

SE Lab-7

Name: Anirudh Sudhir

SRN: PES1UG23CS917

2

```
sem-5/se/lab-7-coverage main*
● venv > python -m coverage run -m pytest test_processor_CS917.py
=====
platform darwin -- Python 3.14.0, pytest-9.0.0, pluggy-1.6.0
rootdir: /Users/anirudh/Documents/code/sem-5/se/lab-7-coverage
collected 2 items

test_processor_CS917.py .. [100%]

=====
2 passed in 0.01s =====
```

3

```
sem-5/se/lab-7-coverage main*
● venv > python -m coverage report -m
Name           Stmts   Miss  Cover  Missing
-----[{"file": "order_processor.py", "stmts": 22, "miss": 15, "cover": "32%", "missing": "4, 9-15, 19-29"}, {"file": "test_processor_CS917.py", "stmts": 6, "miss": 0, "cover": "100%"}, {"file": "TOTAL", "stmts": 28, "miss": 15, "cover": "46%"}]
```

4

Coverage report: 46%

[Files](#) [Functions](#) [Classes](#)

coverage.py v7.11.3, created at 2025-11-10 09:24 +0530

File ▲	statements	missing	excluded	coverage
order_processor.py	22	15	0	32%
test_processor_CS917.py	6	0	0	100%
Total	28	15	0	46%

coverage.py v7.11.3, created at 2025-11-10 09:24 +0530

5

```
sem-5/se/lab-7-coverage main*
● venv > python -m coverage run -m pytest test_processor_CS917.py
=====
platform darwin -- Python 3.14.0, pytest-9.0.0, pluggy-1.6.0
rootdir: /Users/anirudh/Documents/code/sem-5/se/lab-7-coverage
collected 10 items

test_processor_CS917.py ..... [100%]

=====
10 passed in 0.01s =====
```

```
sem-5/se/lab-7-coverage main*
● venv > python -m coverage report -m
Name           Stmts  Miss  Cover  Missing
-----
order_processor.py      22     0   100%
test_processor_CS917.py 31     0   100%
-----
TOTAL                  53     0   100%
```

Coverage report: 100%

[Files](#) [Functions](#) [Classes](#)

coverage.py v7.11.3, created at 2025-11-10 09:26 +0530

File ▲	statements	missing	excluded	coverage
order_processor.py	22	0	0	100%
test_processor_CS917.py	31	0	0	100%
Total	53	0	0	100%

coverage.py v7.11.3, created at 2025-11-10 09:26 +0530

Reflection

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

Run coverage reports after every code change to identify untested paths. Update existing tests when function logic changes and add new tests for any new branches or conditions. Use CI/CD pipelines to automatically run tests and enforce minimum coverage thresholds (e.g., 90%). Review coverage reports during code reviews to catch gaps before merging.

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

Writing more tests increases coverage and catches edge cases but requires higher maintenance effort and longer test execution time. Many tests can become brittle and break frequently with code changes. Fewer high-quality tests focus on critical business logic, are easier to maintain, and run faster, but may miss edge cases and result in lower coverage. The best approach balances both prioritising comprehensive testing of critical paths while accepting lower coverage for trivial code.