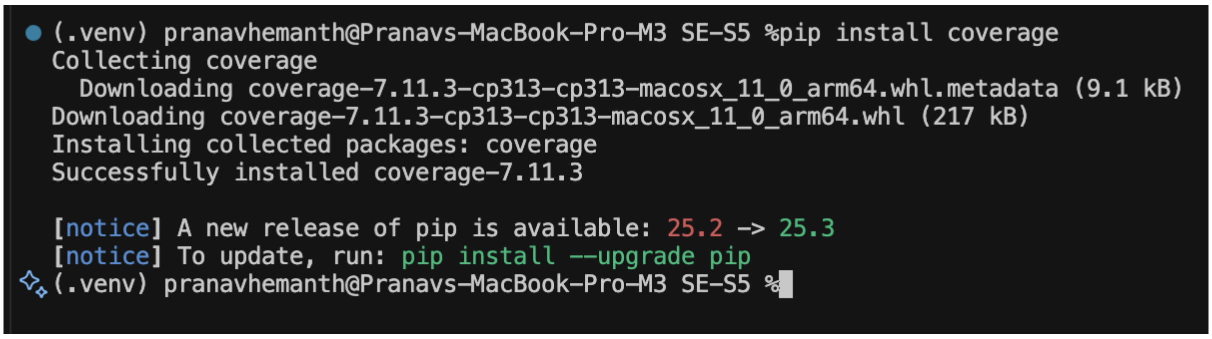
Deliverables:

- Coverage analysis & report (Screenshots)
- Test_processor.py (after completing the test cases)
- Reflection answers in the same document

Tasks:

1. Install dependencies

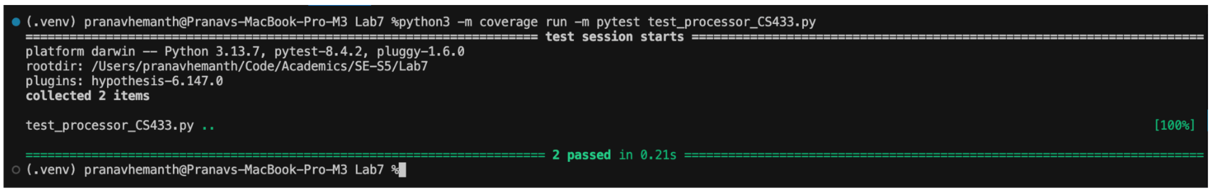
a. pip install coverage



```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 SE-S5 %pip install coverage
  Collecting coverage
    Downloading coverage-7.11.3-cp313-cp313-macosx_11_0_arm64.whl.metadata (9.1 kB)
  Downloading coverage-7.11.3-cp313-cp313-macosx_11_0_arm64.whl (217 kB)
  Installing collected packages: coverage
  Successfully installed coverage-7.11.3

  [notice] A new release of pip is available: 25.2 -> 25.3
  [notice] To update, run: pip install --upgrade pip
  (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 SE-S5 %
```

2. Run the original test file given:

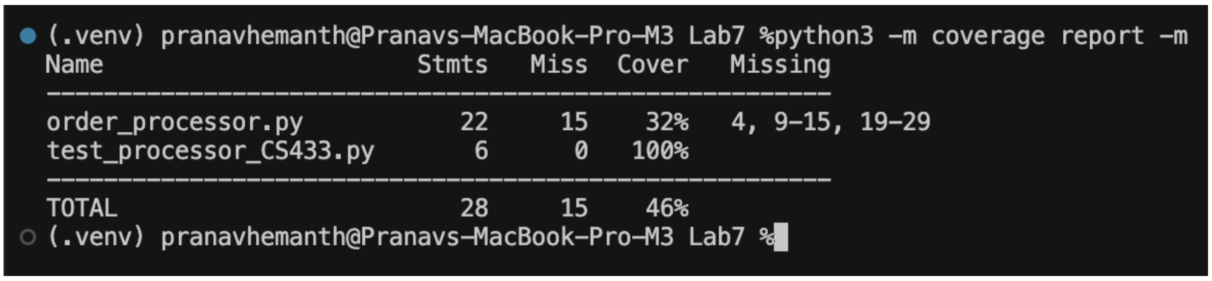
Command: python3 -m coverage run -m pytest test_processor_CS433.py



```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %python3 -m coverage run -m pytest test_processor_CS433.py
====================================================================== test session starts ======================================================================
platform darwin -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0
rootdir: /Users/pranavhemanth/Code/Academics/SE-S5/Lab7
plugins: hypothesis-6.147.0
collected 2 items

test_processor_CS433.py ..                                                                                                                                [100%]

======================================================================= 2 passed in 0.21s =======================================================================
○ (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %
```

3. Analyse the coverage report:

Command: python3 -m coverage report -m

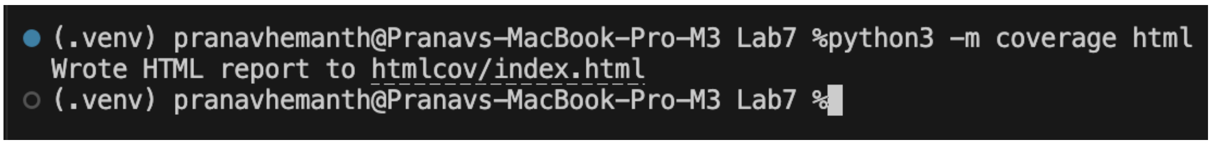


```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %python3 -m coverage report -m
  Name                         Stmts   Miss  Cover   Missing
  ----------------------------------------------------------
  order_processor.py              22     15    32%   4, 9-15, 19-29
  test_processor_CS433.py          6      0   100%
  ----------------------------------------------------------
  TOTAL                           28     15    46%
○ (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %
```

4. Run the coverage html file:

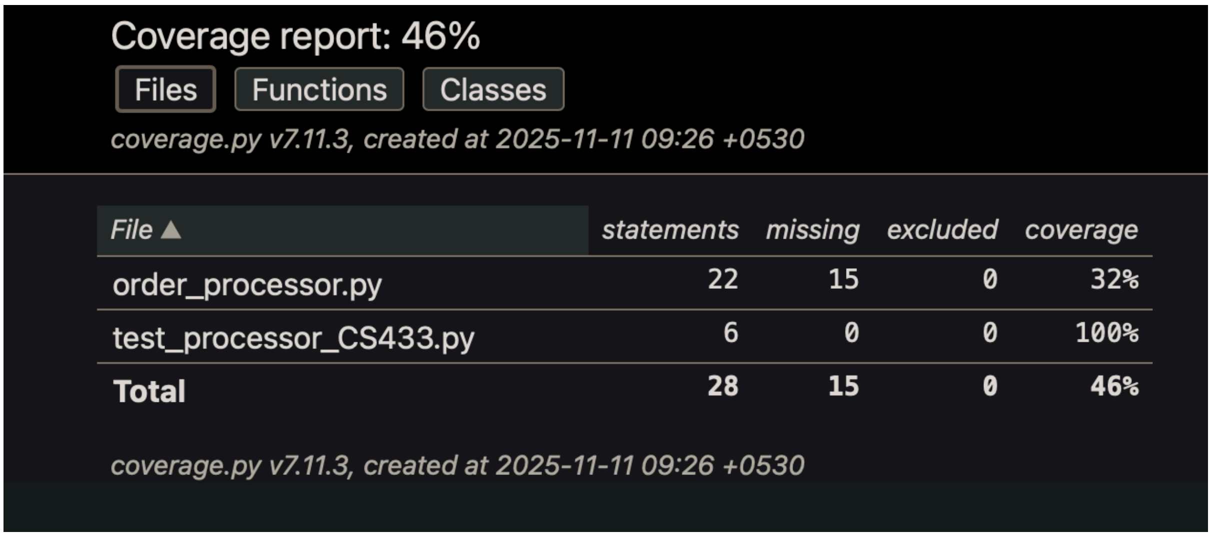
Command: python3 -m coverage html

Note: *After running python -m coverage html, you might see a file path printed in the terminal (e.g., htmlcov/index.html) instead of a clickable link. This path points to a folder named htmlcov, which contains the generated coverage report. To view the report, open the index.html file from the htmlcov folder located in your current directory manually.*



```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %python3 -m coverage html
  Wrote HTML report to htmlcov/index.html
○ (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %
```

Coverage report - Files:



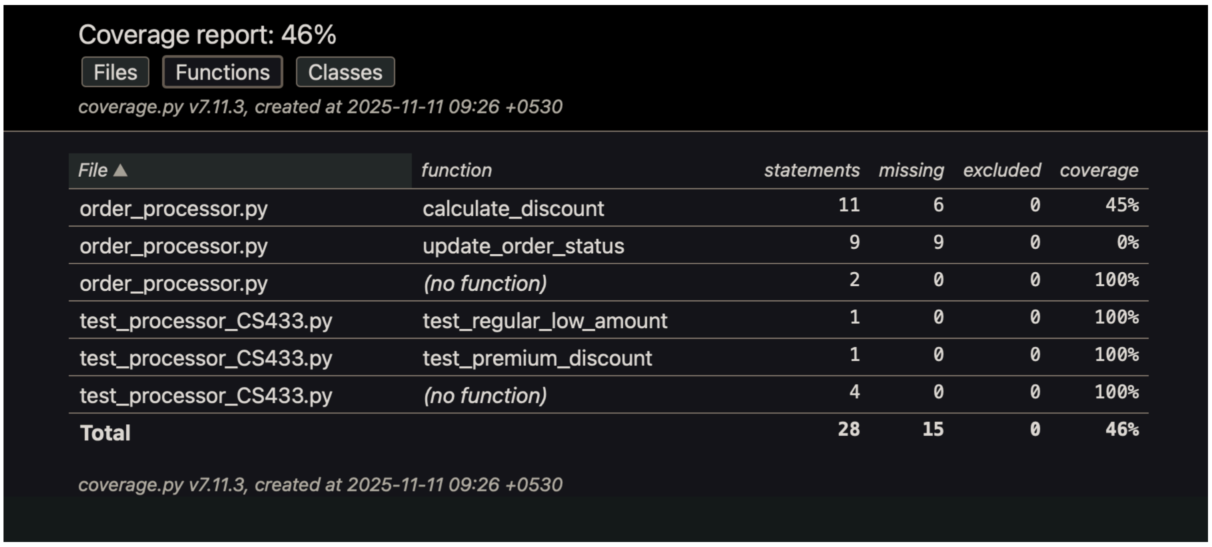
Coverage report: 46%

coverage.py v7.11.3, created at 2025-11-11 09:26 +0530

File ▲	statements	missing	excluded	coverage
order_processor.py	22	15	0	32%
test_processor_CS433.py	6	0	0	100%
Total	**28**	**15**	**0**	**46%**

coverage.py v7.11.3, created at 2025-11-11 09:26 +0530

Coverage report - Functions:



Coverage report: 46%

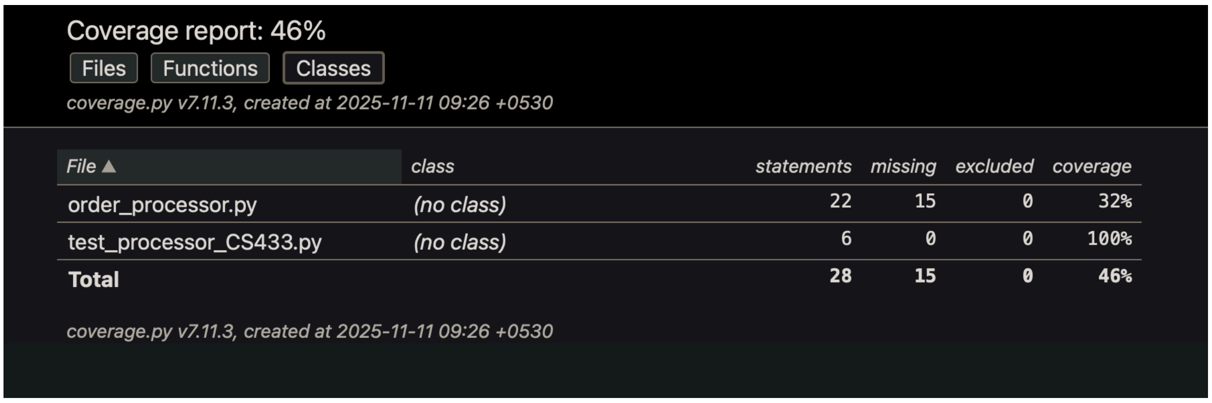
Files | Functions | Classes

coverage.py v7.11.3, created at 2025-11-11 09:26 +0530

File ▲	function	statements	missing	excluded	coverage
order_processor.py	calculate_discount	11	6	0	45%
order_processor.py	update_order_status	9	9	0	0%
order_processor.py	*(no function)*	2	0	0	100%
test_processor_CS433.py	test_regular_low_amount	1	0	0	100%
test_processor_CS433.py	test_premium_discount	1	0	0	100%
test_processor_CS433.py	*(no function)*	4	0	0	100%
Total		**28**	**15**	**0**	**46%**

coverage.py v7.11.3, created at 2025-11-11 09:26 +0530

Coverage report - Classes:



Coverage report: 46%

Files | Functions | Classes

coverage.py v7.11.3, created at 2025-11-11 09:26 +0530

File ▲	class	statements	missing	excluded	coverage
order_processor.py	*(no class)*	22	15	0	32%
test_processor_CS433.py	*(no class)*	6	0	0	100%
Total		**28**	**15**	**0**	**46%**

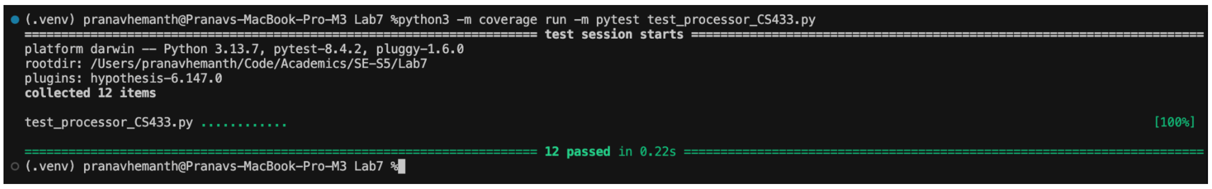
coverage.py v7.11.3, created at 2025-11-11 09:26 +0530

5. Complete the coverage test cases and rerun the test file:

Code:

Run the completed test file–

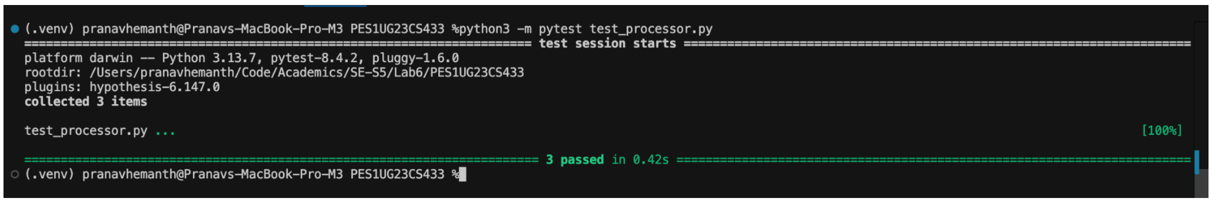
Command : python3 -m coverage run -m pytest test_processor_CS433.py



```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %python3 -m coverage run -m pytest test_processor_CS433.py
====================================================================== test session starts ======================================================================
platform darwin -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0
rootdir: /Users/pranavhemanth/Code/Academics/SE-S5/Lab7
plugins: hypothesis-6.147.0
collected 12 items

test_processor_CS433.py ............                                                                                                                      [100%]

====================================================================== 12 passed in 0.22s ======================================================================
○ (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %
```

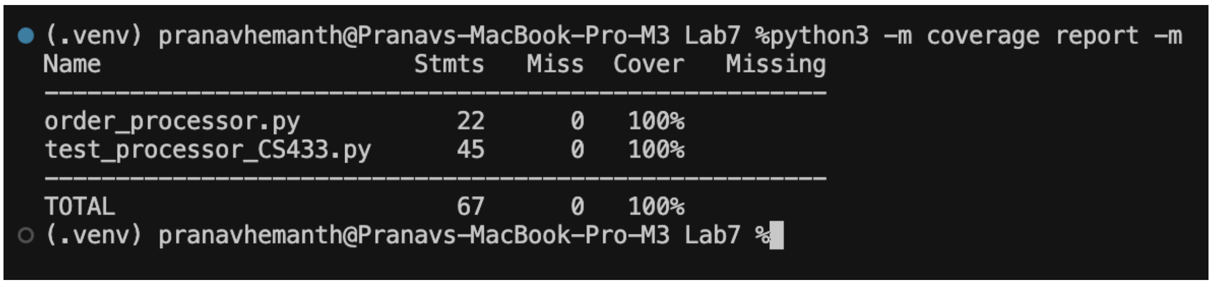
6. Re-Run the test_processor.py wrt to fixed processor and take the ss of the output:

Command: pytest test_processor.py
Or
python3 -m pytest test_processor.py (if pytest is installed but still showing not recognised)



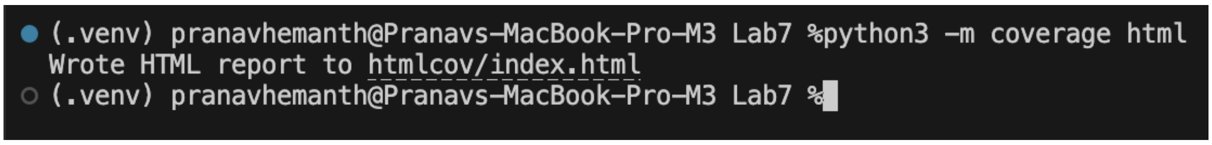
```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 PES1UG23CS433 %python3 -m pytest test_processor.py
====================================================================== test session starts ======================================================================
platform darwin -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0
rootdir: /Users/pranavhemanth/Code/Academics/SE-S5/Lab6/PES1UG23CS433
plugins: hypothesis-6.147.0
collected 3 items

test_processor.py ...                                                                                                                                    [100%]

======================================================================= 3 passed in 0.42s =======================================================================
○ (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 PES1UG23CS433 %
```

Command: python3 -m coverage report -m



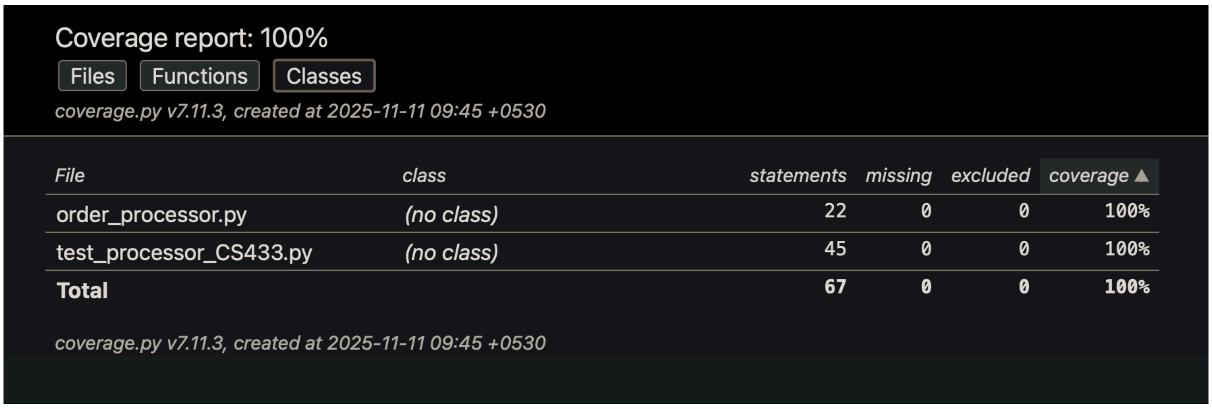
```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %python3 -m coverage report -m
  Name                           Stmts   Miss  Cover   Missing
  ------------------------------------------------------------
  order_processor.py                22      0   100%
  test_processor_CS433.py           45      0   100%
  ------------------------------------------------------------
  TOTAL                             67      0   100%
○ (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %
```

Command: python3 -m coverage html



```
● (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %python3 -m coverage html
  Wrote HTML report to htmlcov/index.html
○ (.venv) pranavhemanth@Pranavs-MacBook-Pro-M3 Lab7 %
```

Coverage report - Files:



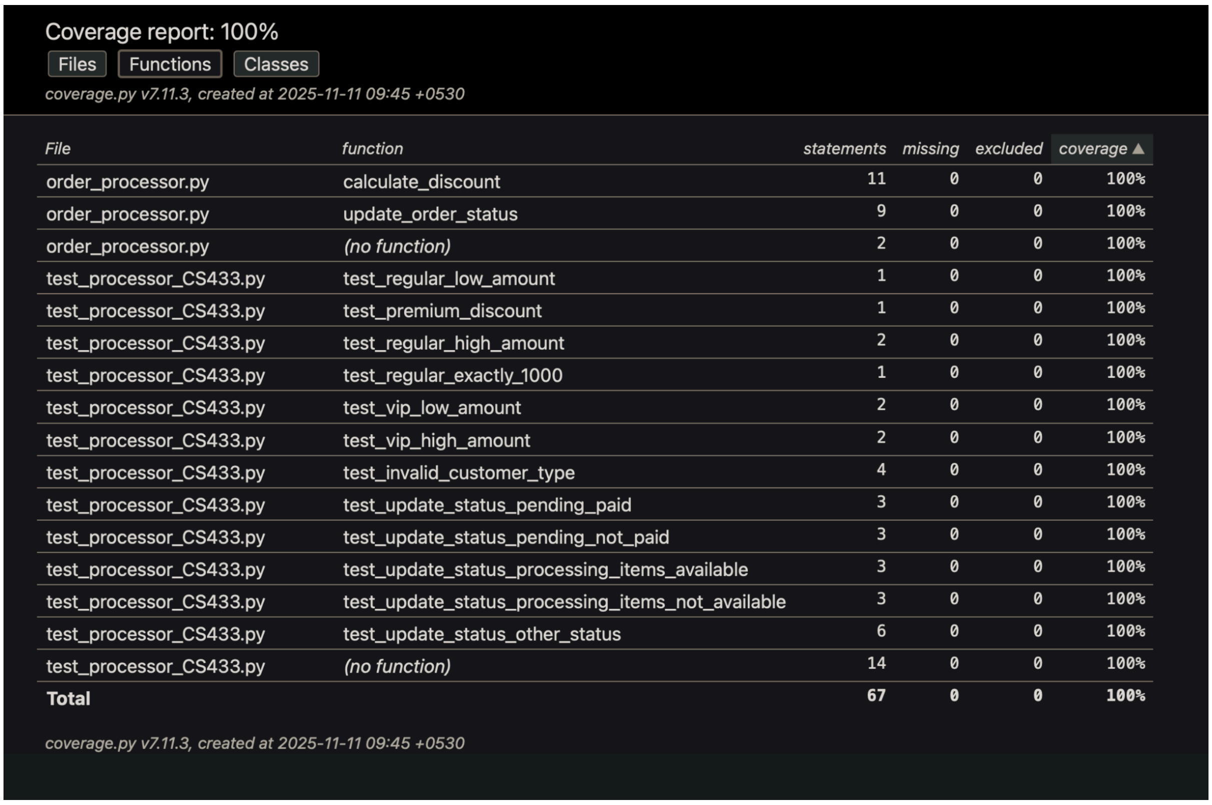
Coverage report: 100%

coverage.py v7.11.3, created at 2025-11-11 09:45 +0530

File	class	statements	missing	excluded	coverage ▲
order_processor.py	(no class)	22	0	0	100%
test_processor_CS433.py	(no class)	45	0	0	100%
Total		**67**	**0**	**0**	**100%**

coverage.py v7.11.3, created at 2025-11-11 09:45 +0530

Coverage report - Functions:



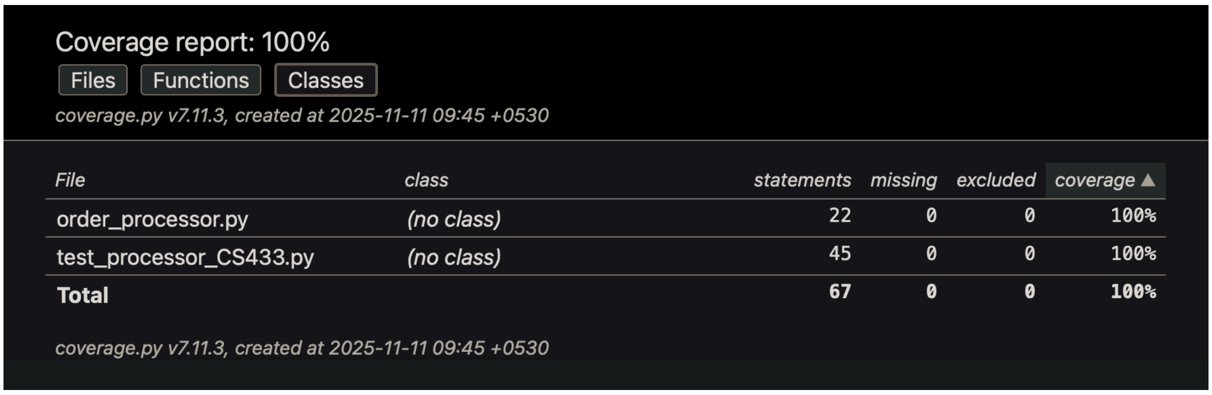
Coverage report: 100%

coverage.py v7.11.3, created at 2025-11-11 09:45 +0530

File	function	statements	missing	excluded	coverage ▲
order_processor.py	calculate_discount	11	0	0	100%
order_processor.py	update_order_status	9	0	0	100%
order_processor.py	(no function)	2	0	0	100%
test_processor_CS433.py	test_regular_low_amount	1	0	0	100%
test_processor_CS433.py	test_premium_discount	1	0	0	100%
test_processor_CS433.py	test_regular_high_amount	2	0	0	100%
test_processor_CS433.py	test_regular_exactly_1000	1	0	0	100%
test_processor_CS433.py	test_vip_low_amount	2	0	0	100%
test_processor_CS433.py	test_vip_high_amount	2	0	0	100%
test_processor_CS433.py	test_invalid_customer_type	4	0	0	100%
test_processor_CS433.py	test_update_status_pending_paid	3	0	0	100%
test_processor_CS433.py	test_update_status_pending_not_paid	3	0	0	100%
test_processor_CS433.py	test_update_status_processing_items_available	3	0	0	100%
test_processor_CS433.py	test_update_status_processing_items_not_available	3	0	0	100%
test_processor_CS433.py	test_update_status_other_status	6	0	0	100%
test_processor_CS433.py	(no function)	14	0	0	100%
Total		**67**	**0**	**0**	**100%**

coverage.py v7.11.3, created at 2025-11-11 09:45 +0530

Coverage report - Classes:



Coverage report: 100%

coverage.py v7.11.3, created at 2025-11-11 09:45 +0530

File	class	statements	missing	excluded	coverage ▲
order_processor.py	(no class)	22	0	0	100%
test_processor_CS433.py	(no class)	45	0	0	100%
Total		**67**	**0**	**0**	**100%**

coverage.py v7.11.3, created at 2025-11-11 09:45 +0530

7. Reflection:

1. If the logic in order_processor.py changes, what's your strategy to ensure tests and coverage stay updated?

My approach to ensuring tests and coverage remain comprehensive when modifying order_processor.py follows these steps:

1. Baseline Assessment: Before implementing any changes, I establish a baseline by running the existing test suite to verify all tests pass and reviewing the current coverage report to identify which code paths are already tested. This provides a clear picture of the system's current state.

2. Test-Driven Development Approach: I adopt a test-first methodology by writing test cases for new functionality or modified behavior before altering the production code. This approach ensures that:

- Requirements are clearly defined through tests
- New code is immediately testable
- Edge cases are considered upfront
- The implementation has clear success criteria

3. Validation and Coverage Analysis: After implementing changes, I execute the full test suite to identify any regressions or failures. I then analyze the coverage report to ensure:

- Overall coverage remains at or above 90%
- All new code paths are tested
- No previously covered code has become untested
- Critical branches and error handling are adequately tested

If coverage drops, I investigate the uncovered lines and add targeted tests to restore comprehensive coverage.

4. Automated Quality Gates: To prevent coverage regression and maintain code quality, I implement continuous integration (CI/CD) pipelines using tools like GitHub Actions. These pipelines:

- Automatically run the complete test suite on every commit
- Generate and analyze coverage reports
- Block pull request merges if tests fail or coverage falls below the threshold
- Provide immediate feedback to developers before code reaches the main branch

This systematic approach ensures that code modifications are thoroughly tested, coverage standards are maintained, and quality issues are caught early in the development cycle.

2. What are the trade-offs between writing more tests for coverage vs writing fewer high- quality tests?

1 High Coverage Approach (Writing More Tests)

Advantages:

- Achieves higher test coverage metrics (often 90-100%)
- Identifies more edge cases and boundary conditions
- Helps uncover unforeseen bugs and behavior deviations
- Offers detailed documentation of system behavior through test outcomes

Disadvantages:

- Significantly increases the time required for authoring and maintenance
- Leads to slower test execution and longer feedback cycles
- A single code change can necessitate updates across numerous test cases
- Can introduce redundancy, reducing the overall value of additional tests

2. Focused High-Quality Testing (Fewer, More Strategic Tests)

Advantages:

- Faster to implement, execute, and maintain
- Easier to adapt when code or requirements evolve
- Directs effort toward mission critical components and high risk logic
- Produces clear, intentional, and easy-to-understand test cases

Disadvantages:

- May miss rare or less obvious edge cases
- Typically results in lower test coverage percentages
- Increases the likelihood of undetected issues in untested areas
- Provides less comprehensive behavioral assurance

3. Recommended Balanced Approach

An optimal testing strategy balances breadth and depth rather than leaning entirely toward either extreme.

- Aim for 80–95% coverage: a practical threshold that ensures broad validation without over-investment
- Adopt risk-based testing: prioritize testing of critical business logic, complex algorithms, and frequently modified components
- Ensure essential path coverage: include tests for major control flow branches, exceptions, and error-handling paths
- Eliminate redundancy: test representative data points and critical boundaries instead of every possible value
- Emphasize meaningful validation: each test should serve a clear purpose and enhance confidence in system stability

Conclusion:
High coverage should be an outcome of effective testing practices, not the primary objective. The goal is to maintain a well-balanced, sustainable test suite that thoroughly validates key behaviors, minimizes redundancy, and provides lasting confidence in the reliability and correctness of the system.