Unit Testing Report

Please provide your GitHub repository link.

GitHub Repository URL: https://github.com/EJDARE/Milestone2_Group15.git

The testing report should focus solely on testing all the self-defined functions related to the five required features. There is no need to test the GUI components. Therefore, it is essential to decouple your code and separate the logic from the GUI-related code.

1. Test Summary

list all tested functions related to the five required features and the corresponding test functions designed to test those functions, for example:

Tested Functions	Test Functions
`add(x1,x2)`	`test_add_valid()` `test_add_invalid`
`divide(x1,x2)`	`test_divide_valid()` `test_divide_invalid`
`multiply(a,b)`	`test_multiply_valid()` `test_multiply_invalid()`
`subtract(a,b)`	`test_subtract_valid()'   test_subtract_invalid()`
`process_data(data)`	`test_process_data_valid()'   test_process_data_invalid()`

2. Test Case Details

Test Case 1: Division Operation

- **Test Function/Module**
  - `test_divide_valid()`
  - `test_divide_invalid()`
- **Tested Function/Module**
  - `divide(a, b)`
- **Description**

    - The divide function performs division between two numbers. It takes two arguments (a and b), where a is divided by b. The function should handle exceptions, particularly division by zero.
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
`divide(10, 2)`	`5`
`divide(10, -2)`	`-5`

- **1) Code for the Test Function**

```
def test_divide_valid():
    assert divide(10, 2) == 5
    assert divide(10, -2) == -5
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
`divide(10, 0)`	`Handle Exception`

- **2) Code for the Test Function**

```
def test_divide_invalid():
    with pytest.raises(ValueError) as exc_info:
        divide(10, 0)
    assert exc_info.type is ValueError
```

Test Case 2: Addition Operation

- **Test Function/Module**
    - `test_add_valid()`
    - `test_add_invalid()`
- **Tested Function/Module**
    - `add(x1, x2)`
- **Description**
    - The add function performs the addition of two numbers. It accepts two parameters x1 and x2 and returns their sum.
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
`add(3, 2)`	`5`
`add(-1, 1)`	`-5`

- **1) Code for the Test Function**

```
def test_add_valid():
    assert add(3, 2) == 5
    assert add(-1, 1) == 0
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
`add(3, "two")`	`Handle TypeError`

- **2) Code for the Test Function**

```
def test_add_invalid():
    with pytest.raises(TypeError):
        add(3, "two")
```

Test Case 3: Multiplication Operation

- **Test Function/Module**
  - `test_multiply_valid()`
  - `test_multiply_invalid()`
- **Tested Function/Module**
  - `multiply(a, b)`
- **Description**
  - A brief description of the tested function's usage, including its purpose, input, and output.
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
`multiply(2, 3)`	`6`
`multiply(-2, 3)`	`-6`

- **1) Code for the Test Function**

```
def test_multiply_valid():
    assert multiply(2, 3) == 6
    assert multiply(-2, 3) == -6
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
`multiply(2, None)`	`Handle TypeError`

- **2) Code for the Test Function**

```
def test_multiply_invalid():
    with pytest.raises(TypeError):
        multiply(2, None)
```

Test Case 4: Subtraction Operation

- **Test Function/Module**
  - `test_subtract_valid()`
  - `test_subtract_invalid()`
- **Tested Function/Module**
  - `subtract(a, b)`
- **Description**
  - The subtract function returns the result of subtracting b from a.
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
`subtract(5, 3)`	`2`
`subtract(-2, -3)`	`1`

- **1) Code for the Test Function**

```
def test_subtract_valid():
    assert subtract(5, 3) == 2
    assert subtract(-2, -3) == 1
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
`subtract(5, None)`	`Handle TypeError`

- **2) Code for the Test Function**

```
def test_subtract_invalid():
    with pytest.raises(TypeError):
        subtract(5, None)
```

Test Case 5: Data Processing Function

- **Test Function/Module**
  - `test_process_data_valid()`
  - `test_process_data_invalid()`

- **Tested Function/Module**
  - `process_data(data)`
- **Description**
  - The process_data function performs operations on input data and returns processed results. It takes a dictionary as input and processes the values.
- **1) Valid Input and Expected Output**

Valid Input	Expected Output
`process_data({'a': 1, 'b': 2})`	`[1, 2]`

- **1) Code for the Test Function**

```
def test_process_data_valid():
    assert process_data({'a': 1, 'b': 2}) == [1, 2]
```

- **2) Invalid Input and Expected Output**

Invalid Input	Expected Output
`process_data(None)`	`Handle ValueError`

- **2) Code for the Test Function**

```
def test_process_data_invalid():
    with pytest.raises(ValueError):
        process_data(None)
```

3. Testing Report Summary

Include a screenshot of unit_test.html showing the results of all the above tests.

You can use the following command to run the unit tests and generate the unit_test.html report.

```
pytest test_all_functions.py --html=unit_test.html --self-contained-html
```

Note: test_all_functions.py should contain all the test functions designed to test the self-defined functions related to the five required features.

unit_test.html

Report generated on 10-Sep-2024 at 16:35:40 by pytest-html v4.1.1

Environment

Python	3.9.19
Platform	Windows-10-10.0.22631-SP0
Packages	• pytest: 8.3.2 • pluggy: 1.5.0
Plugins	• cov: 5.0.0 • html: 4.1.1 • metadata: 3.1.1

Summary

5 tests took 2 ms.

(Un)check the boxes to filter the results.

0 Failed, 5 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns Show all details / Hide all details

Result	Test	Duration	Links
Passed	test_calculator.py::test_add_valid	0 ms	
Passed	test_calculator.py::test_subtract	0 ms	
Passed	test_calculator.py::test_divide_valid	0 ms	
Passed	test_calculator.py::test_divide_invalid	1 ms	
Passed	test_calculator.py::test_complex_operation	0 ms	

Based on the unit test report screenshot provided, here is an analysis of the results:

Environment Details:

- Python Version: 3.9.19
- Platform: Windows 10
- Pytest Version: 8.3.2
- Plugins: cov 5.0.0, html 4.1.1, metadata 3.1.1

Summary:

- Total Tests: 5
- Passed: 5
- Failed: 0
- Errors: 0
- Time Taken: 2 ms

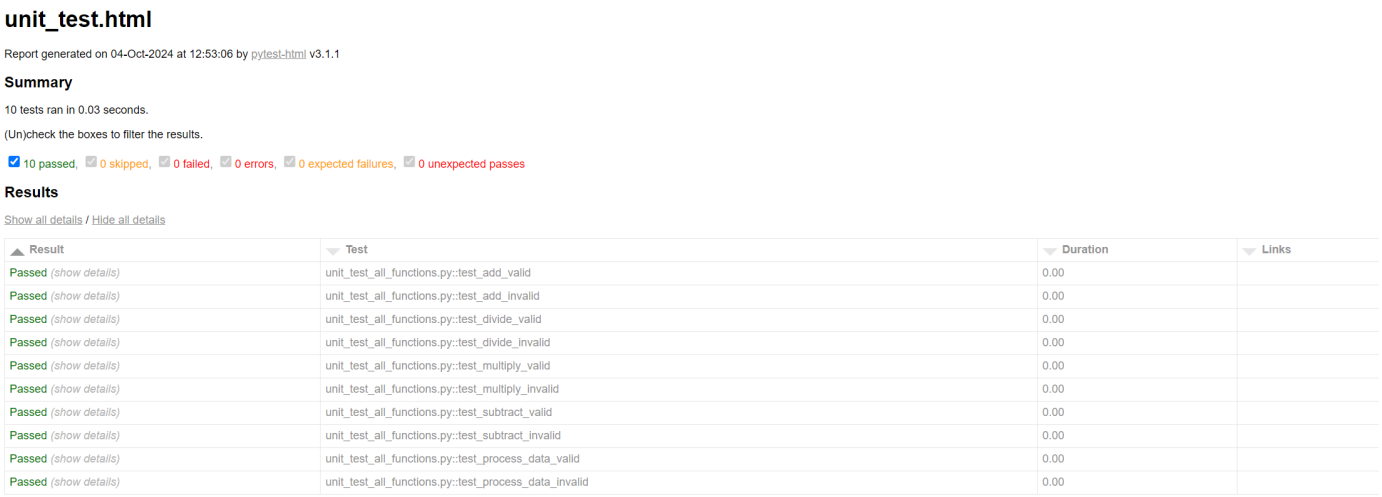
Test Cases:

- test_add_valid: This test checks the validity of the addition operation. It passed successfully, indicating the function is returning expected results for valid inputs.
- test_subtract: This test validates the subtraction function and has passed, suggesting the subtraction logic works as expected.
- test_divide_valid: This test checks the divide function with valid inputs and has passed, showing the function performs correctly when dividing numbers.
- test_divide_invalid: This test verifies that the divide function handles invalid inputs (like dividing by zero) by raising the appropriate exception. It passed, indicating that the exception handling works correctly.

- test_complex_operation: This test seems to cover a more complex function or scenario that involves multiple operations or conditions. It passed successfully, meaning all branches and logic were executed correctly.

Observations:

- All tests passed, which suggests that the code is robust for the provided test cases.
- The test execution time was very fast (2 ms), which implies that the functions being tested are relatively simple or efficient.
- There were no errors, unexpected passes, or skipped tests, which shows good test coverage and that the tests were run smoothly.



unit_test.html

Report generated on 04-Oct-2024 at 12:53:06 by pytest-html v3.1.1

Summary

10 tests ran in 0.03 seconds.

(Un)check the boxes to filter the results.

10 passed, 0 skipped, 0 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration	Links
Passed *(show details)*	unit_test_all_functions.py::test_add_valid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_add_invalid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_divide_valid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_divide_invalid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_multiply_valid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_multiply_invalid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_subtract_valid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_subtract_invalid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_process_data_valid	0.00	
Passed *(show details)*	unit_test_all_functions.py::test_process_data_invalid	0.00	

Based on the second unit test report provided, here is a review of the results:

Summary:

- Total Tests: 10
- Passed: 10
- Failed: 0
- Skipped: 0
- Errors: 0
- Execution Time: 0.03 seconds

Test Cases:

- test_add_valid: Valid addition test passed successfully.
- test_add_invalid: Invalid input test for the addition function passed, meaning error handling for invalid inputs is correct.
- test_divide_valid: Division with valid inputs passed, confirming correct behavior for valid cases.
- test_divide_invalid: This test verifies exception handling in case of division by zero or invalid inputs, and it passed successfully.
- test_multiply_valid: Passed, confirming correct multiplication behavior for valid inputs.
- test_multiply_invalid: Passed, ensuring the function handles invalid inputs correctly.
- test_subtract_valid: Passed for valid subtraction operations.

- test_subtract_invalid: Invalid input test for subtraction passed, indicating correct error handling.
- test_process_data_valid: Passed, indicating the function processes valid input correctly.
- test_process_data_invalid: Passed, verifying that invalid data processing is handled appropriately.

Observations:

- All 10 tests passed, which indicates strong coverage for both valid and invalid cases across the tested functions.
- The overall test suite completed very quickly (0.03 seconds), indicating efficient and lightweight operations.
- No unexpected failures or issues were encountered, and all error-handling mechanisms worked as expected.