# Addis Ababa Institute of Technology
# School of Information Technology and Scientific Computing

## Quality Assurance and Software Testing

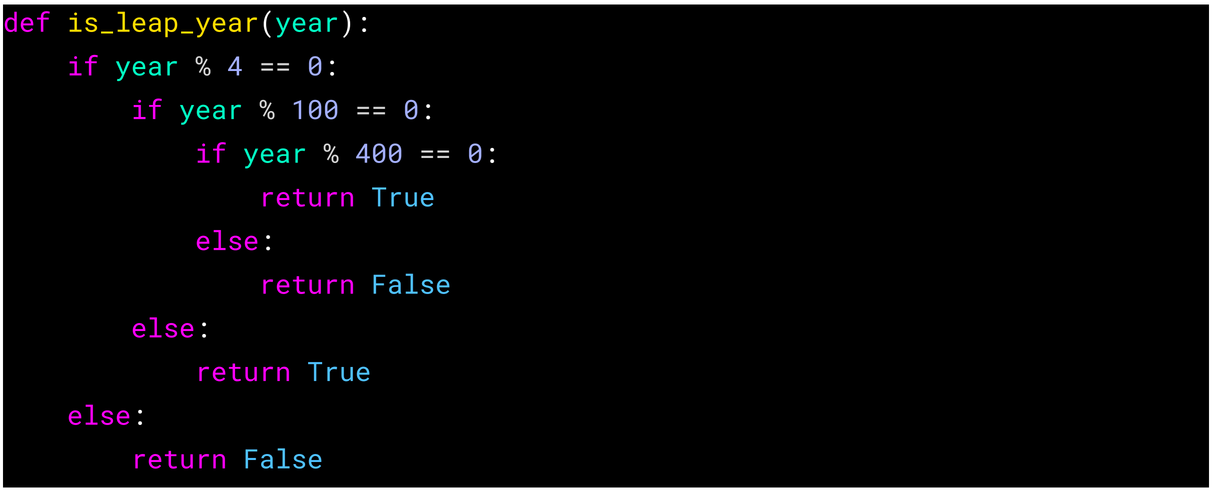
# White Box Testing Techniques
# Lab Report

Fikremariam Anteneh
UGR/9301/13
Software Section 1

Date: May 23, 2025
Submitted to: Instructor Wondimagegn Desta

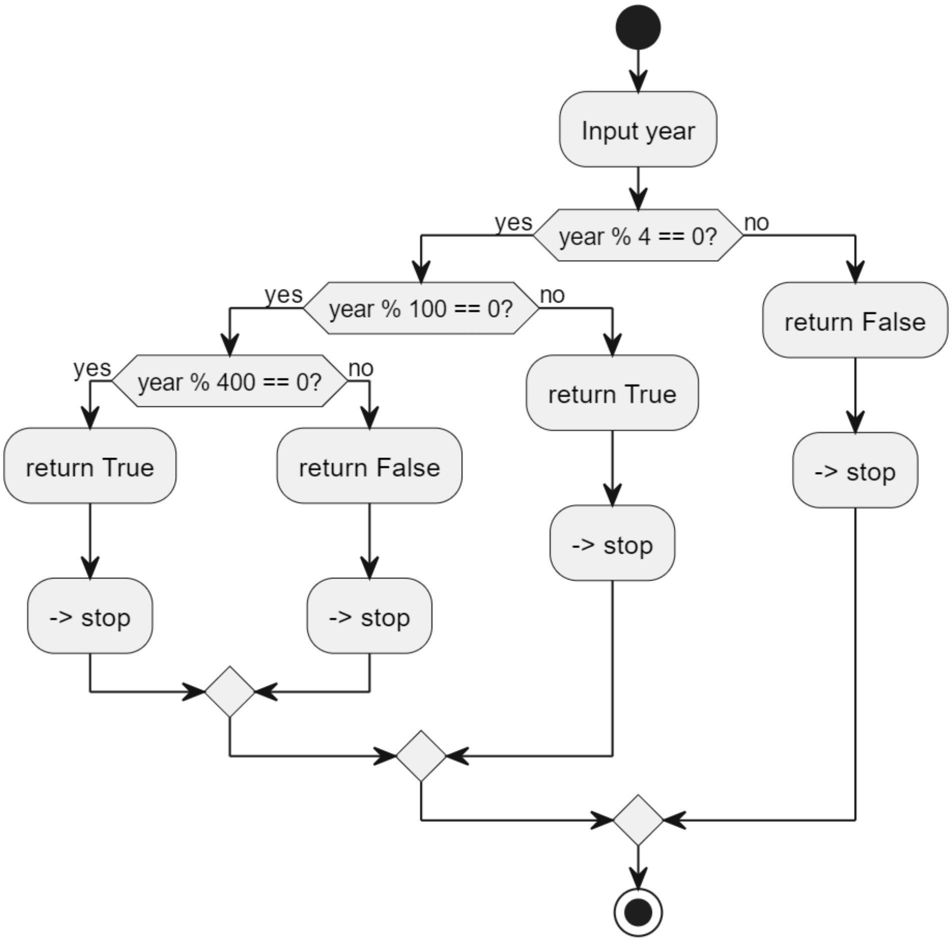
# 1. Activity 1: Control Flow Graph & Cyclomatic Complexity

The Function I selected for this task is checking whether a year is a leap year. A leap year is identified by a simple rule: a year is a leap year if it's divisible by 4, but with a few exceptions for century years. Century years (years divisible by 100) are leap years only if they are also divisible by 400. Github Link to the Activity



```python
def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

## Control Flow Graph



## Cyclomatic Complexity

| Number of Nodes | Number of Edges |
|---|---|
| 1. Start<br>2. Input year<br>3. if (year % 4 == 0?)<br>4. if (year % 100 == 0?)<br>5. if (year % 400 == 0?)<br>6. return True (400 branch)<br>7. return False (400 branch)<br>8. return True (100 branch)<br>9. return False (4 branch)<br>10. stop | 1. start → Input year<br>2. Input year → if (year % 4 == 0?)<br>3. if (year % 4 == 0?) yes → if (year % 100 == 0?)<br>4. if (year % 4 == 0?) no → return False<br>5. if (year % 100 == 0?) yes → if (year % 400 == 0?)<br>6. if (year % 100 == 0?) no → return True<br>7. if (year % 400 == 0?) yes → return True<br>8. if (year % 400 == 0?) no → return False<br>9. return True (400) → stop<br>10. return False (400) → stop<br>11. return True (100) → stop<br>12. return False (4) → stop |

P is the number of connected components, and it is 1.

**C = E - N + 2P**
**= 12 - 10 + 2(1)**
**= 4**

## Linearly Independent Paths

| Path | Execution Steps | Return Value |
|---|---|---|
| Path 1 | if year % 4 == 0 → False | False |
| Path 2 | if year % 4 == 0 → True<br>if year % 100 == 0 → False | True |
| Path 3 | if year % 4 == 0 → True<br>if year % 100 == 0 → True<br>if year % 400 == 0 → True | True |
| Path 4 | if year % 4 == 0 → True<br>if year % 100 == 0 → True<br>if year % 400 == 0 → False | False |

These four paths are linearly independent and cover all possible execution flows in the function.

## Test Case for Each Path

| Path | Test Input (year) | Why it works | Expected Output |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Path 1 | 2019 | Not divisible by 4 | False |
| Path 2 | 2024 | Divisible by 4 but not by 100 | True |
| Path 3 | 2000 | Divisible by 400 | True |
| Path 4 | 1900 | Divisible by 100 but not by 400 | False |

## 2. Activity 2: Statement, Branch, and Condition Coverage

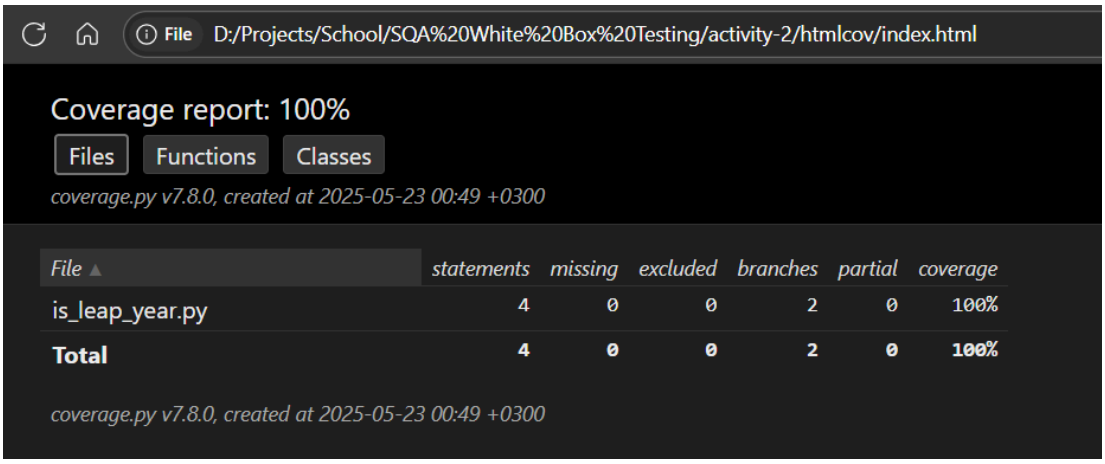
I used the same function for leap year for this one. But a little bit of modification so that it has a compound conditional statement. Github Link For the Activity

```
def is_leap_year(year):
    if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
        return True
    else:
        return False
```

```

class TestLeapYear(unittest.TestCase):
    def test_path_1(self):
        self.assertFalse(is_leap_year(2019))
    def test_path_2(self):
        self.assertTrue(is_leap_year(2024))
    def test_path_3(self):
        self.assertTrue(is_leap_year(2000))
    def test_path_4(self):
        self.assertFalse(is_leap_year(1900))
```

This is the coverage report



D:/Projects/School/SQA%20White%20Box%20Testing/activity-2/htmlcov/index.html

Coverage report: 100%

*coverage.py v7.8.0, created at 2025-05-23 00:49 +0300*

| *File* ▲ | *statements* | *missing* | *excluded* | *branches* | *partial* | *coverage* |
|---|---|---|---|---|---|---|
| is_leap_year.py | 4 | 0 | 0 | 2 | 0 | 100% |
| **Total** | **4** | **0** | **0** | **2** | **0** | **100%** |

*coverage.py v7.8.0, created at 2025-05-23 00:49 +0300*

### 100% statement coverage

From the report, we see that coverage is 100%.

### 100% branch (decision) coverage

We must make the entire if condition evaluate to True and False at least once.

Branches in this function:

The if condition: True → returns True with the example 2024

The if condition: False → returns False with the example 2019

**Achieves 100% branch coverage.**

### 100% condition coverage

We need every individual condition inside the compound expression to evaluate to True and False at least once. The conditional is the below one and we have 3 atomic condition.

```
if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)
```

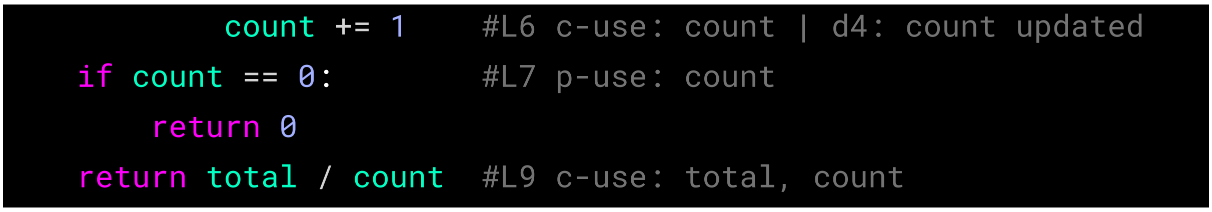
| Test Case | Year | %4==0 | %100!=0 | %400==0 | Result | Why |
|---|---|---|---|---|---|---|
| TC1 | 2024 | True | True | False | True | leap year |
| TC2 | 2000 | True | False | True | True | Div by 400 |
| TC3 | 1900 | True | False | False | False | Div by 100 but not 400 |
| TC4 | 2019 | False | – | – | False | Not div by 4 |

These 4 cases force each condition to be both True and False and achieve 100% condition coverage.

## 3. Activity 3: Data Flow Testing

For this task, I selected a function that accepts an array and returns the average of only the even numbers. Github Link For the Activity

```
def average_of_evens(arr):
    total = 0              #L1 d1: total
    count = 0              #L2 d2: count
    for num in arr:        #L3 p-use: arr | loop var: num
        if num % 2 == 0:   #L4 p-use: num
            total += num   #L5 c-use: total,num | d3: total updated
```



```
            count += 1    #L6 c-use: count | d4: count updated
    if count == 0:        #L7 p-use: count
        return 0
    return total / count  #L9 c-use: total, count
```

**Identify definition (d), computation-use (c-use), and predicate-use (p-use)points.**

| **Variable** | **Definition (d)** | **c-use (Computation)** | **p-use (Predicate)** |
|---|---|---|---|
| total | d1, d3 | total += num, total / count | |
| count | d2, d4 | count += 1, total / count | if count == 0 |
| arr | (param) | | for num in arr |
| num | (loop var) | num % 2, total += num | if num % 2 == 0 |

**Create DU pairs and**

For total

| Definition | Use | Type |
|---|---|---|
| d1: total = 0 | total += num | c-use |
| d1: total = 0 | return total / count | c-use (if loop doesn't run) |
| d3: total += num | return total / count | c-use |

For count

| Definition | Use | Type |
|---|---|---|
| d2: count = 0 | count += 1 | c-use |
| d2: count = 0 | if count == 0 | p-use (if loop doesn't run) |
| d2: count = 0 | return total / count | c-use (if loop doesn't run) |
| d4: count += 1 | if count == 0 | p-use |
| d4: count += 1 | return total / count | c-use |

**DU paths (L1 L2 express the line number and are annotated in the code**

| No | Variable | Def | Use Location & Type | DU Path | Valid Only If |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| 1 | total | d1 | L5: c-use<br>total += num | L1 → L3 → L4 → L5 | At least one even number exists → total is updated. |
| 2 | total | d1 | L9: c-use<br>return total / count | L1 → L3 → L7 → L9 | Loop doesn’t run or no even numbers → total is not updated. |
| 3 | total | d3 | L9: c-use<br>return total / count | L1 → L3 → L4 → L5 → L7 → L9 | At least one even number exists → total is updated then used. |
| 4 | count | d2 | L6: c-use<br>count += 1 | L2 → L3 → L4 → L6 | At least one even number exists → count is incremented. |
| 5 | count | d2 | L7: p-use<br>if count == 0: | L2 → L3 → L7 | Loop doesn’t run or no evens → count stays 0. |
| 6 | count | d2 | L9: c-use<br>return total / count | L2 → L3 → L7 → L9 | No even number → count remains 0 (division by zero is guarded). |
| 7 | count | d4 | L7: p-use<br>if count == 0: | L2 → L3 → L4 → L6 → L7 | At least one even number exists → count is updated and used in check. |
| 8 | count | d4 | L9: c-use<br>return total / count | L2 → L3 → L4 → L6 → L7 → L9 | At least one even number exists → count is updated and used in return. |

**Test Case and DU Coverage Table**

| Test Case | Covers DU Paths | Why it Covers Them |
|---|---|---|
| Test 1: [] | 2, 5, 6 | Loop doesn't run → count and total not updated; tests original definitions use. |
| Test 2: [1, 3, 5] | 2, 5, 6 | Loop runs, but no even numbers → same coverage as empty list. |
| Test 3: [2] | 1, 3, 4, 7, 8 | Single even → total & count updated and used in predicate and final return. |
| Test 4: [2, 4, 6] | 1, 3, 4, 7, 8 | All even → multiple updates to total & count → full coverage of redefinitions. |
| Test 5: [1, 2, 3, 4] | 1, 3, 4, 7, 8 | Mixed input → at least one even ensures count/total are redefined & used. |

```
class TestAverageOfEvens(unittest.TestCase):
    def test_average_of_evens(self):
        # Test Case 1 - Empty List
        self.assertEqual(average_of_evens([]), 0)
        # Test Case 2 - No Even Numbers
        self.assertEqual(average_of_evens([1, 3, 5]), 0)
        # Test Case 3 - Only Even Numbers
        self.assertEqual(average_of_evens([2, 4, 6]), 4.0)
        # Test Case 4 - Mix of Even and Odd
        self.assertEqual(average_of_evens([1, 2, 3, 4]), 3.0)
        # Test Case 5 - Single Even Number
        self.assertEqual(average_of_evens([8]), 8.0)
```

## 4. Activity 4: Mutation Testing

For this activity, I selected the is_leap_year function implemented in Activity 1. I ran the previous test case to see how they would perform on the mutants, how good they are, and other tests. Github Link For the Activity

Mutation 1 code:

```
def is_leap_year_mutant1(year):
    if year % 4 != 0:  # Mutation: '==' → '!='
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

Mutation 2 code:

```
def is_leap_year_mutant2(year):
    if year % 4 == 0:
        if year % 100 != 0:  # Mutation: '==' → '!='
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

Mutation 3 Code:

```
def is_leap_year_mutant3(year):
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 != 0:  # Mutation: '==' → '!='
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

Mutation 4 code:

```
def is_leap_year_mutant4(year):
    if year % 4 == 0:
        if (
            year % 100 == 0 or year % 400 != 0
        ):  # Mutation: Composition of conditions and the logic must be connected with and instead of or
            # This condition is incorrect because it will return True for years that are divisible by 100 but not by 400
            return True
        else:
            return False
    else:
        return False
```

Test Cases Used: [2019, 2024, 2000, 1900, 2100, 2400, 2023, 1996]

| Mutant # | Mutation Description | Code Change Snippet | Test Case Failed | Status |
|---|---|---|---|---|
| 1 | Changed % 4 == 0 to % 4 != 0 | if year % 4 != 0: | 2019, 2024, 2000, 2400, 2023, 1996 | Killed |
| 2 | Changed % 100 == 0 to % 100 != 0 | if year % 100 != 0: | 2024, 1900, 2100, 1996 | Killed |
| 3 | Changed % 400 == 0 to % 400 != 0 | if year % 400 != 0: | 2000, 1900, 2100, 2400 | Killed |
| 4 | Incorrect logic using or instead of and | if year % 100 == 0 or year % 400 != 0: | 1900, 2100 | Killed |

### Mutation Score

- **Total Mutants**: 4
- **Killed Mutants**: 4
- **Survived Mutants**: 0
- **Mutation Score**: **100%**

All introduced faults were detected by the current test suite.

## 5. Activity 5: JUnit Unit Testing

Github Link For the Activity

Calculator Code

```
public class Calculator {

    public int add(int a, int b) {
        return a + b;
    }
    public int subtract(int a, int b) {
        return a - b;
    }
    public int multiply(int a, int b) {
        return a * b;
    }
    public double divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero.");
        }
        return (double) a / b;
    }
}
```

Calculator Test Code

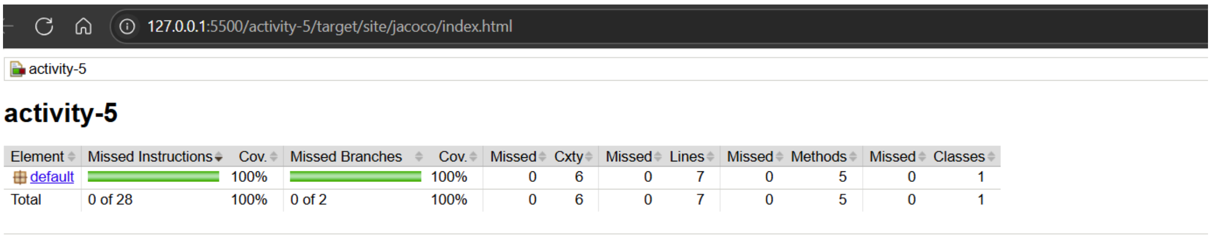
```
public class CalculatorTest {

    Calculator calc = new Calculator();

    @Test
    public void testAdd() {
        assertEquals(7, calc.add(3, 4), "3 + 4 should equal 7");
    }
    @Test
    public void testAddWithAssertTrue() {
        assertTrue(calc.add(3, 4) == 7, "3 + 4 should equal 7");
    }
    @Test
    public void testSubtract() {
        assertEquals(5, calc.subtract(10, 5), "10 - 5 should equal 5");
    }
    @Test
    public void testSubtractWithAssertTrue() {
        assertTrue(calc.subtract(10, 5) == 5, "10 - 5 should equal 5");
    }
    @Test
    public void testMultiply() {
        assertEquals(20, calc.multiply(4, 5), "4 * 5 should equal 20");
    }
    @Test
    public void testMultiplyWithAssertTrue() {
        assertTrue(calc.multiply(4, 5) == 20, "4 * 5 should equal 20");
    }
    @Test
    public void testDivide() {
        assertEquals(2.5, calc.divide(5, 2), 0.0001, "5 / 2 should equal 2.5");
    }
    @Test
    public void testDivideWithAssertTrue() {
        assertTrue(Math.abs(calc.divide(5, 2) - 2.5) < 0.0001, "5 / 2 should equal 2.5");
    }
    @Test
    public void testDivideByZero() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            calc.divide(5, 0);
        });
        assertEquals("Cannot divide by zero.", exception.getMessage());
    }
}
```

## Test Document and Result Screenshot

| Test Case ID | Method Tested | Input | Expected Output | Assertion Used | Result |
|---|---|---|---|---|---|
| TC01 | add(int, int) | 3, 4 | 7 | assertEquals | Pass |
| TC02 | add(int, int) | 3, 4 | 7 | assertTrue | Pass |
| TC03 | subtract(int, int) | 10, 5 | 5 | assertEquals | Pass |
| TC04 | subtract(int, int) | 10, 5 | 5 | assertTrue | Pass |
| TC05 | multiply(int, int) | 4, 5 | 20 | assertEquals | Pass |
| TC06 | multiply(int, int) | 4, 5 | 20 | assertTrue | Pass |
| TC07 | divide(int, int) | 5, 2 | 2.5 | assertEquals (Δ=0.0001) | Pass |
| TC08 | divide(int, int) | 5, 2 | 2.5 | assertTrue with tolerance | Pass |
| TC09 | divide(int, int) | 5, 0 | Exception thrown | assertThrows | Pass |

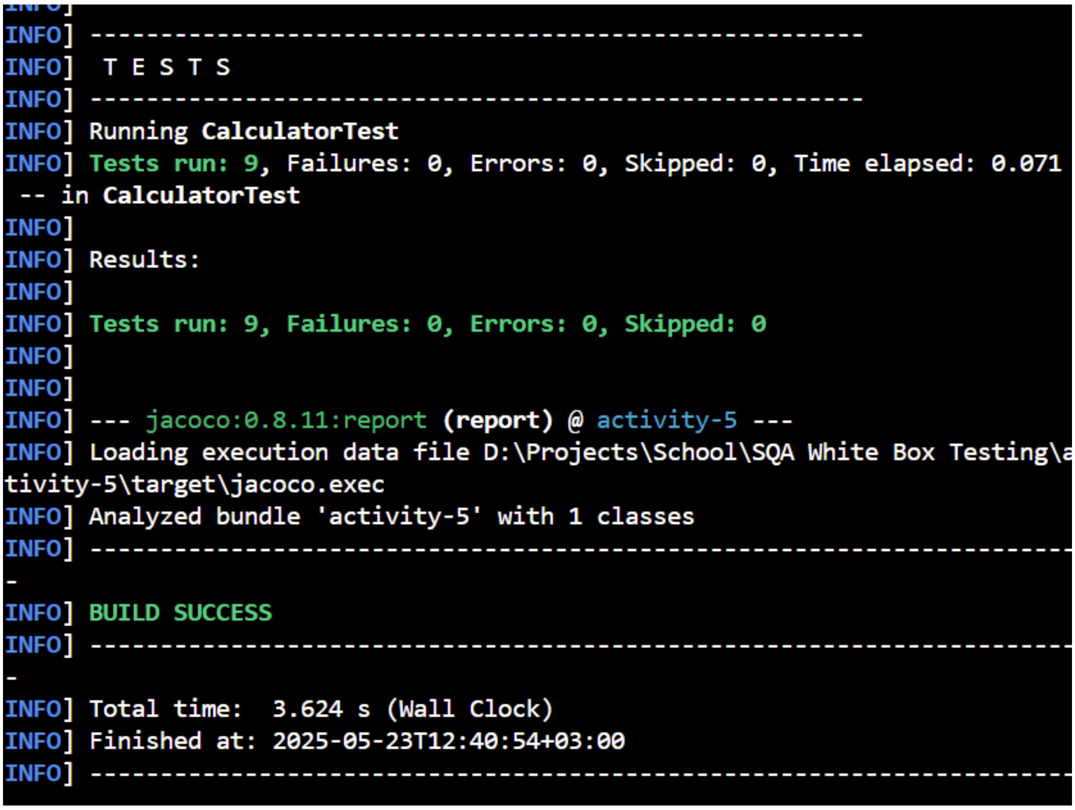


127.0.0.1:5500/activity-5/target/site/jacoco/index.html

activity-5

## activity-5

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| default | 🟩 | 100% | 🟩 | 100% | 0 | 6 | 0 | 7 | 0 | 5 | 0 | 1 |
| Total | 0 of 28 | 100% | 0 of 2 | 100% | 0 | 6 | 0 | 7 | 0 | 5 | 0 | 1 |



```
INFO] -------------------------------------------------------
INFO]  T E S T S
INFO] -------------------------------------------------------
INFO] Running CalculatorTest
INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.071
 -- in CalculatorTest
INFO]
INFO] Results:
INFO]
INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
INFO]
INFO]
INFO] --- jacoco:0.8.11:report (report) @ activity-5 ---
INFO] Loading execution data file D:\Projects\School\SQA White Box Testing\a
tivity-5\target\jacoco.exec
INFO] Analyzed bundle 'activity-5' with 1 classes
INFO] ------------------------------------------------------------------------
-
INFO] BUILD SUCCESS
INFO] ------------------------------------------------------------------------
-
INFO] Total time:  3.624 s (Wall Clock)
INFO] Finished at: 2025-05-23T12:40:54+03:00
INFO] ------------------------------------------------------------------------
```