TU856/3
SOFTWARE TESTING

LAB 5

10/03/2025

PAULINA CZARNOTA C21365726

White Box Testing

1. Running Unit Tests

To execute the unit tests for the grader module, ensure that the project directory is set correctly. You can run the tests using the following command in your terminal:

```
PS C:\Users\35389\Desktop\TU856 Modules\YEAR 3\Year 3 - Semester 2\Software Testing - Robert O'Dea\Labs\Week 7\Lab6_WhiteBoxTesting> python -m unittest test_grader.py
.........
----------------------------------------------------------------------
Ran 9 tests in 0.002s

OK
```

This will run the unit tests in the tests directory and display the results in the terminal. If everything is set up correctly, the output will show that the tests passed or failed.

2. Checking Code Coverage

To measure the code coverage for your tests, use the following command:

```
PS C:\Users\35389\Desktop\TU856 Modules\YEAR 3\Year 3 - Semester 2\Software Testing - Robert O'Dea\Labs\Week 7\Lab6_WhiteBoxTesting> coverage run -m unittest test_grader.py
.........
----------------------------------------------------------------------
Ran 9 tests in 0.003s

OK
```

This command will run the tests while measuring the code coverage, and the results will be displayed in the terminal as well as in the coverage report.

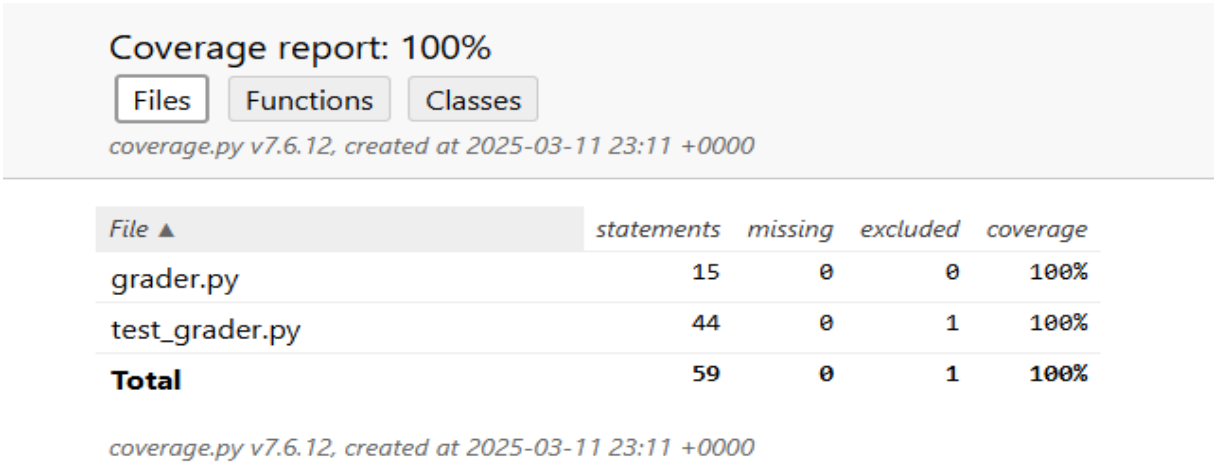
```
PS C:\Users\35389\Desktop\TU856 Modules\YEAR 3\Year 3 - Semester 2\Software Testing - Robert O'Dea\Labs\Week 7\Lab6_WhiteBoxTesting> coverage report -m
Name            Stmts   Miss  Cover   Missing
---------------------------------------------
grader.py          15      0   100%
test_grader.py     44      0   100%
---------------------------------------------
TOTAL              59      0   100%
```

3. Generating HTML Coverage Report

To generate a detailed HTML coverage report that shows which parts of the code have been tested, use the following command:

```
PS C:\Users\35389\Desktop\TU856 Modules\YEAR 3\Year 3 - Semester 2\Software Testing - Robert O'Dea\Labs\Week 7\Lab6_WhiteBoxTesting> coverage html
Wrote HTML report to htmlcov\index.html
```

This creates a report in the htmlcov/index.html file, which you can open in your browser to view the coverage details.



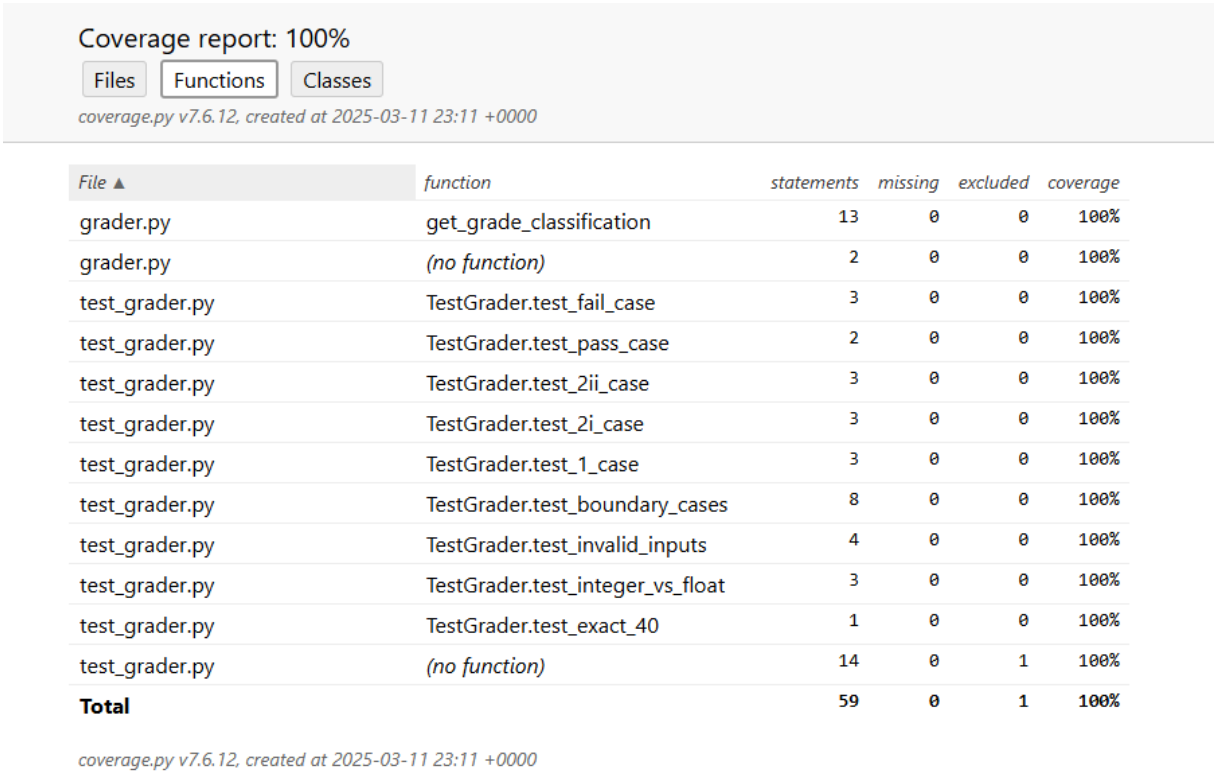
Coverage report: 100%

Files | Functions | Classes

coverage.py v7.6.12, created at 2025-03-11 23:11 +0000

File ▲	statements	missing	excluded	coverage
grader.py	15	0	0	100%
test_grader.py	44	0	1	100%
Total	**59**	**0**	**1**	**100%**

coverage.py v7.6.12, created at 2025-03-11 23:11 +0000



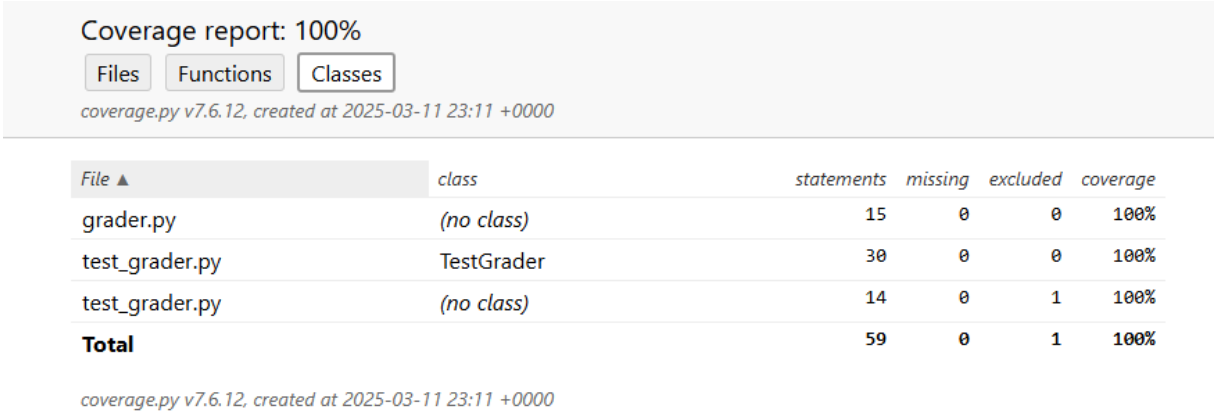
Coverage report: 100%

Files | Functions | Classes

coverage.py v7.6.12, created at 2025-03-11 23:11 +0000

File ▲	function	statements	missing	excluded	coverage
grader.py	get_grade_classification	13	0	0	100%
grader.py	*(no function)*	2	0	0	100%
test_grader.py	TestGrader.test_fail_case	3	0	0	100%
test_grader.py	TestGrader.test_pass_case	2	0	0	100%
test_grader.py	TestGrader.test_2ii_case	3	0	0	100%
test_grader.py	TestGrader.test_2i_case	3	0	0	100%
test_grader.py	TestGrader.test_1_case	3	0	0	100%
test_grader.py	TestGrader.test_boundary_cases	8	0	0	100%
test_grader.py	TestGrader.test_invalid_inputs	4	0	0	100%
test_grader.py	TestGrader.test_integer_vs_float	3	0	0	100%
test_grader.py	TestGrader.test_exact_40	1	0	0	100%
test_grader.py	*(no function)*	14	0	1	100%
Total		**59**	**0**	**1**	**100%**

coverage.py v7.6.12, created at 2025-03-11 23:11 +0000


Coverage report: 100%

Files Functions Classes

coverage.py v7.6.12, created at 2025-03-11 23:11 +0000

File ▲	class	statements	missing	excluded	coverage
grader.py	*(no class)*	15	0	0	100%
test_grader.py	TestGrader	30	0	0	100%
test_grader.py	*(no class)*	14	0	1	100%
Total		59	0	1	100%

coverage.py v7.6.12, created at 2025-03-11 23:11 +0000

4. Statement Coverage vs Decision Coverage

Statement Coverage:

While running the tests, you might see that the initial coverage is 66%. This means that not all the lines of code in the grader.py file were executed during the tests.

- **Action:** Add another test case that ensures full coverage, bringing it up to 100%.

Decision Coverage:

Decision coverage ensures that every possible branch in the if statements is covered. Initially, your tests might not account for all conditions, such as exactly 40.0, which may cause some logic to fail.

- **Action:** Update the code to handle edge cases like 40.0 and re-run the tests. Adjust PyCharm's settings to enable decision coverage rather than just statement coverage.

5. Closing Out the Tests

Testing for Invalid Inputs:

Your function should raise an error if a non-numeric input is provided, such as a string. Specifically, it should raise a ValueError if the input is invalid.

- **Tests to Add:**
  - Pass an integer (40 without a decimal point).
  - Pass a string ("forty") and ensure the function raises a ValueError.
- **Action:** Write the new tests, then update the grader.py function to handle these cases. Once the updates are made, rerun the tests to ensure they pass.

6. Code Coverage Summary

Using PyCharm's coverage tool helps track which parts of the code have been tested. Achieving 100% coverage is a good practice, but it doesn't necessarily mean your testing is complete. Decision coverage helps to ensure that all branches of logic are tested, and boundary values are properly covered.

The get_grade_classification function should handle the following grade classifications:

- **Fail:** 0–39
- **Pass:** 40–49
- **2 ii:** 50–59
- **2 i:** 60–69
- **1:** 70–100

7. Test Coverage Practice Questions

1. Equivalence Partitioning

The functionality of get_grade_classification was extended, and the new grade ranges are:

- **0–39** = Fail
- **40–49** = Pass
- **50–59** = 2 ii
- **60–69** = 2 i
- **70–100** = 1

Tests have been implemented for the following inputs:

- 20, 39, 55, 60, 70

Partition Coverage Calculation:

- **Total Partitions**: 6 (Fail, Pass, 2 ii, 2 i, 1, Invalid input)
- **Covered Partitions**: 5 (Fail, Pass, 2 ii, 2 i, 1)
- **Partition Coverage**: 5/6 x 100 = 83.3%

2. Decision Table Coverage

The ice cream app has the following options:

- Sprinkles (with/without)
- Lime Syrup (with/without)
- Strawberry Syrup (with/without)
- Flake (with/without)

The app developers have created one test using the following setup:

- Sprinkles, Lime Syrup, Strawberry Syrup, and a Flake.

Total Possible Combinations: 16 (2^4)

Covered Combinations: 1 (The one test created)

Decision Table Coverage: 1/16 x 100 = 6.25%

3. State Transition Coverage

The password authentication system includes the following states:

- Start
- Incorrect password (3 attempts)
- Correct password
- Access granted
- Account locked

Tests:

1. Open interface, enter an incorrect password three times.
2. Open interface, enter the correct password.

State Coverage Calculation:

- **Total states**: 5
- **Covered states**: 4
- **State Coverage**: 4/5 x 100 = 80%

Transition Coverage Calculation:

- **Total transitions**: 5
- **Covered transitions**: 3
- **Transition Coverage**: 3/5 x 100 = 60%

Conclusion

In this lab, we used statement coverage and decision coverage to ensure that all lines of code and logic branches were tested. Achieving 100% coverage is a good target, but it's important to use decision coverage and boundary testing to make sure the tests are complete. These tools, alongside well-structured tests, help ensure that edge cases and complex logic paths are covered effectively.