

## **Lab 8.2: Test-Driven Development with AI**

Name: B.TejasKumar

Hall Ticket No: 2303A51489

### **Task 1 – Even/Odd Number Validator (TDD)**

#### **AI Prompt Used:**

Generate unittest test cases for a function `is_even(n)` that validates integer input, handles zero, negative numbers, and large integers.

#### **Test Cases:**

```
import unittest

class TestIsEven(unittest.TestCase):
    def test_positive_even(self):
        self.assertTrue(is_even(2))

    def test_positive_odd(self):
        self.assertFalse(is_even(7))

    def test_zero(self):
        self.assertTrue(is_even(0))

    def test_negative_even(self):
        self.assertTrue(is_even(-4))

    def test_negative_odd(self):
        self.assertFalse(is_even(9))

    def test_large_number(self):
        self.assertTrue(is_even(1000000))

    def test_invalid_input(self):
        with self.assertRaises(TypeError):
            is_even("a")
```

### Implementation:

```
def is_even(n):
    if not isinstance(n, int):
        raise TypeError("Input must be an integer")
    return n % 2 == 0
```

### Expected Output:

All tests passed successfully.

```
main.py
```

```
1 # Task 1 - Even/Odd Validator
2
3 def is_even(n):
4     if not isinstance(n, int):
5         return "Error: Input must be an integer"
6     return n % 2 == 0
7
8
9 print("is_even(2) =", is_even(2))
10 print("is_even(7) =", is_even(7))
11 print("is_even(0) =", is_even(0))
12 print("is_even(-4) =", is_even(-4))
13 print("is_even(9) =", is_even(9))
14 print("is_even('a') =", is_even("a"))
15
```

Output

```
ERROR!
is_even(2) = True
is_even(7) = False
is_even(0) = True
is_even(-4) = True
is_even(9) = False
is_even('a') = Error: Input must be an integer

== Code Execution Successful ==
```

## Task 2 – String Case Converter (TDD)

### Implementation:

```
def to_uppercase(text):
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return text.upper()

def to_lowercase(text):
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return text.lower()
```

### Expected Output:

```
to_uppercase("ai coding") → "AI CODING"
to_lowercase("TEST") → "test"
to_uppercase("") → "" to_lowercase(None)
→ TypeError
```

The screenshot shows a Jupyter Notebook interface with a sidebar of icons for different languages (Python, R, SQL, etc.). The main area has tabs for 'main.py' and 'Output'. The code in 'main.py' is a string case converter. The 'Output' tab shows the execution results, including error messages for non-string inputs and successful outputs for strings.

```
1 # Task 2 - String Case Converter
2
3 def to_uppercase(text):
4     if not isinstance(text, str):
5         return "Error: Input must be a string"
6     return text.upper()
7
8 def to_lowercase(text):
9     if not isinstance(text, str):
10        return "Error: Input must be a string"
11    return text.lower()
12
13
14 print('to_uppercase("ai coding") =', to_uppercase("ai coding"))
15 print('to_lowercase("TEST") =', to_lowercase("TEST"))
16 print('to_uppercase("") =', to_uppercase(""))
17 print('to_lowercase(None) =', to_lowercase(None))
18
```

Output:

```
ERROR!
to_uppercase("ai coding") = AI CODING
to_lowercase("TEST") = test
to_uppercase("") =
to_lowercase(None) = Error: Input must be a string

== Code Execution Successful ==
```

### Task 3 – List Sum Calculator (TDD)

#### Implementation:

```
def sum_list(numbers):
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")
    total = 0
    for item in numbers:
        if isinstance(item, (int, float)):
            total += item
    return total
```

#### Expected Output:

```
sum_list([1,2,3]) → 6
sum_list([]) → 0
sum_list([-1,5,-4]) → 0
sum_list([2,"a",3]) → 5
```

```

main.py | Run | Output
1 # Task 3 - List Sum Calculator
2
3 def sum_list(numbers):
4     if not isinstance(numbers, list):
5         return "Error: Input must be a list"
6
7     total = 0
8     for item in numbers:
9         if isinstance(item, (int, float)):
10             total += item
11     return total
12
13
14 print("sum_list([1,2,3]) =", sum_list([1,2,3]))
15 print("sum_list([]) =", sum_list([]))
16 print("sum_list([-1.5,-4]) =", sum_list([-1.5,-4]))
17 print('sum_list([2,"a",3]) =', sum_list([2,"a",3]))
18

```

Output:

```

sum_list([1,2,3]) = 6
sum_list([]) = 0
sum_list([-1.5,-4]) = 0
sum_list([2,"a",3]) = 5

== Code Execution Successful ==

```

## Task 4 – StudentResult Class (TDD)

### Implementation:

```

class StudentResult:
    def __init__(self):
        self.marks = []

    def add_marks(self, mark):
        if not isinstance(mark, (int, float)):
            raise TypeError("Mark must be numeric")
        if mark < 0 or mark > 100:
            raise ValueError("Mark must be between 0 and 100")
        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):
        avg = self.calculate_average()
        return "Pass" if avg >= 40 else "Fail"

```

### Expected Output:

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

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

Marks: [-10] → ValueError

```
main.py
5-     def __init__(self):
6-         self.marks = []
7-
8-     def add_marks(self, mark):
9-         if not isinstance(mark, (int, float)):
10-             return "Error: Mark must be numeric"
11-         if mark < 0 or mark > 100:
12-             return "Error: Mark must be between 0 and 100"
13-         self.marks.append(mark)
14-
15-     def calculate_average(self):
16-         if not self.marks:
17-             return 0
18-         return sum(self.marks) / len(self.marks)
19-
20-     def get_result(self):
21-         average = self.calculate_average()
22-         return "Pass" if average >= 40 else "Fail"
23-
24-
25- # Testing
26- s1 = StudentResult()
27- s1.add_marks(60)
28- s1.add_marks(70)
29- s1.add_marks(80)
```

Output

```
Marks: [60,70,80]
Average = 70.0
Result = Pass
ERROR!

Marks: [30,35,40]
Average = 35.0
Result = Fail

Adding invalid mark (-10): Error: Mark must be between 0 and 100
== Code Execution Successful ==
```

## Task 5 – Username Validator (TDD)

### Implementation:

```
def is_valid_username(username):
    if not isinstance(username, str):
        return False
    if len(username) < 5:
        return False
    if " " in username:
        return False
    if not username.isalnum():
        return False
    return True
```

### Expected Output:

```
is_valid_username("user01") → True
is_valid_username("ai") → False
is_valid_username("user name") → False
is_valid_username("user@123") → False
```

The screenshot shows a Python code editor interface with the following details:

- File:** main.py
- Code (Content):**

```
1 # Task 5 - Username Validator
2
3 def is_valid_username(username):
4     if not isinstance(username, str):
5         return False
6     if len(username) < 5:
7         return False
8     if " " in username:
9         return False
10    if not username.isalnum():
11        return False
12    return True
13
14
15 print('is_valid_username("user01") =', is_valid_username("user01"))
16 print('is_valid_username("ai") =', is_valid_username("ai"))
17 print('is_valid_username("user name") =', is_valid_username("user
18 name"))
19 print('is_valid_username("user@123") =', is_valid_username("user@123"
))
```

- Output (Results):**

```
is_valid_username("user01") = True
is_valid_username("ai") = False
is_valid_username("user name") = False
is_valid_username("user@123") = False
== Code Execution Successful ==
```

## Conclusion:

In this lab, Test-Driven Development (TDD) was implemented using AI-generated test cases. Each function was written only after defining expected test behavior. This approach ensures reliable, validated, and clean code development.