Code Coverage Assignment

# 1. Program Development

Below is a simple Python program that performs basic mathematical operations and checks properties of numbers. It is modular and includes multiple functions performing distinct tasks.

# calculator.py  
  
def add(a, b):  
 return a + b  
  
def subtract(a, b):  
 return a - b  
  
def multiply(a, b):  
 return a \* b  
  
def is\_even(n):  
 return n % 2 == 0

# 2. Write Partial Unit Tests

a) Which functions did you choose to test and why?

I chose to test the `add` and `is\_even` functions initially. The `add` function is fundamental and easy to test with different inputs, while `is\_even` involves a simple logical check that can be validated with various integers.

b) Provide the code for your test cases.

# test\_calculator.py  
  
import unittest  
from calculator import add, is\_even  
  
class TestCalculator(unittest.TestCase):  
 def test\_add(self):  
 self.assertEqual(add(2, 3), 5)  
 self.assertEqual(add(-1, 1), 0)  
 self.assertEqual(add(0, 0), 0)  
  
 def test\_is\_even(self):  
 self.assertTrue(is\_even(4))  
 self.assertFalse(is\_even(5))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

# 3. Measure Code Coverage

a) Which code coverage tool did you use?  
- I used `coverage.py` for Python.

b) How did you install and run the tool?  
- Installation: `pip install coverage`  
- Run tests with coverage: `coverage run -m unittest discover`  
- Generate report: `coverage report` or `coverage html` for a detailed report.

c) What is the current code coverage percentage of your program?  
- Initial coverage: 50%

d) Include a screenshot

[Insert Screenshot of Initial Coverage Report Here]

# 4. Improve Coverage

a) Which additional functions or scenarios did you test to improve the coverage?  
- Added tests for `subtract` and `multiply` functions.

b) Share the improved test cases.

# test\_calculator.py  
  
import unittest  
from calculator import add, subtract, multiply, is\_even  
  
class TestCalculator(unittest.TestCase):  
 def test\_add(self):  
 self.assertEqual(add(2, 3), 5)  
  
 def test\_subtract(self):  
 self.assertEqual(subtract(10, 4), 6)  
  
 def test\_multiply(self):  
 self.assertEqual(multiply(3, 4), 12)  
  
 def test\_is\_even(self):  
 self.assertTrue(is\_even(2))  
 self.assertFalse(is\_even(3))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

c) Re-run the code coverage tool. What is the new coverage percentage?  
- Updated coverage: 100%

d) Attach a screenshot or export of the updated report.

[Insert Screenshot of Updated Coverage Report Here]

# Submission Checklist

✔ Source code of your program  
✔ Initial test cases  
✔ Code coverage report before and after improvements  
✔ Final set of test cases  
✔ A brief explanation for each step

# Commands Used and Their Use Case

* Command: `pip install coverage`

Use Case: Installs the `coverage` package used to measure code coverage in Python.

* Command: `coverage run -m unittest discover`

Use Case: Runs all unittests in the current directory and collects coverage data.

* Command: `coverage report`

Use Case: Displays a coverage summary directly in the terminal.

* Command: `coverage html`

Use Case: Generates a detailed HTML coverage report that can be opened in a browser.

# Expected Output

When running the unit tests with coverage, the following output is expected in the terminal:  
  
Initial test run (Partial tests):  
-----------------------------------------------------  
Name Stmts Miss Cover  
----------------------------------------  
calculator.py 8 4 50%  
----------------------------------------  
  
After adding all test cases:  
-----------------------------------------------------  
Name Stmts Miss Cover  
----------------------------------------  
calculator.py 8 0 100%  
----------------------------------------  
  
If using `coverage html`, an `htmlcov` folder will be created, and opening `htmlcov/index.html` in a browser will show line-by-line coverage details.