

AI ASSISTED CODING

Name: P.Swaran Raj

Roll.No: 2303A52247

Batch: 37

Assignment- 8.2

Task 1 – Test-Driven Development for Even/Odd Number Validator

- Use AI tools to first generate test cases for a function `is_even(n)` and then implement the function so that it satisfies all generated Tests.

Prompt:

“#Generate unit test cases for a Python function `is_even(n)` and then implement the function to pass all tests.”

The screenshot shows a Python script named `AI Assist8.2.py` in a code editor. The code defines a test function `test_is_even()` with five assertions for the `is_even` function. It then runs this test function. The terminal below shows the command run and the output "All test cases passed!".

```
AI Assist8.2.py > ...
1 #Generate unit test cases for a Python function is_even(n) and then implement the function to pass all
2 # Unit test cases for the is_even function
3 def test_is_even():
4     assert is_even(2) == True, "Test case 1 failed"
5     assert is_even(3) == False, "Test case 2 failed"
6     assert is_even(0) == True, "Test case 3 failed"
7     assert is_even(-2) == True, "Test case 4 failed"
8     assert is_even(-3) == False, "Test case 5 failed"
9     print("All test cases passed!")
10 # Implementation of the is_even function
11 def is_even(n):
12     return n % 2 == 0
13 # Run the test cases
14 test_is_even()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + × ... | ☰ ×

PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding> & C:/Users/pulla/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pulla/OneDrive/Documents/SRU/AI Asisted Coding/AI Assist8.2.py"
All test cases passed!
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding>
```

Explanation:

AI first generated test cases covering positive, negative, zero, and large integers. The function was implemented to return True for even numbers and False for odd numbers.

Output:

Unit Testing:

Test cases validate behavior for valid integers and edge cases.

All test cases passed!

is even(2) → True

```
is_even(7) → False  
is_even(0) → True  
is_even(-4) → True
```

Observation:

Writing tests first helped define correct behavior clearly.

TDD ensures reliable and predictable function logic.

Task 2 – Test-Driven Development for String Case Converter

- Ask AI to generate test cases for two functions:
 - `to_uppercase(text)`
 - `to_lowercase(text)`

Prompt:

“#Generate unit tests for `to_uppercase(text)` and `to_lowercase(text)` with edge cases.”

Explanation:

AI generated tests for empty strings, mixed-case text, and invalid inputs. Functions were implemented to safely convert strings while handling invalid inputs.

Output:

Unit Testing:

Tests verify correct case conversion and safe handling of None or invalid values.

`to_uppercase("ai coding") → "AI CODING"`

`to_lowercase("TEST") → "test"`

`to_uppercase("")` → `""`

Observation:

AI-generated tests improved robustness of string handling.

Edge-case testing prevents unexpected runtime errors.

Task 3 – Test-Driven Development for List Sum Calculator

- Use AI to generate test cases for a function `sum_list(numbers)` that calculates the sum of list elements.

Prompt:

“#Generate unit test cases for a function sum_list(numbers) handling edge cases.”

Explanation:

The function sums numeric values while ignoring non-numeric elements.

Empty lists and negative values are handled safely.

Output:

Unit Testing:

Tests validate normal lists, empty lists, and mixed-type inputs.

sum_list([1, 2, 3]) → 6

sum_list([]) → 0

sum_list([2, "a", 3]) → 5

Observation:

TDD helped identify corner cases early.

The function became more reliable through test coverage.

Task 4 – Test Cases for Student Result Class

- Generate test cases for a StudentResult class with the following methods:
 - add_marks(mark)
 - calculate_average()
 - get_result()

Prompt:

“#Generate unit tests for a StudentResult class with marks validation and result logic.”

Explanation:

The class stores marks, calculates average, and determines pass/fail status.

Input validation ensures marks remain within valid range.

Output:

Marks: [60, 70, 80] → Average: 70 → Result: Pass

Marks: [30, 35, 40] → Average: 35 → Result: Fail

Observation:

AI-generated tests guided correct class behavior.

TDD improved reliability of object-oriented logic.

Task 5 – Test-Driven Development for Username Validator

Requirements:

- Minimum length: 5 characters
 - No spaces allowed

- Only alphanumeric characters

Prompt:

"#Generate unit tests for a username validation function based on given constraints.."

```

AI Assist8.2.py > ...
105 #Generate unit tests for a username validation function based on given constraints
106 def is_valid_username(username):
107     if len(username) < 5 or len(username) > 20:
108         return False
109     if not username.isalnum():
110         return False
111     return True
112 # Unit test cases for the is_valid_username function
113 def test_is_valid_username():
114     assert is_valid_username("user1") == True, "Test case 1 failed" # Valid username
115     assert is_valid_username("us") == False, "Test case 2 failed" # Too short
116     assert is_valid_username("thisisaverylongusername") == False, "Test case 3 failed" # Too long
117     assert is_valid_username("user_name") == False, "Test case 4 failed" # Contains underscore
118     assert is_valid_username("user-name") == False, "Test case 5 failed" # Contains hyphen
119     assert is_valid_username("user123") == True, "Test case 6 failed" # Valid username with numbers
120     print("All test cases for is_valid_username passed!")
121 # Run the test cases
122 test_is_valid_username()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding> & C:/Users/pulla/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/pulla/OneDrive/Documents/SRU/AI Asisted Coding/AI Assist8.2.py"
All test cases for is_valid_username passed!
PS C:\Users\pulla\OneDrive\Documents\SRU\AI Asisted Coding>

Explanation:

The function checks length, spaces, and alphanumeric characters. Only usernames meeting all conditions are accepted.

Output:

Assertion Testing:

Assertions were used to validate valid and invalid usernames.

Each test checks whether the function returns True or False based on given conditions.

is_valid_username("user01") → True

is_valid_username("ai") → False

is_valid_username("user name") → False

Observation:

The test-first approach clarified the validation rules clearly.

AI-generated tests improved input validation accuracy.

Conclusion:

This lab demonstrates how test-driven development using AI improves code reliability by defining clear expectations through unit testing before implementation.