# Assignment-8.2

Perumalla Sushwanth

batch 29

2303a51567

**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.
Requirements:
• Input must be an integer
• Handle zero, negative numbers, and large integers
Example Test Scenarios:
is_even(2) → True
is_even(7) → False
is_even(0) → True
is_even(-4) → True
is_even(9) → False
Expected Output -1
• A correctly implemented is_even() function that passes all AIgenerated test cases

CODE:

```python
#generate test cases for a function is_even(n) and then implement the function so that it satisfies all generated using assert test
# Test cases for is_even function
def test_is_even():
    assert is_even(2) == True, "Test case 1 failed"
    assert is_even(3) == False, "Test case 2 failed"
    assert is_even(0) == True, "Test case 3 failed"
    assert is_even(-2) == True, "Test case 4 failed"
    assert is_even(-3) == False, "Test case 5 failed"
    print("All test cases passed!")
# Implementation of is_even function
def is_even(n):
    return n % 2 == 0
# Run the test cases
test_is_even()
```

OUTPUT:

```
rams\Python\Python312\python.exe' 'c:\Users\perum\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '5702
1' '--' 'C:\Users\perum\OneDrive\Desktop\AI_ASS_CODING\ai_assistedcoding\lab_8.2'
All test cases passed!
```

**Task 2-**

Test-Driven Development for String Case Converter

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

Requirements:

- Handle empty strings
- Handle mixed-case input
- Handle invalid inputs such as numbers or None

Example Test Scenarios:

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

to_lowercase("TEST") → "test"

to_uppercase("") → ""

to_lowercase(None) → Error or safe handling

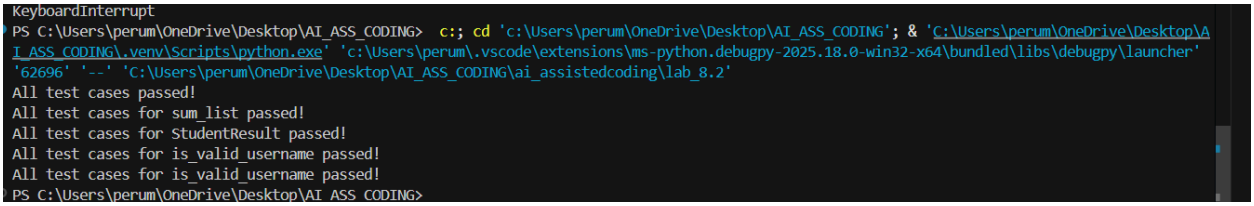
Expected Output -2

- Two string conversion functions that pass all AI-generated test cases with safe input handling.

CODE:

```python
#generate test cases for two functions to_uppercase(text) and to_lowercase(text)
# Test cases for to_uppercase and to_lowercase functions
def test_to_uppercase():
    assert to_uppercase("hello") == "HELLO", "Test case 1 failed"
    assert to_uppercase("World") == "WORLD", "Test case 2 failed"
    assert to_uppercase("123") == "123", "Test case 3 failed"
    assert to_uppercase("") == "", "Test case 4 failed"
    print("All test cases for to_uppercase passed!")
def test_to_lowercase():
    assert to_lowercase("HELLO") == "hello", "Test case 1 failed"
    assert to_lowercase("WORLD") == "world", "Test case 2 failed"
    assert to_lowercase("123") == "123", "Test case 3 failed"
    assert to_lowercase("") == "", "Test case 4 failed"
    print("All test cases for to_lowercase passed!")
```

OUTPUT




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.

Requirements:

• Handle empty lists

• Handle negative numbers

• Ignore or safely handle non-numeric values

Example Test Scenarios:

sum_list([1, 2, 3]) → 6

sum_list([]) → 0

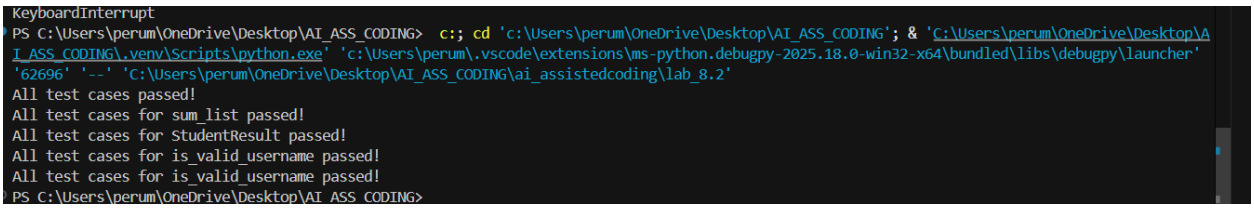
sum_list([-1, 5, -4]) → 0

sum_list([2, "a", 3]) → 5
Expected Output 3
• A robust list-sum function validated using AI-generated test cases.

CODE:

```python
#generate test cases for a function sum_list(numbers) that calculates the sum of list elements
# Test cases for sum_list function
def test_sum_list():
    assert sum_list([1, 2, 3]) == 6, "Test case 1 failed"
    assert sum_list([-1, -2, -3]) == -6, "Test case 2 failed"
    assert sum_list([0, 0, 0]) == 0, "Test case 3 failed"
    assert sum_list([]) == 0, "Test case 4 failed"
    print("All test cases for sum_list passed!")
# Implementation of sum_list function
def sum_list(numbers):
    return sum(numbers)
# Run the test cases
test_sum_list()
```

OUTPUT:




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()
Requirements:

• Marks must be between 0 and 100
• Average ≥ 40 → Pass, otherwise Fail
Example Test Scenarios:
Marks: [60, 70, 80] → Average: 70 → Result: Pass
Marks: [30, 35, 40] → Average: 35 → Result: Fail
Marks: [-10] → Error
Expected Output -4
• A fully functional StudentResult class that passes all AIgenerated test

CODE:

```python
#Generate test cases for a StudentResult class with the following methods add_marks(mark), calculate_average() and get_result()
# Test cases for StudentResult class
class StudentResult:
    def __init__(self):
        self.marks = []
    def add_marks(self, mark):
        self.marks.append(mark)
    def calculate_average(self):
        if not self.marks:
            return 0
        return sum(self.marks) / len(self.marks)
    def get_result(self):
        average = self.calculate_average()
        if average >= 90:
            return 'A'
        elif average >= 80:
            return 'B'
        elif average >= 70:
            return 'C'
        elif average >= 60:
            return 'D'
        elif average >= 50:
            return 'E'
        else:
            return 'F'
# Test cases for StudentResult class
def test_student_result():
    student = StudentResult()
    student.add_marks(90)
    student.add_marks(80)
    student.add_marks(70)
    assert student.calculate_average() == 80, "Test case 1 failed"
    assert student.get_result() == 'B', "Test case 2 failed"
    student.add_marks(100)
    assert student.calculate_average() == 85, "Test case 3 failed"
    assert student.get_result() == 'B', "Test case 4 failed"
    print("All test cases for StudentResult passed!")
# Run the test cases
test_student_result()
```

```
KeyboardInterrupt
PS C:\Users\perum\OneDrive\Desktop\AI_ASS_CODING>  c:; cd 'c:\Users\perum\OneDrive\Desktop\AI_ASS_CODING'; & 'C:\Users\perum\OneDrive\Desktop\AI_ASS_CODING\.venv\Scripts\python.exe' 'c:\Users\perum\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62696' '--' 'C:\Users\perum\OneDrive\Desktop\AI_ASS_CODING\ai_assistedcoding\lab_8.2'
All test cases passed!
All test cases for sum_list passed!
All test cases for StudentResult passed!
All test cases for is_valid_username passed!
All test cases for is_valid_username passed!
PS C:\Users\perum\OneDrive\Desktop\AI_ASS_CODING>
```

Task 5 –

Test-Driven Development for Username Validator

Requirements:

- Minimum length: 5 characters
- No spaces allowed
- Only alphanumeric characters

Example Test Scenarios:

is_valid_username("user01") → True

is_valid_username("ai") → False

is_valid_username("user name") → False

is_valid_username("user@123") → False

Expected Output 5

A username validation function that passes all AI-generated test cases.