TU856/3
SOFTWARE TESTING

LAB 3

24/02/2025

PAULINA CZARNOTA C21365726

Unit Testing and Test-Driven Development

Summary

This lab covers unit testing and test-driven development (TDD). We have explored unit testing in three ways:

1. Defect detection using black-box testing.
2. Testing based on business requirements.
3. Developing a Wordle game using TDD.

Unit tests help detect defects early in the software lifecycle, reducing development costs. TDD improves software design and reliability by ensuring all features have corresponding tests before implementation.

Step 1: Unit Testing for Defect Detection

Task: Identify and fix defects in the `Rectangle` class.

Instructions:

- Create a `rectangle.py` file and copy the given code.
- Analyze the provided implementation.
- Identify any potential bugs using unit testing.
- Fix the bugs and ensure all tests pass.

Bugs Found & Fixed:

- Typo in `self.lenght` → Fixed to `self.length`.
- Incorrect `rotate()` implementation → Now correctly swaps width and length.
- Area update issue when modifying width or length → Ensured `self.area` is updated correctly.

Fixed `Rectangle` Class Implementation

Refer to `rectangle.py` for the corrected code.

Unit Tests (`test_rectangle.py`)

- Tested initialization, set methods, area calculation, and rotation.
- Used `unittest` framework for correctness.

Step 2: Unit Testing Based on Business Requirements

Task: Validate the `Calculator` class based on defined functional requirements.

Instructions:

- Implement unit tests based on provided business requirements.
- Execute failing tests first.
- Implement code to satisfy the test cases.
- Re-run the tests and ensure all pass.

Issues Fixed:

- Multiplication operator (`**`) incorrect → Changed to `*`.
- Subtraction logic was reversed → Fixed operand order.
- Handled division by zero → Now raises `ValueError`.

Fixed `Calculator` Class Implementation

Refer to `calculator.py` for the corrected implementation.

Unit Tests (`test_calculator.py`)

- Validated addition, subtraction, multiplication, division.
- Checked edge cases, including division by zero.

Step 3: Test-Driven Development - Wordle Game

Task: Implement a Wordle game using TDD principles.

Instructions:

1. Write failing tests first.
2. Implement the minimal code to pass the test.
3. Refactor the code to improve design while keeping tests green.

Core Features Developed:

- Word validation (must be 5 letters).
- Six-attempt limit for guessing.
- Basic logic for correct and incorrect guesses.

Wordle Class Implementation (`wordle.py`)

Refer to `wordle.py` for details.

Unit Tests (`test_wordle.py`)

- Checked valid word length (must be 5 letters).
- Ensured game ends after six incorrect attempts.
- Verified valid words only can be guessed.

Step 4: Unit Test Execution Proof

The screenshot below shows the successful execution of unit tests in Visual Studio Code (VS Code) using `unittest`:

- The command `python -m unittest discover` was used to run all test cases.
- Each individual test file (`test_rectangle.py`, `test_calculator.py`, `test_wordle.py`) was executed separately.
- The "OK" output confirms that all tests passed without errors.