Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases

NAME – ANDAPALLY SHIVANI

HTNO – 2403A51400

BATCH NO – 16

## TASK 1:

Generate test cases for a function is\_prime(n) and implement the function. Requirements:

- Only integers > 1 can be prime.

- Check edge cases: 0, 1, 2, negative numbers, and large primes

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

## EXPLANATION:

This code defines an is\_prime(n) function that checks if a number n is prime. It handles edge cases like 0, 1, 2, and negative numbers. For numbers greater than 2, it efficiently checks for divisibility by odd numbers up to the square root of n. The code also includes a list of test\_cases to verify the function's correctess. It then iterates through these test cases, compares the is\_prime function's output to the expected\_result, prints whether each test passed or failed, and provides a summary of the test results.

## TASK 2:

Generate test cases for celsius\_to\_fahrenheit(c) and fahrenheit\_to\_celsius(f). Requirements: - Validate known pairs: 0°C = 32°F, 100°C = 212°F. - Include decimals and invalid inputs like strings or None.

A screenshot of a computer

AI-generated content may be incorrect.

## OUTPUT:

A screenshot of a computer program

AI-generated content may be incorrect.

## EXPLANATION:

The code defines two functions: celsius\_to\_fahrenheit and fahrenheit\_to\_celsius. Both functions convert temperatures and handle non-numeric inputs by returning None. Test cases are provided for both conversions, including valid and invalid inputs. The code then runs these tests, comparing the function outputs to expected results. It uses math.isclose for Fahrenheit to Celsius tests to handle potential floating-point inaccuracies, printing the outcome of each test and a summary for both sets of conversions.

## TASK 3:

Write test cases for a function count\_words(text) that returns the number of words in a sentence. Requirements: - Handle normal text, multiple spaces, punctuation, and empty strings.

A screenshot of a computer

AI-generated content may be incorrect.

## OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

## EXPLANATION:

* **Generate test cases**: Generate a list of test cases for the count\_words(text) function, including normal text, multiple spaces, punctuation, and empty strings.
* **Implement count words(text) function**: Implement the count\_words(text) function to accurately count words based on the requirements.
* **Write and run tests**: Write code to run the generated test cases against the count\_words(text) function and display the results, clearly indicating which tests pass or fail.
* **Finish task**: Put the findings from the earlier stages into a format that anyone can read.

## TASK 4:

Generate test cases for a BankAccount class with: - Methods: deposit(amount), withdraw(amount), check\_balance() - Requirements: - Negative deposits/withdrawals should raise an error. - Cannot withdraw more than balance

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

## EXPLANATION:

This code attempts to implement a count\_words function. It went through several revisions trying to accurately count words while handling spaces and punctuation based on provided test cases. The initial attempts used splitting and regex for cleaning, but struggled with hyphenated words and attached punctuation. The final attempt splits by whitespace and counts tokens containing at least one word character. While it passed most tests, some specific cases with hyphens and punctuation remained challenging due to the test cases' implicit word definition.

## TASK 5:

Prompt

Generate test cases for is\_number\_palindrome(num), which checks if an integer reads the same backward. Examples:

- 121 → True

- 123 → False

- 0, negative numbers → handled gracefully

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## EXPLANATION:

This code defines a function is\_number\_palindrome(num) to determine if an integer is a palindrome. It first checks if the input number is negative and returns False if it is, as negative numbers are typically not considered palindromes. For non-negative numbers, it converts the absolute value of the number to a string. Then, it reverses this string. Finally, it compares the original string representation with the reversed string. If they are the same, the function returns True, indicating the number is a palindrome; otherwise, it returns False. The code also includes a list of test\_cases to validate the function. It iterates through these cases, calls is\_number\_palindrome, and prints the test results, including a summary of passed tests.