AI ASSISTED CODING

Sai Thrishool **2303A51127**

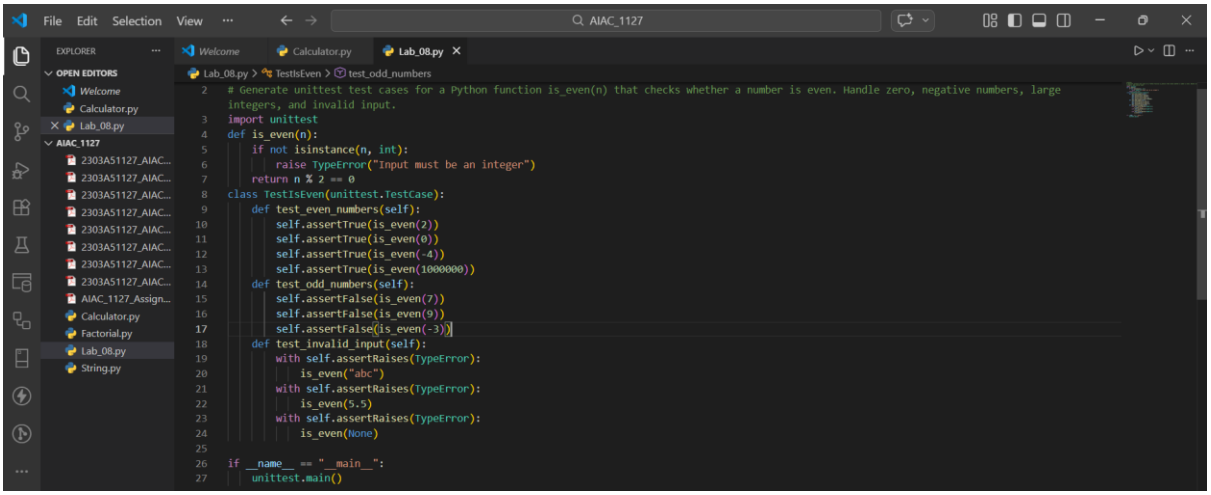
BACTH – 03 **17 – 02 – 2026**

ASSIGNMENT – 08

LAB – 08 : Test – Driven Development with AI – Generating and Working with Test Cases.

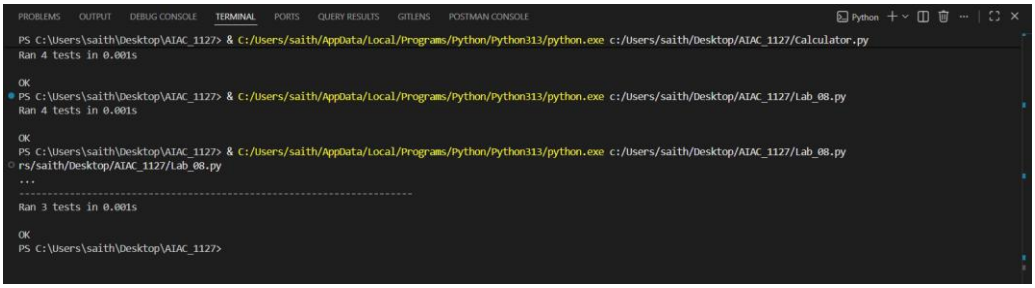
Task – 01 : Test – Driven Development for Odd/Even Number Validator.

Prompt : Generate unittest test cases for a Python function is_even(n) that checks whether a number is even. Handle zero, negative numbers, large integers, and invalid input. **Code :**



```python
# Generate unittest test cases for a Python function is_even(n) that checks whether a number is even. Handle zero, negative numbers, large integers, and invalid input.
import unittest
def is_even(n):
    if not isinstance(n, int):
        raise TypeError("Input must be an integer")
    return n % 2 == 0
class TestIsEven(unittest.TestCase):
    def test_even_numbers(self):
        self.assertTrue(is_even(2))
        self.assertTrue(is_even(0))
        self.assertTrue(is_even(-4))
        self.assertTrue(is_even(1000000))
    def test_odd_numbers(self):
        self.assertFalse(is_even(7))
        self.assertFalse(is_even(9))
        self.assertFalse(is_even(-3))
    def test_invalid_input(self):
        with self.assertRaises(TypeError):
            is_even("abc")
        with self.assertRaises(TypeError):
            is_even(5.5)
        with self.assertRaises(TypeError):
            is_even(None)

if __name__ == "__main__":
    unittest.main()
```

Output :

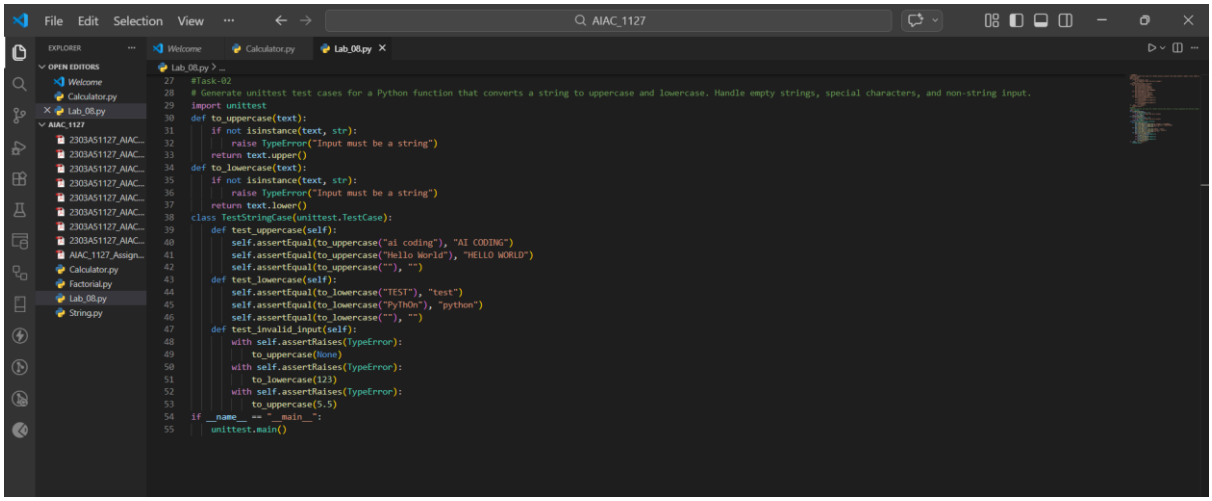


```
PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Calculator.py
Ran 4 tests in 0.001s

OK
PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Lab_08.py
Ran 4 tests in 0.001s

OK
PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Lab_08.py
rs/saith/Desktop/AIAC_1127/Lab_08.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
PS C:\Users\saith\Desktop\AIAC_1127>
```

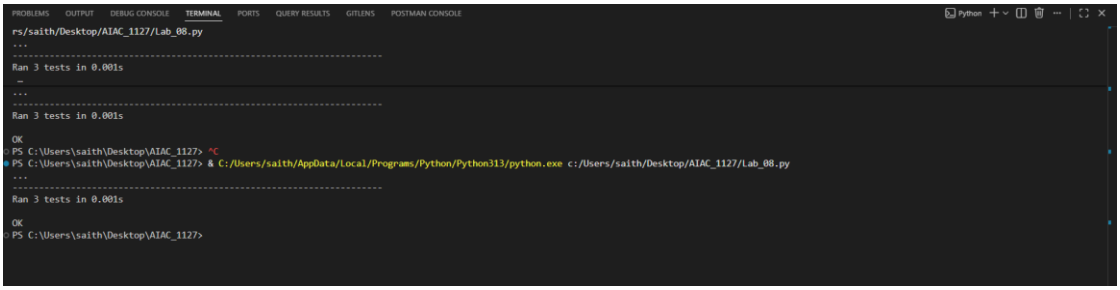
Explanation :

The function first validates that the input is an integer. It then checks divisibility by 2 using modulus operator. It handles zero, negative, and large integers correctly.

Task – 02 : Test – Driven Development for String Case Converter.
Prompt : Generate test cases for to_uppercase(text) and to_lowercase(text) handling empty strings, mixed-case input, and invalid inputs. **Code :**



Output:



Explanation :
Both functions validate input type and use built-in string methods .upper() and .lower() for conversion.
Task – 03 : Test – Driven Development for List sum Calculator.
Prompt : Generate test cases for sum_list(numbers) that handles empty lists, negative numbers, and ignores non-numeric values.
Code :

```python
#Task-03
#Generate test cases for sum_list(numbers) that calculates the sum of list elements, but handle non-numeric values in the list by ignoring them.
import unittest
def sum_list(numbers):
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")
    total = 0
    for num in numbers:
        if isinstance(num, (int, float)):
            total += num
    return total
class TestSumList(unittest.TestCase):
    def test_normal_list(self):
        self.assertEqual(sum_list([1, 2, 3]), 6)
    def test_empty_list(self):
        self.assertEqual(sum_list([]), 0)
    def test_negative_numbers(self):
        self.assertEqual(sum_list([-1, 5, -4]), 0)
    def test_mixed_values(self):
        self.assertEqual(sum_list([2, "a", 3]), 5)

    def test_invalid_input(self):
        with self.assertRaises(TypeError):
            sum_list("123")

if __name__ == "__main__":
    unittest.main()
```

Output :

```
PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Calculator.py
Ran 4 tests in 0.001s

OK
PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Lab_08.py
Ran 4 tests in 0.001s

OK
PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Lab_08.py
rs/saith/Desktop/AIAC_1127/Lab_08.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
PS C:\Users\saith\Desktop\AIAC_1127>
```

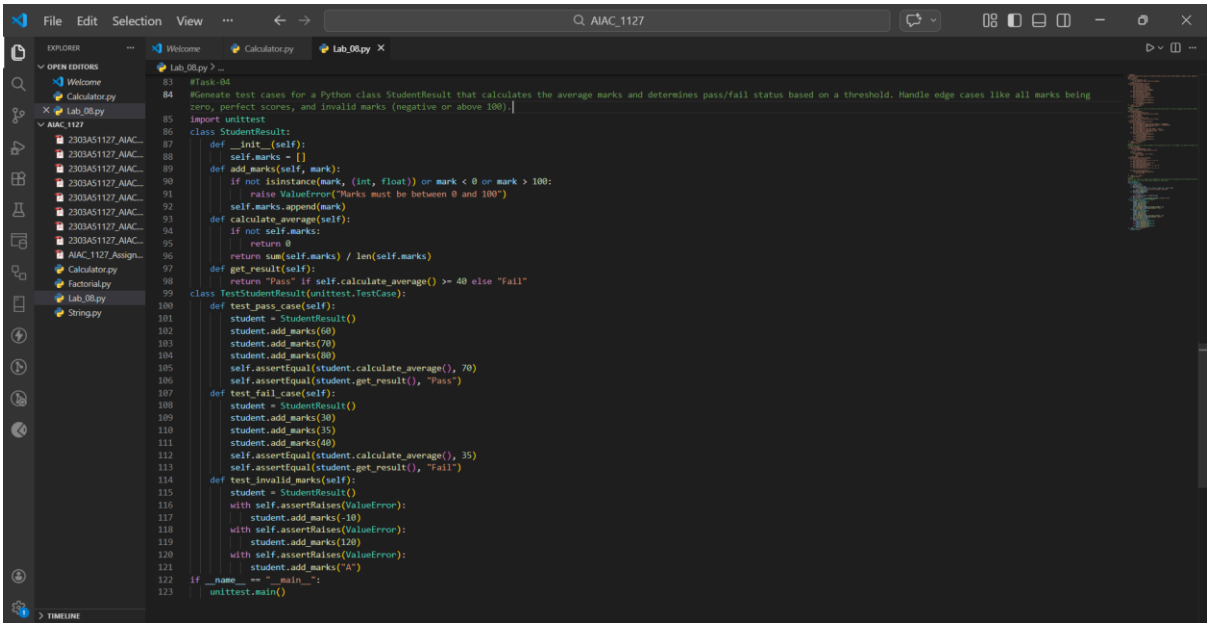
Explanation :

The function iterates through the list and adds only numeric values. It safely ignores non-numeric elements and returns 0 for empty lists.

Task – 04 : Test Cases for Student Result Class.

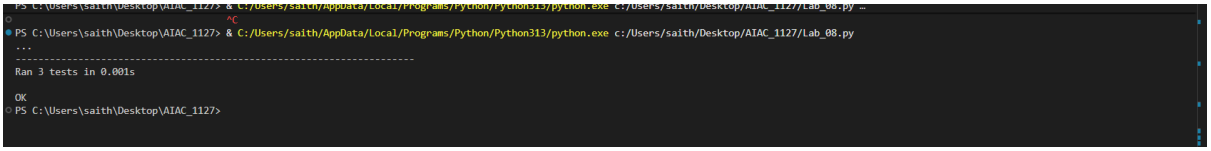
Prompt : Generate test cases for a StudentResult class with methods: add_marks, calculate_average, get_result. Marks must be between 0 and 100.

Code :



```
#Task-04
#Geneate test cases for a Python class StudentResult that calculates the average marks and determines pass/fail status based on a threshold. Handle edge cases like all marks being
zero, perfect scores, and invalid marks (negative or above 100).
import unittest
class StudentResult:
    def __init__(self):
        self.marks = []
    def add_marks(self, mark):
        if not isinstance(mark, (int, float)) or mark < 0 or mark > 100:
            raise ValueError("Marks 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):
        return "Pass" if self.calculate_average() >= 40 else "Fail"
class TestStudentResult(unittest.TestCase):
    def test_pass_case(self):
        student = StudentResult()
        student.add_marks(60)
        student.add_marks(70)
        student.add_marks(80)
        self.assertEqual(student.calculate_average(), 70)
        self.assertEqual(student.get_result(), "Pass")
    def test_fail_case(self):
        student = StudentResult()
        student.add_marks(30)
        student.add_marks(35)
        student.add_marks(40)
        self.assertEqual(student.calculate_average(), 35)
        self.assertEqual(student.get_result(), "Fail")
    def test_invalid_marks(self):
        student = StudentResult()
        with self.assertRaises(ValueError):
            student.add_marks(-10)
        with self.assertRaises(ValueError):
            student.add_marks(120)
        with self.assertRaises(ValueError):
            student.add_marks("A")
if __name__ == "__main__":
    unittest.main()
```

Output:



```
^C
PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Lab_08.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
PS C:\Users\saith\Desktop\AIAC_1127>
```

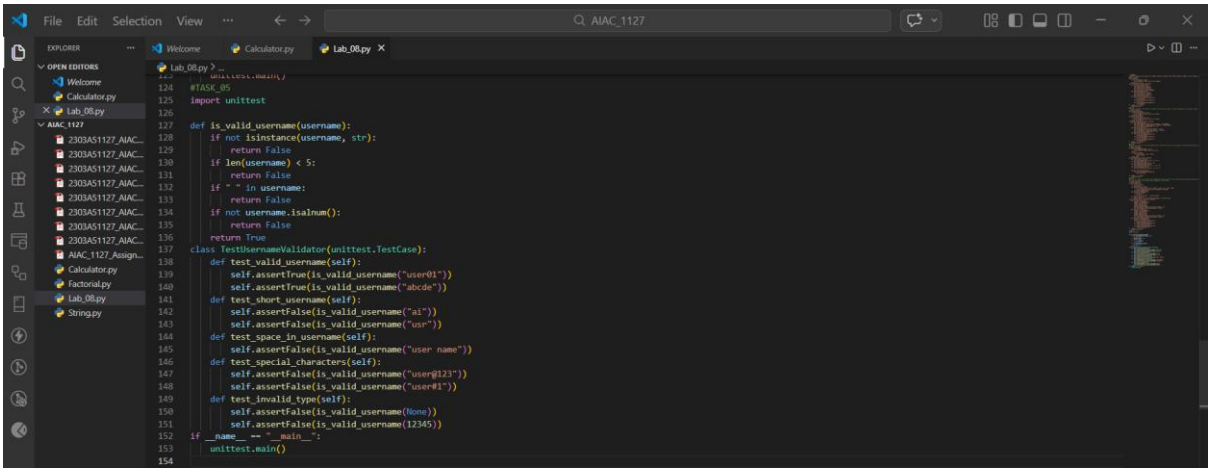
Explanation :

The class validates marks before storing them. It calculates average dynamically and determines result based on 40% threshold.

Task – 05 : Test – Driven Development for username Validator.

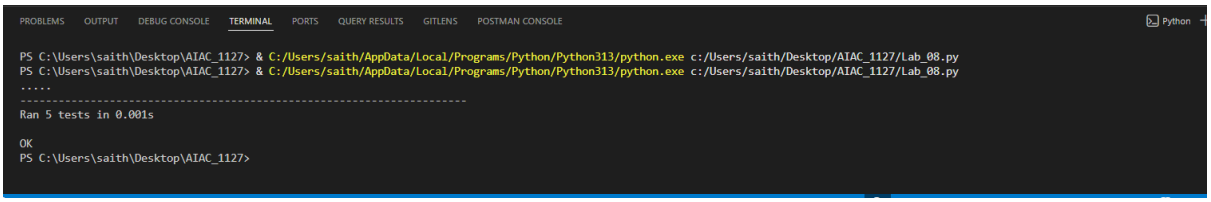
Prompt : Generate test cases for is_valid_username(username) with minimum 5 characters, no spaces, and only alphanumeric characters.

Code :



```python
#TASK_05
import unittest

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
class TestUsernameValidator(unittest.TestCase):
    def test_valid_username(self):
        self.assertTrue(is_valid_username("user01"))
        self.assertTrue(is_valid_username("abcde"))
    def test_short_username(self):
        self.assertFalse(is_valid_username("ai"))
        self.assertFalse(is_valid_username("usr"))
    def test_space_in_username(self):
        self.assertFalse(is_valid_username("user name"))
    def test_special_characters(self):
        self.assertFalse(is_valid_username("user@123"))
        self.assertFalse(is_valid_username("user#1"))
    def test_invalid_type(self):
        self.assertFalse(is_valid_username(None))
        self.assertFalse(is_valid_username(12345))
if __name__ == "__main__":
    unittest.main()
```

Output :



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  QUERY RESULTS  GITLENS  POSTMAN CONSOLE

PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Lab_08.py
PS C:\Users\saith\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/Desktop/AIAC_1127/Lab_08.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
PS C:\Users\saith\Desktop\AIAC_1127>
```

Explanation :

The function checks length, space restriction, and alphanumeric condition using built-in string validation methods.