# 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:**

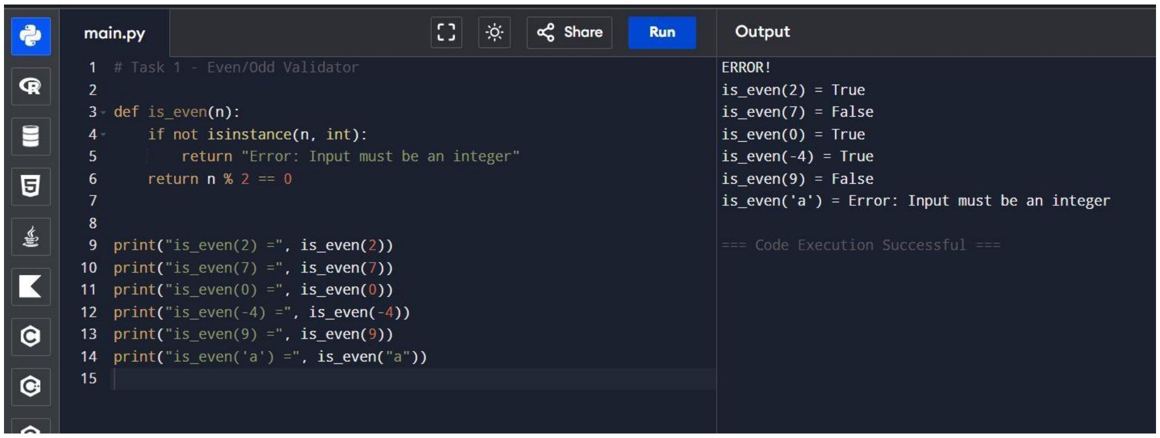
```
import unittest

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.



```
# Task 1 - Even/Odd Validator

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


print("is_even(2) =", is_even(2))
print("is_even(7) =", is_even(7))
print("is_even(0) =", is_even(0))
print("is_even(-4) =", is_even(-4))
print("is_even(9) =", is_even(9))
print("is_even('a') =", is_even("a"))
```

```
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 ===
```

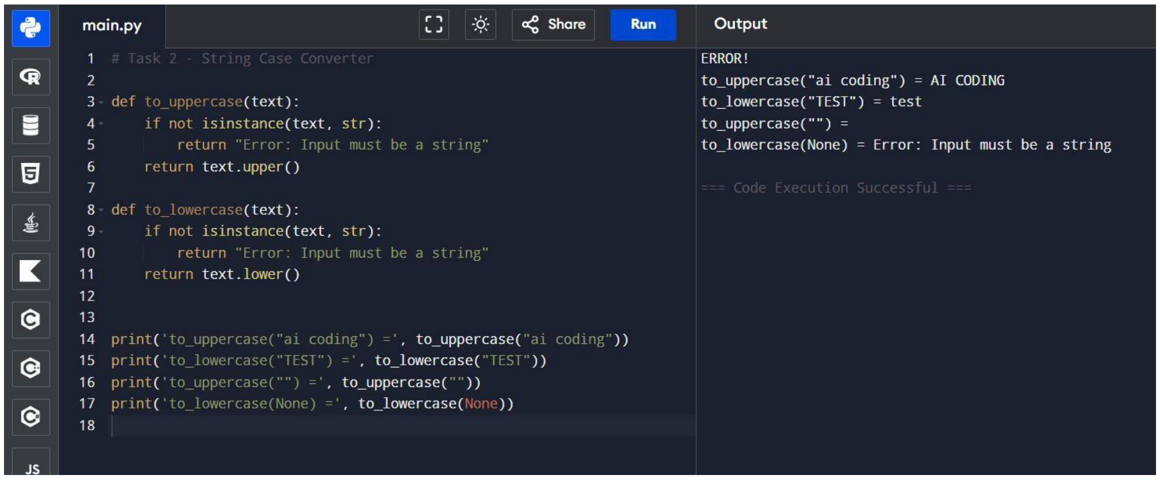
## Task 2 – String Case Converter (TDD)

**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



```python
# Task 2 - String Case Converter

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

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


print('to_uppercase("ai coding") =', to_uppercase("ai coding"))
print('to_lowercase("TEST") =', to_lowercase("TEST"))
print('to_uppercase("") =', to_uppercase(""))
print('to_lowercase(None) =', to_lowercase(None))
```

```
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 ===
```

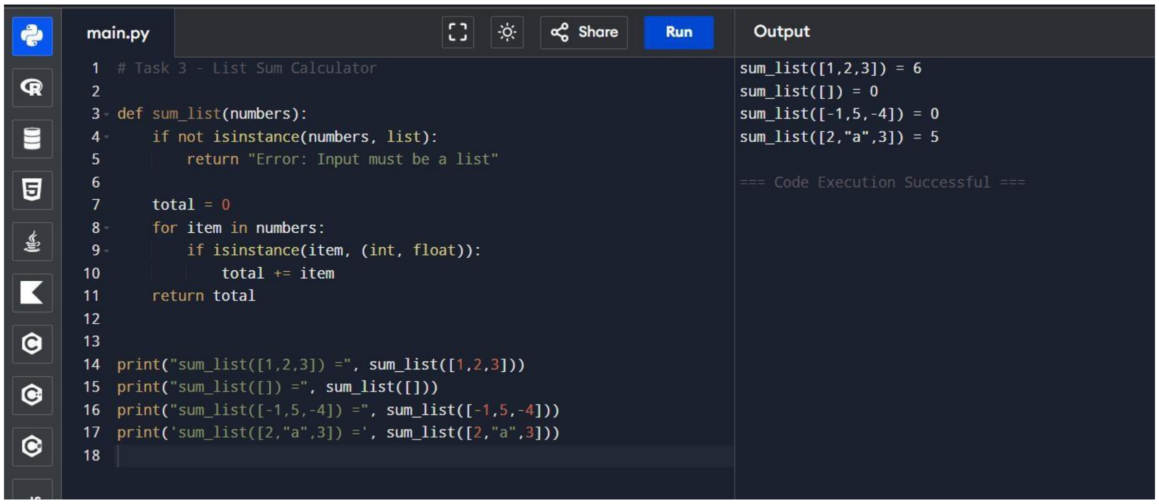
## Task 3 – List Sum Calculator (TDD)

**Implementation:**

```python
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
```



```
# Task 3 - List Sum Calculator

def sum_list(numbers):
    if not isinstance(numbers, list):
        return "Error: Input must be a list"

    total = 0
    for item in numbers:
        if isinstance(item, (int, float)):
            total += item
    return total


print("sum_list([1,2,3]) =", sum_list([1,2,3]))
print("sum_list([]) =", sum_list([]))
print("sum_list([-1,5,-4]) =", sum_list([-1,5,-4]))
print('sum_list([2,"a",3]) =', sum_list([2,"a",3]))
```

```
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)

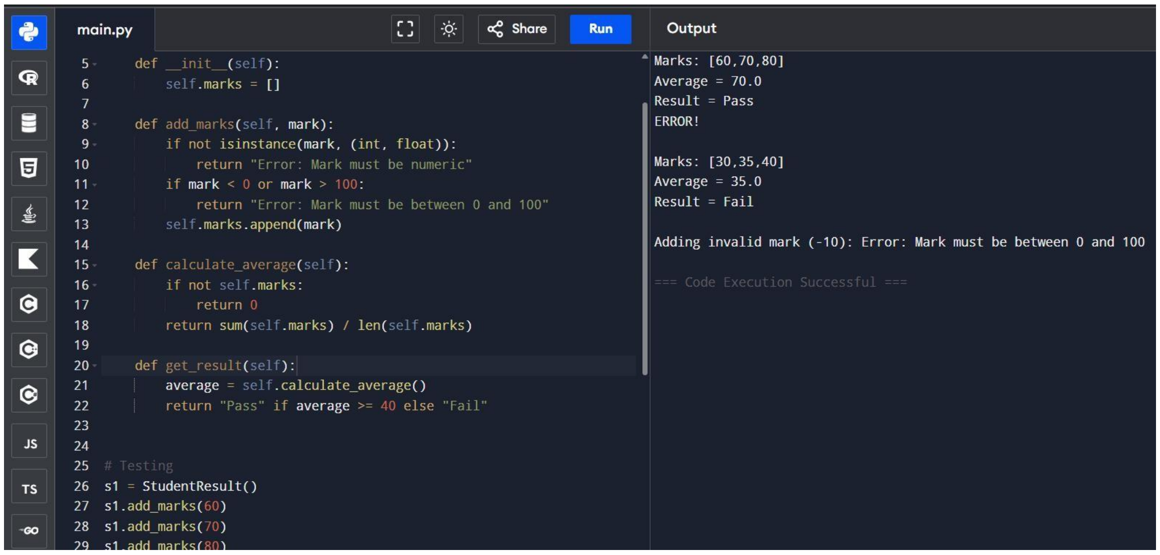
**Implementation:**

```
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



```
    def __init__(self):
        self.marks = []

    def add_marks(self, mark):
        if not isinstance(mark, (int, float)):
            return "Error: Mark must be numeric"
        if mark < 0 or mark > 100:
            return "Error: 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):
        average = self.calculate_average()
        return "Pass" if average >= 40 else "Fail"


# Testing
s1 = StudentResult()
s1.add_marks(60)
s1.add_marks(70)
s1.add_marks(80)
```

```
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 ===
```

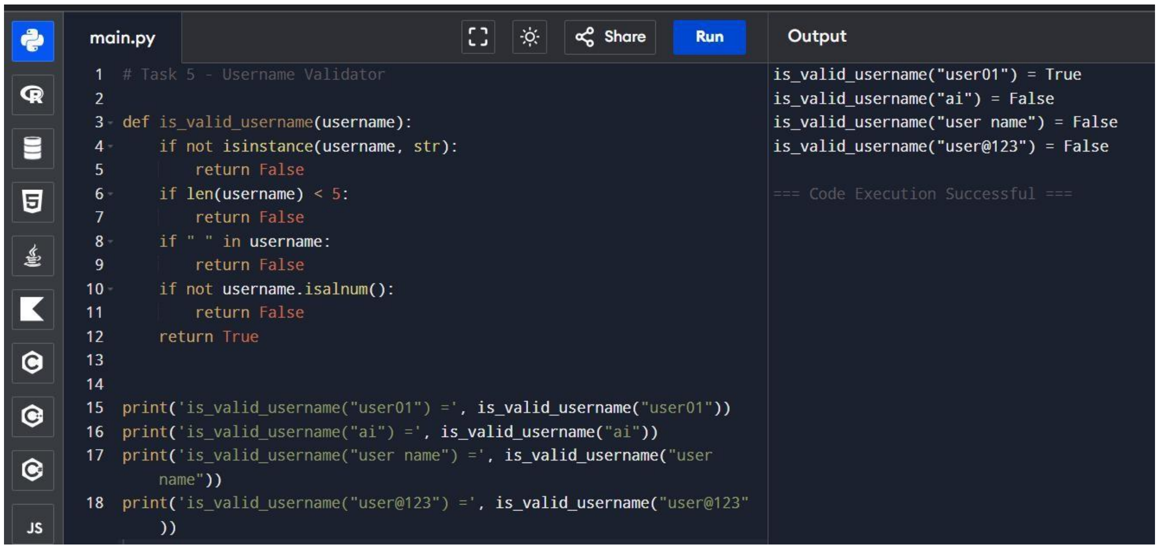
## Task 5 – Username Validator (TDD)

**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



```python
# Task 5 - Username Validator

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


print('is_valid_username("user01") =', is_valid_username("user01"))
print('is_valid_username("ai") =', is_valid_username("ai"))
print('is_valid_username("user name") =', is_valid_username("user
    name"))
print('is_valid_username("user@123") =', is_valid_username("user@123"
    ))
```

```
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.