Lab 7: Code and Branch Coverage Analysis

Objective

- 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**.

Duration:

90 minutes

Software Requirements

- Python 3.x
- coverage (install via pip)
- Any code editor (VS Code / PyCharm / Sublime / etc.)
- pytest or unittest (either is fine)

💡 Coverage can be used standalone or with test frameworks.

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.

NOTE: Students may be randomly called for a presentation after completing the lab. Please be prepared to discuss your code, bugs you found, and how you fixed them.

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:

PES1UG23CSXXX/

```
├── order_processor.py        # Main code to test
├── test_processor_CSXXX.py   # Incomplete starter test file, replace XXX with SRN
└── Instructions Manual
```

RENAME FOLDER and test_processor file TO YOUR SRNS BEFORE RUNNING.

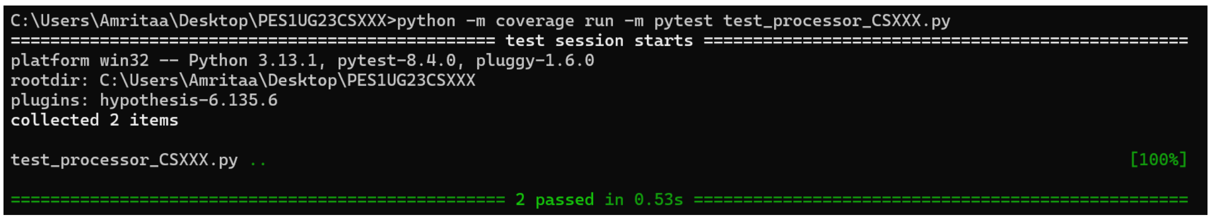
Deliverables:

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

Steps:

1. **Install dependencies:**
   pip install coverage

2. **Run the original test file given to you**
   python -m coverage run -m pytest test_processor_CSXXX.py

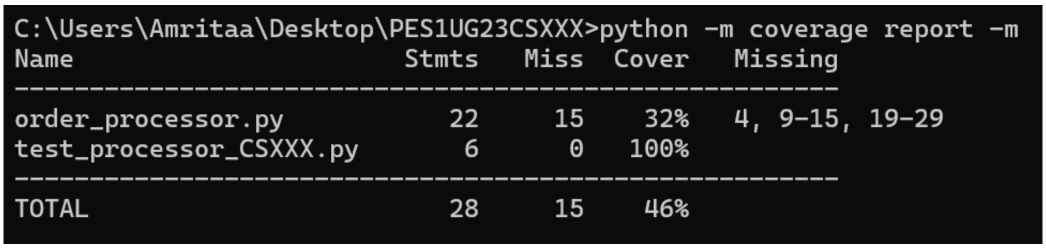


```
C:\Users\Amritaa\Desktop\PES1UG23CSXXX>python -m coverage run -m pytest test_processor_CSXXX.py
================================================= test session starts =================================================
platform win32 -- Python 3.13.1, pytest-8.4.0, pluggy-1.6.0
rootdir: C:\Users\Amritaa\Desktop\PES1UG23CSXXX
plugins: hypothesis-6.135.6
collected 2 items

test_processor_CSXXX.py ..                                                                                      [100%]

================================================== 2 passed in 0.53s ==================================================
```

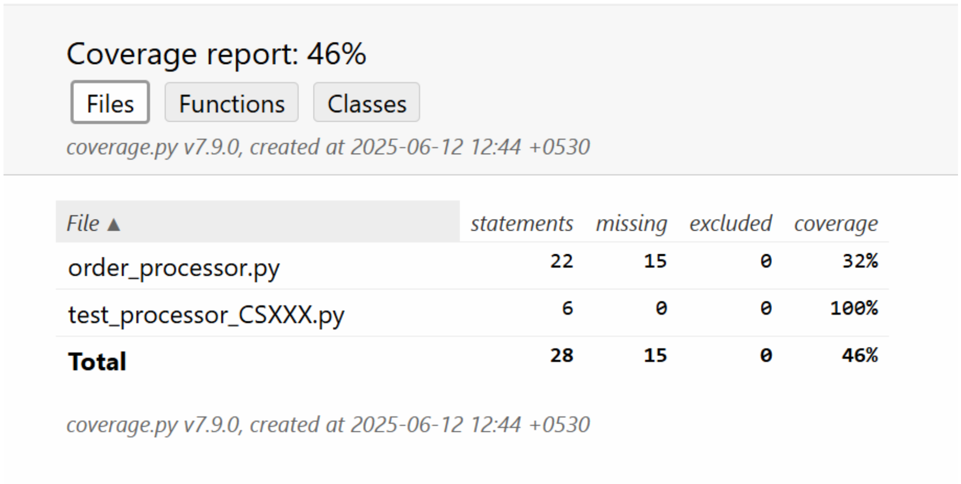
3. python -m coverage report -m



```
C:\Users\Amritaa\Desktop\PES1UG23CSXXX>python -m coverage report -m
Name                          Stmts   Miss  Cover   Missing
-----------------------------------------------------------
order_processor.py               22     15    32%   4, 9-15, 19-29
test_processor_CSXXX.py           6      0   100%
-----------------------------------------------------------
TOTAL                            28     15    46%
```

4. python -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.

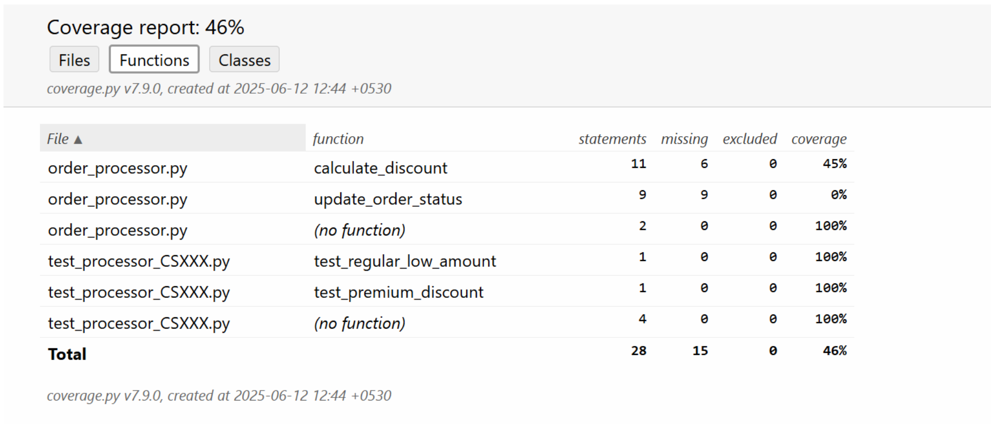


Coverage report: 46%

coverage.py v7.9.0, created at 2025-06-12 12:44 +0530

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

coverage.py v7.9.0, created at 2025-06-12 12:44 +0530

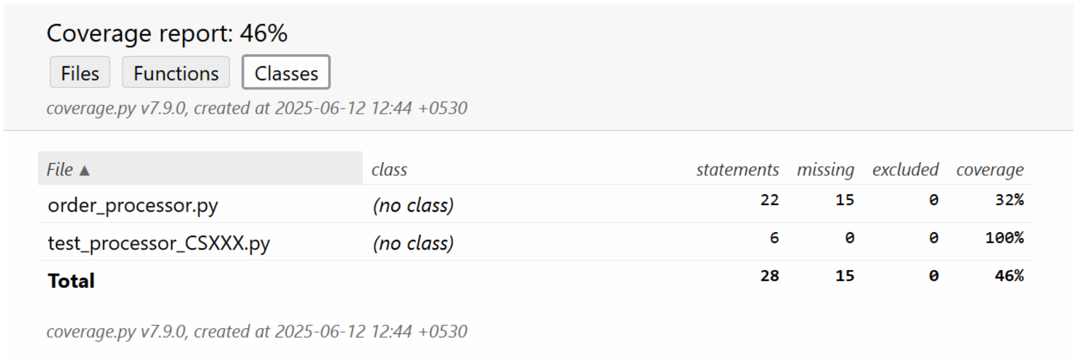


Coverage report: 46%

coverage.py v7.9.0, created at 2025-06-12 12:44 +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_CSXXX.py	test_regular_low_amount	1	0	0	100%
test_processor_CSXXX.py	test_premium_discount	1	0	0	100%
test_processor_CSXXX.py	*(no function)*	4	0	0	100%
Total		28	15	0	46%

coverage.py v7.9.0, created at 2025-06-12 12:44 +0530



Coverage report: 46%

coverage.py v7.9.0, created at 2025-06-12 12:44 +0530

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

coverage.py v7.9.0, created at 2025-06-12 12:44 +0530

Note (for some Python 3 users):
If your coverage report includes too many files or shows files outside your lab folder, it means coverage is tracking more than just your code. You can fix this by telling it exactly which files to check using the --source option.

Use this command instead of the regular one:

python -m coverage run --source=order_processor.py,test_processor_CSXXX.py -m pytest test_processor_CSXXX.py

This keeps the coverage report focused only on the files you're working with.

5. Write more test cases in the test file to achieve atleast 90% coverage. Re-run steps 2-4 and provide similar screenshots as above for the results. Give screenshots ONLY for results after achieving atleast 90% coverage.
   (Please make sure to change your SRNs wherever needed)

Reflection:

1. If the logic in order_processor.py changes, what's your strategy to ensure tests and coverage stay updated?
2. What are the trade-offs between writing more tests for coverage vs writing fewer high-quality tests?