Lab 7: Code and Branch Coverage Analysis

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

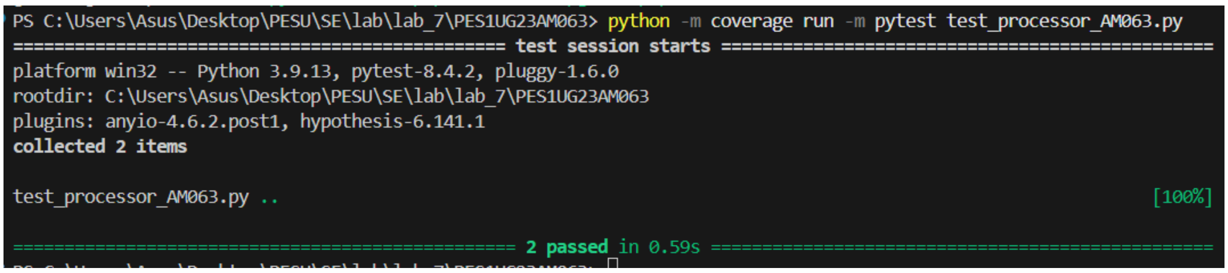
Why It Matters:

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

Steps:

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

2. **Run the original test file given to you** *python -m coverage run -m pytest test_processor_AM063.py*

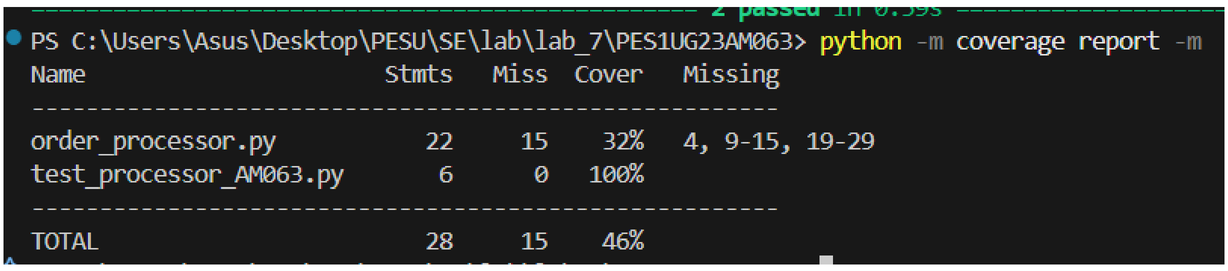


```
PS C:\Users\Asus\Desktop\PESU\SE\lab\lab_7\PES1UG23AM063> python -m coverage run -m pytest test_processor_AM063.py
============================================== test session starts ==============================================
platform win32 -- Python 3.9.13, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\Asus\Desktop\PESU\SE\lab\lab_7\PES1UG23AM063
plugins: anyio-4.6.2.post1, hypothesis-6.141.1
collected 2 items

test_processor_AM063.py ..                                                                                 [100%]

=============================================== 2 passed in 0.59s ===============================================
```

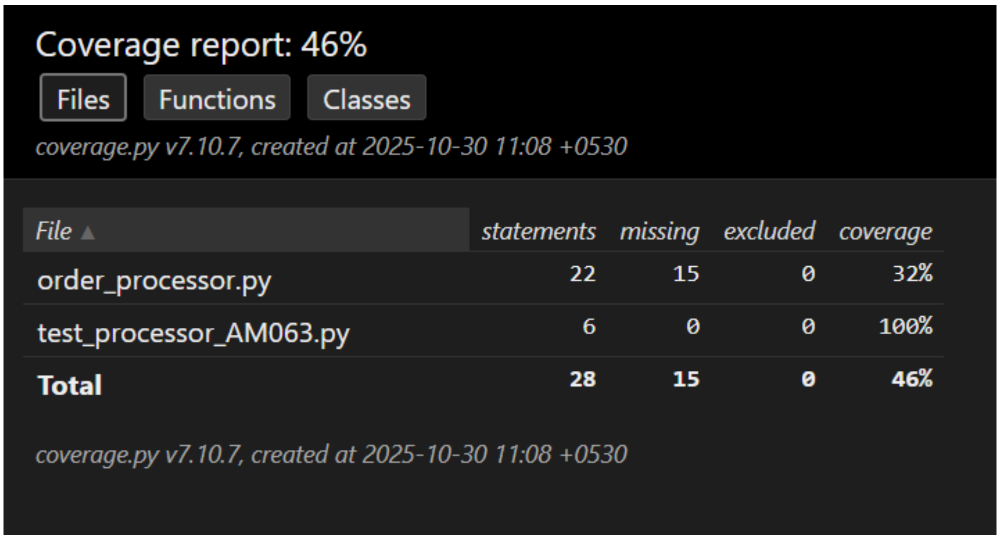
3. python -m coverage report -m



```
PS C:\Users\Asus\Desktop\PESU\SE\lab\lab_7\PES1UG23AM063> python -m coverage report -m
Name                       Stmts   Miss  Cover   Missing
--------------------------------------------------------
order_processor.py            22     15    32%   4, 9-15, 19-29
test_processor_AM063.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.

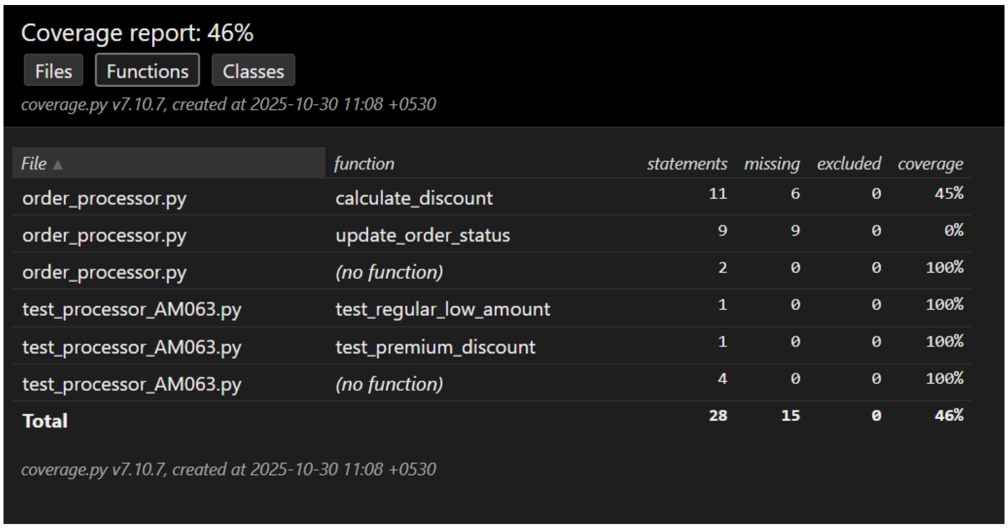


Coverage report: 46%

coverage.py v7.10.7, created at 2025-10-30 11:08 +0530

File ▲	statements	missing	excluded	coverage
order_processor.py	22	15	0	32%
test_processor_AM063.py	6	0	0	100%
Total	**28**	**15**	**0**	**46%**

coverage.py v7.10.7, created at 2025-10-30 11:08 +0530

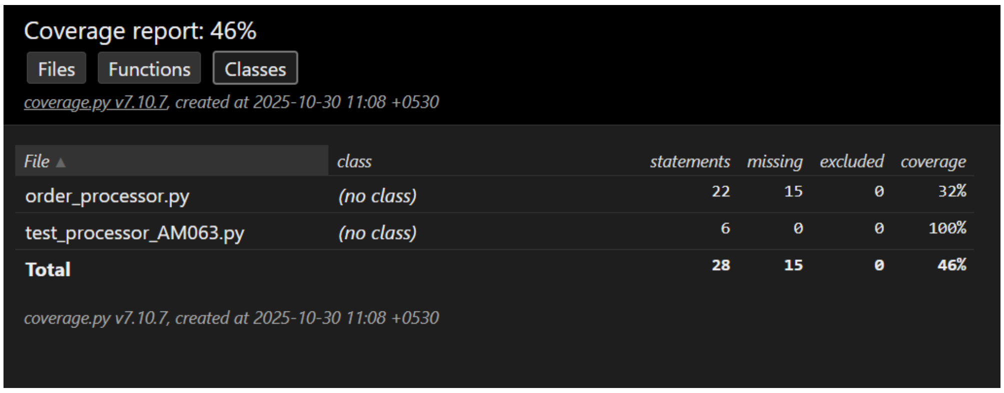


Coverage report: 46%

coverage.py v7.10.7, created at 2025-10-30 11:08 +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_AM063.py	test_regular_low_amount	1	0	0	100%
test_processor_AM063.py	test_premium_discount	1	0	0	100%
test_processor_AM063.py	*(no function)*	4	0	0	100%
Total		**28**	**15**	**0**	**46%**

coverage.py v7.10.7, created at 2025-10-30 11:08 +0530



Coverage report: 46%

coverage.py v7.10.7, created at 2025-10-30 11:08 +0530

File ▲	class	statements	missing	excluded	coverage
order_processor.py	*(no class)*	22	15	0	32%
test_processor_AM063.py	*(no class)*	6	0	0	100%
Total		**28**	**15**	**0**	**46%**

coverage.py v7.10.7, created at 2025-10-30 11:08 +0530

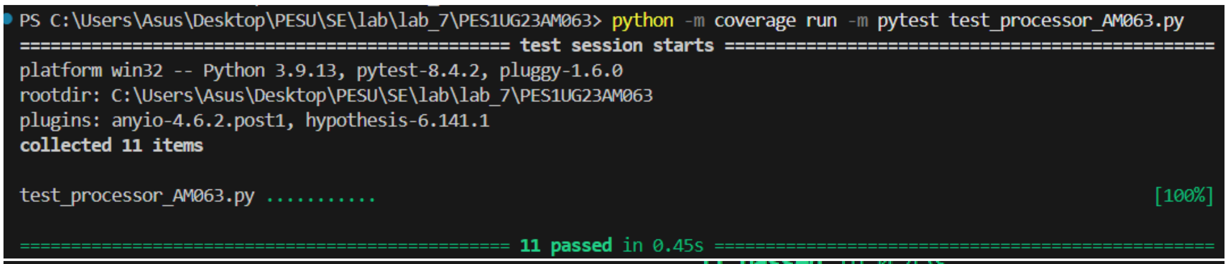
Note (for some Python 3 users):
If your coverage report includes too many files or shows files outside your lab folder, it means coverage is tracking more than just your code. You can fix this by telling it exactly which files to check using the --source option.

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.

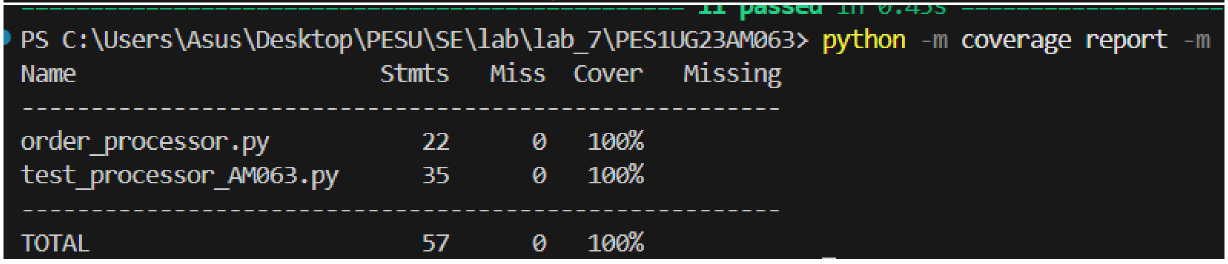
4. Write more test cases in the test file to achieve atleast 90% coverage. Re-run steps 2-4 and provide similar screenshots as above for the results. Give screenshots ONLY for results after achieving atleast 90% coverage.



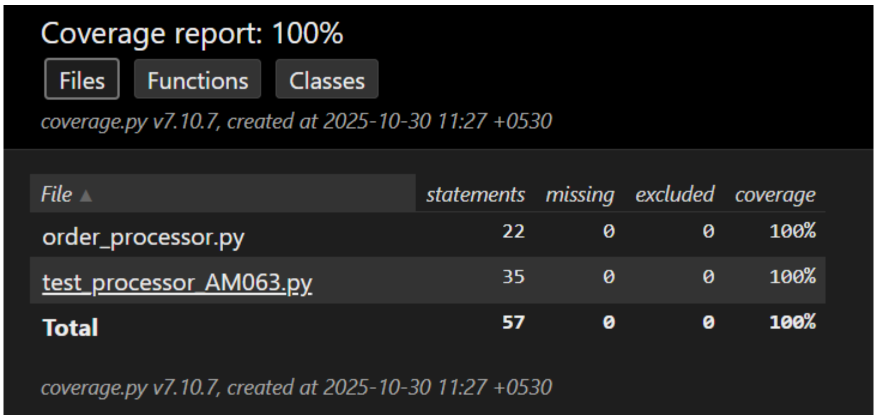
```
PS C:\Users\Asus\Desktop\PESU\SE\lab\lab_7\PES1UG23AM063> python -m coverage run -m pytest test_processor_AM063.py
================================================ test session starts ================================================
platform win32 -- Python 3.9.13, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\Asus\Desktop\PESU\SE\lab\lab_7\PES1UG23AM063
plugins: anyio-4.6.2.post1, hypothesis-6.141.1
collected 11 items

test_processor_AM063.py ...........                                                                           [100%]

================================================ 11 passed in 0.45s ================================================
```



```
PS C:\Users\Asus\Desktop\PESU\SE\lab\lab_7\PES1UG23AM063> python -m coverage report -m
Name                        Stmts   Miss  Cover   Missing
-------------------------------------------------------
order_processor.py             22      0   100%
test_processor_AM063.py        35      0   100%
-------------------------------------------------------
TOTAL                          57      0   100%
```

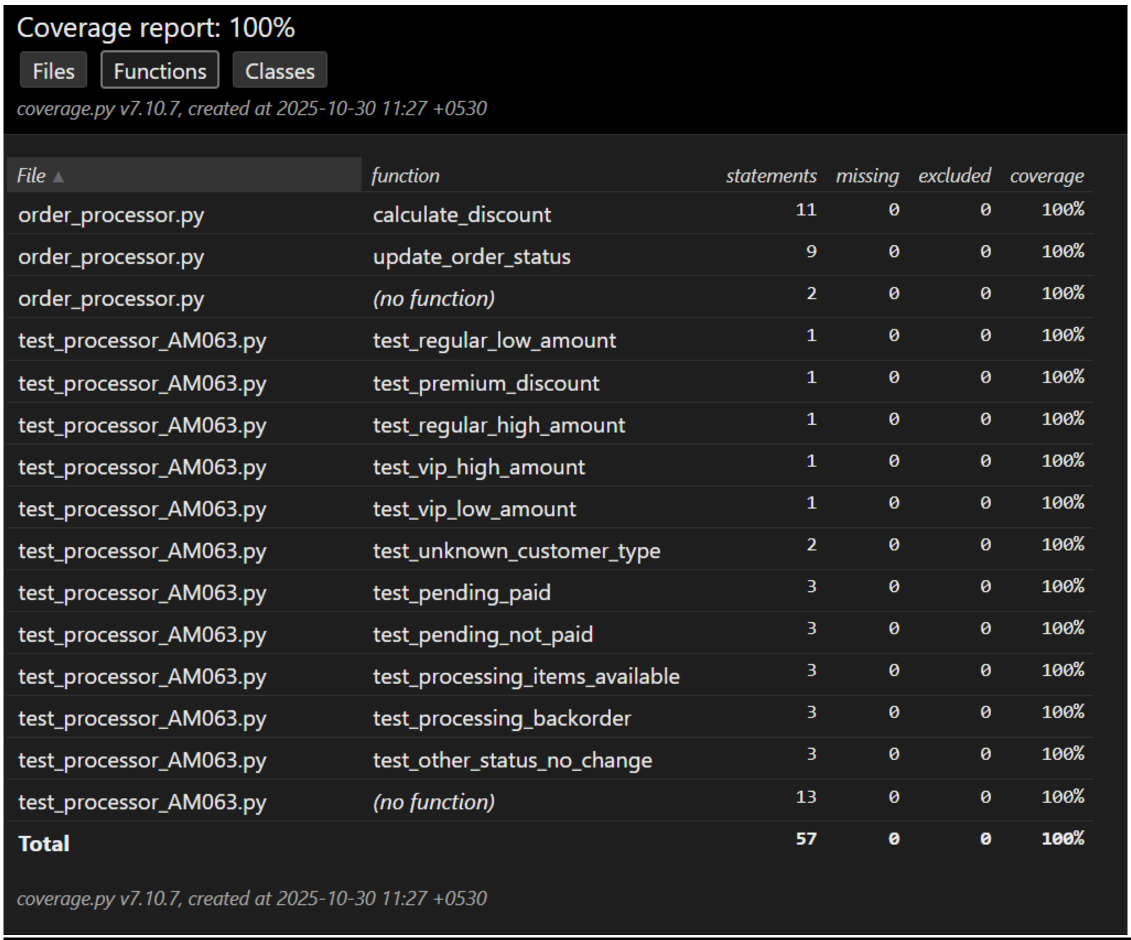


Coverage report: 100%

coverage.py v7.10.7, created at 2025-10-30 11:27 +0530

File ▲	statements	missing	excluded	coverage
order_processor.py	22	0	0	100%
test_processor_AM063.py	35	0	0	100%
Total	**57**	**0**	**0**	**100%**

coverage.py v7.10.7, created at 2025-10-30 11:27 +0530

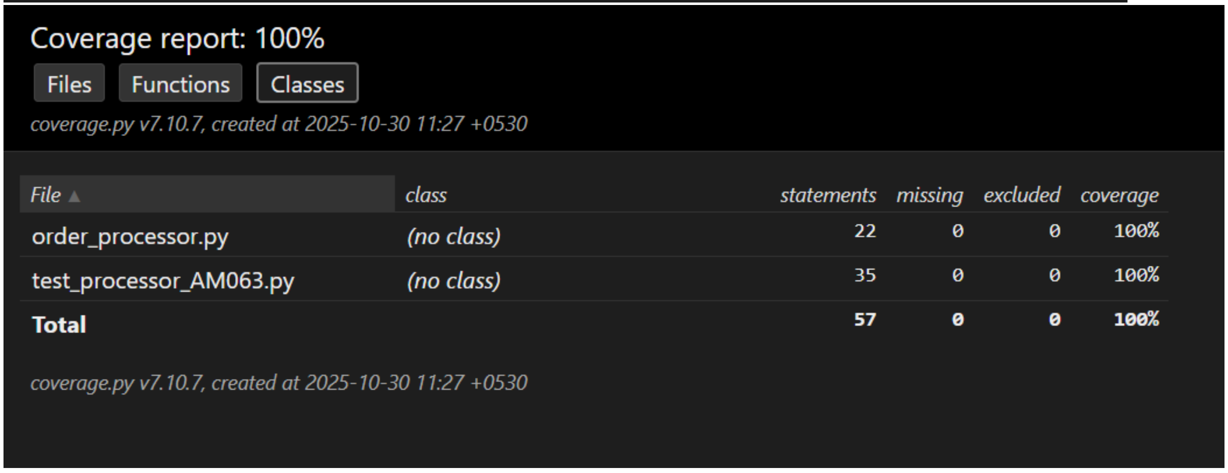

Coverage report: 100%

Files Functions Classes

coverage.py v7.10.7, created at 2025-10-30 11:27 +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_AM063.py	test_regular_low_amount	1	0	0	100%
test_processor_AM063.py	test_premium_discount	1	0	0	100%
test_processor_AM063.py	test_regular_high_amount	1	0	0	100%
test_processor_AM063.py	test_vip_high_amount	1	0	0	100%
test_processor_AM063.py	test_vip_low_amount	1	0	0	100%
test_processor_AM063.py	test_unknown_customer_type	2	0	0	100%
test_processor_AM063.py	test_pending_paid	3	0	0	100%
test_processor_AM063.py	test_pending_not_paid	3	0	0	100%
test_processor_AM063.py	test_processing_items_available	3	0	0	100%
test_processor_AM063.py	test_processing_backorder	3	0	0	100%
test_processor_AM063.py	test_other_status_no_change	3	0	0	100%
test_processor_AM063.py	*(no function)*	13	0	0	100%
Total		**57**	**0**	**0**	**100%**

coverage.py v7.10.7, created at 2025-10-30 11:27 +0530


Coverage report: 100%

Files Functions Classes

coverage.py v7.10.7, created at 2025-10-30 11:27 +0530

File ▲	class	statements	missing	excluded	coverage
order_processor.py	*(no class)*	22	0	0	100%
test_processor_AM063.py	*(no class)*	35	0	0	100%
Total		**57**	**0**	**0**	**100%**

coverage.py v7.10.7, created at 2025-10-30 11:27 +0530

Reflection:

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

   Rerun all existing tests. If a test fails, fix it to match the new correct logic. If the code has new parts (like new if/else conditions), write new tests specifically for those new parts until coverage goes back up.

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

   Writing **more tests for higher coverage** can help catch many small or simple bugs and makes the codebase look well-tested (for example, showing 100% coverage). However, this approach can take a lot of time to write, run, and maintain, and many of the tests may end up being repetitive or not very useful.

On the other hand, writing **fewer but high-quality tests** focuses effort on the most important or complex parts of the code. These tests are easier to maintain and often give more meaningful feedback. The downside is that overall test coverage might look low, and some minor issues in untested areas could go unnoticed.