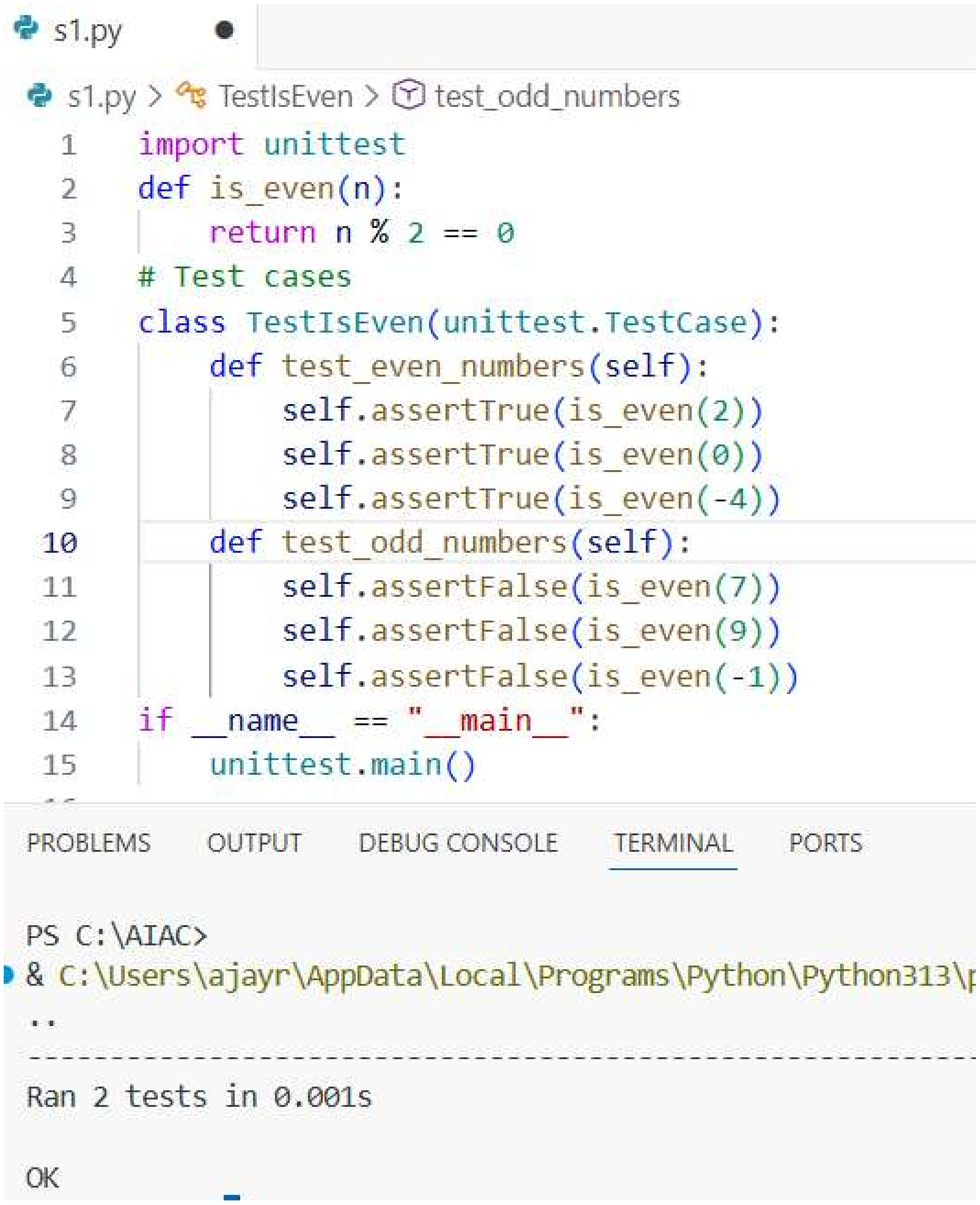
AI - Assisted Coding-Assignment -8

HTNO:2303A52228
Batch:37

Lab 8: Test-Driven Development with AI-Generating and Working with Test Cases.

Task Description-1
Test-Driven Development for Even/Odd Number Validator
Expected Output:
A correctly implemented is_even() function that passes all AI-
generated test cases
Prompt:
Write a Python function is_even(n) that returns True if the given number is even and False if it is odd.
CODE AND OUTPUT:



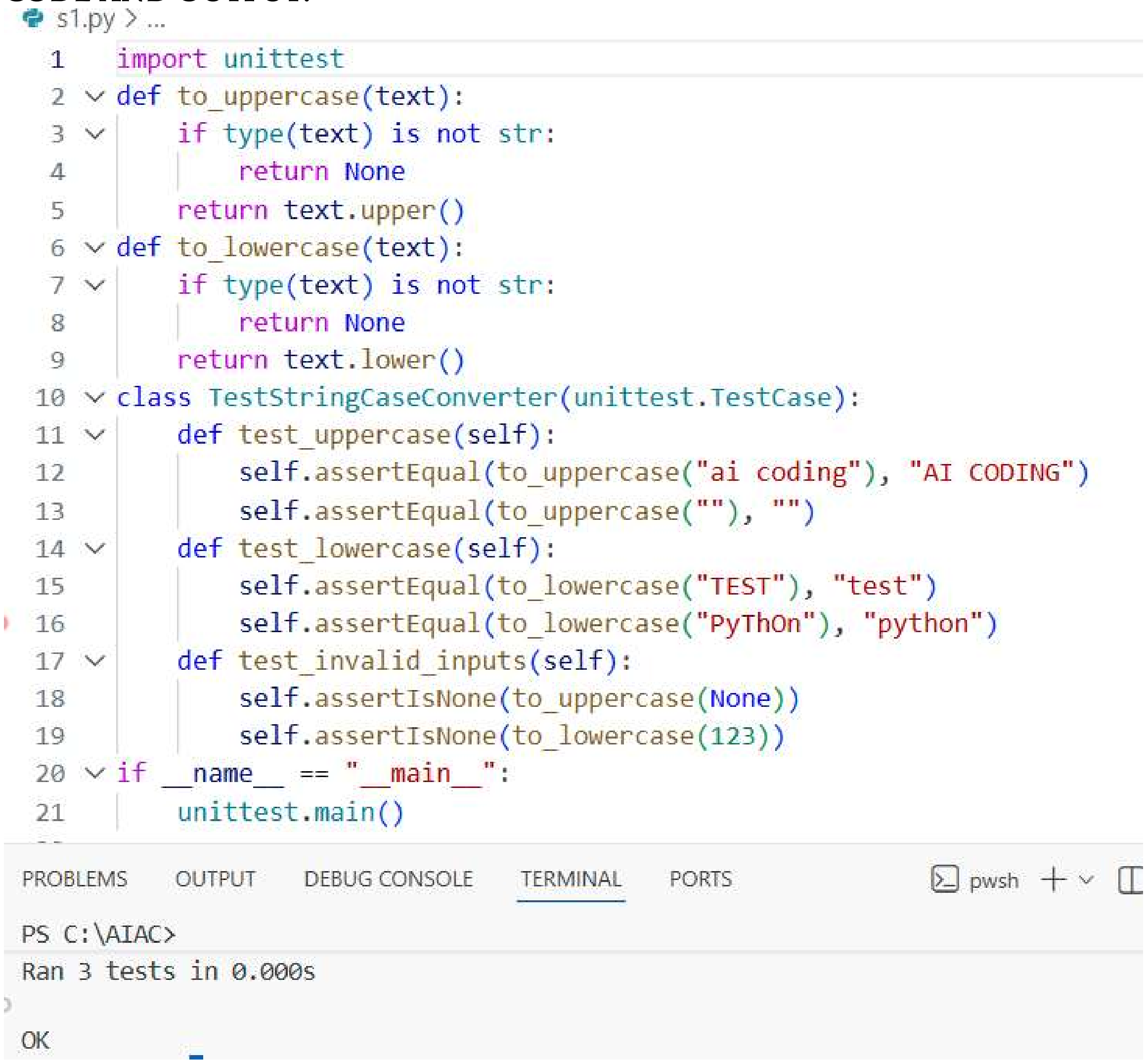
```python
import unittest
def is_even(n):
    return n % 2 == 0
# Test cases
class TestIsEven(unittest.TestCase):
    def test_even_numbers(self):
        self.assertTrue(is_even(2))
        self.assertTrue(is_even(0))
        self.assertTrue(is_even(-4))
    def test_odd_numbers(self):
        self.assertFalse(is_even(7))
        self.assertFalse(is_even(9))
        self.assertFalse(is_even(-1))
if __name__ == "__main__":
    unittest.main()
```

```
PS C:\AIAC>
& C:\Users\ajayr\AppData\Local\Programs\Python\Python313\p
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Explanation:
In this task, test cases for the is_even(n) function are created first using AI, following the Test-Driven Development approach. These tests describe how the function should behave for even, odd, zero, and negative numbers. After defining the tests, the function is implemented to meet all test expectations. The unittest framework is used to run the tests automatically, and when all tests pass, it confirms that the function works correctly and reliably.

Task Description-2
Test-Driven Development for String Case Converter

Expected Output:
Two string conversion functions that pass all AI-generated test cases with safe input handling.
Prompt:
Generate test cases for two Python functions to_uppercase(text) and to_lowercase.The functions should convert strings to upper case and lower case respectively.The test cases must handle empty strings, mixed-case input, and invalid inputs such as numbers.
CODE AND OUTPUT:



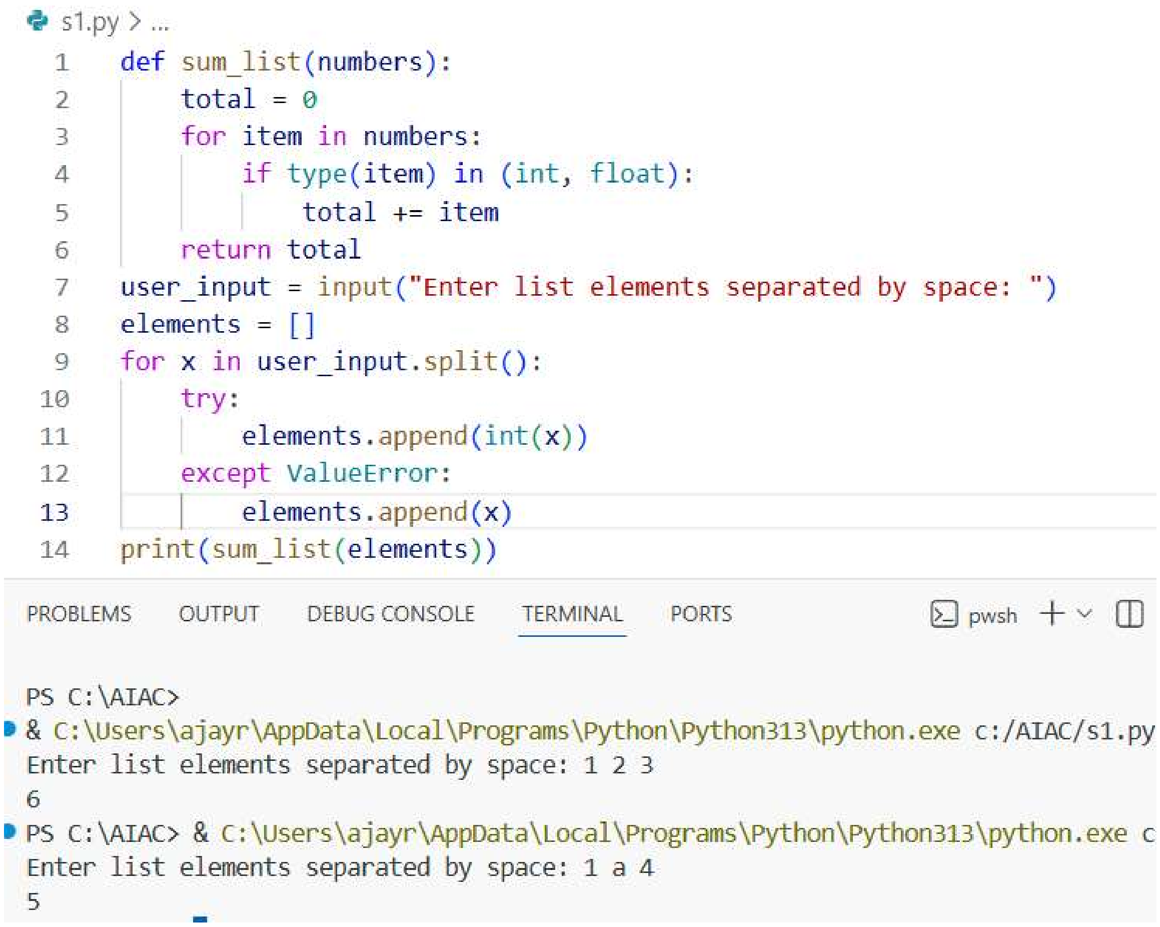
s1.py > ...

```python
import unittest
def to_uppercase(text):
    if type(text) is not str:
        return None
    return text.upper()
def to_lowercase(text):
    if type(text) is not str:
        return None
    return text.lower()
class TestStringCaseConverter(unittest.TestCase):
    def test_uppercase(self):
        self.assertEqual(to_uppercase("ai coding"), "AI CODING")
        self.assertEqual(to_uppercase(""), "")
    def test_lowercase(self):
        self.assertEqual(to_lowercase("TEST"), "test")
        self.assertEqual(to_lowercase("PyThOn"), "python")
    def test_invalid_inputs(self):
        self.assertIsNone(to_uppercase(None))
        self.assertIsNone(to_lowercase(123))
if __name__ == "__main__":
    unittest.main()
```

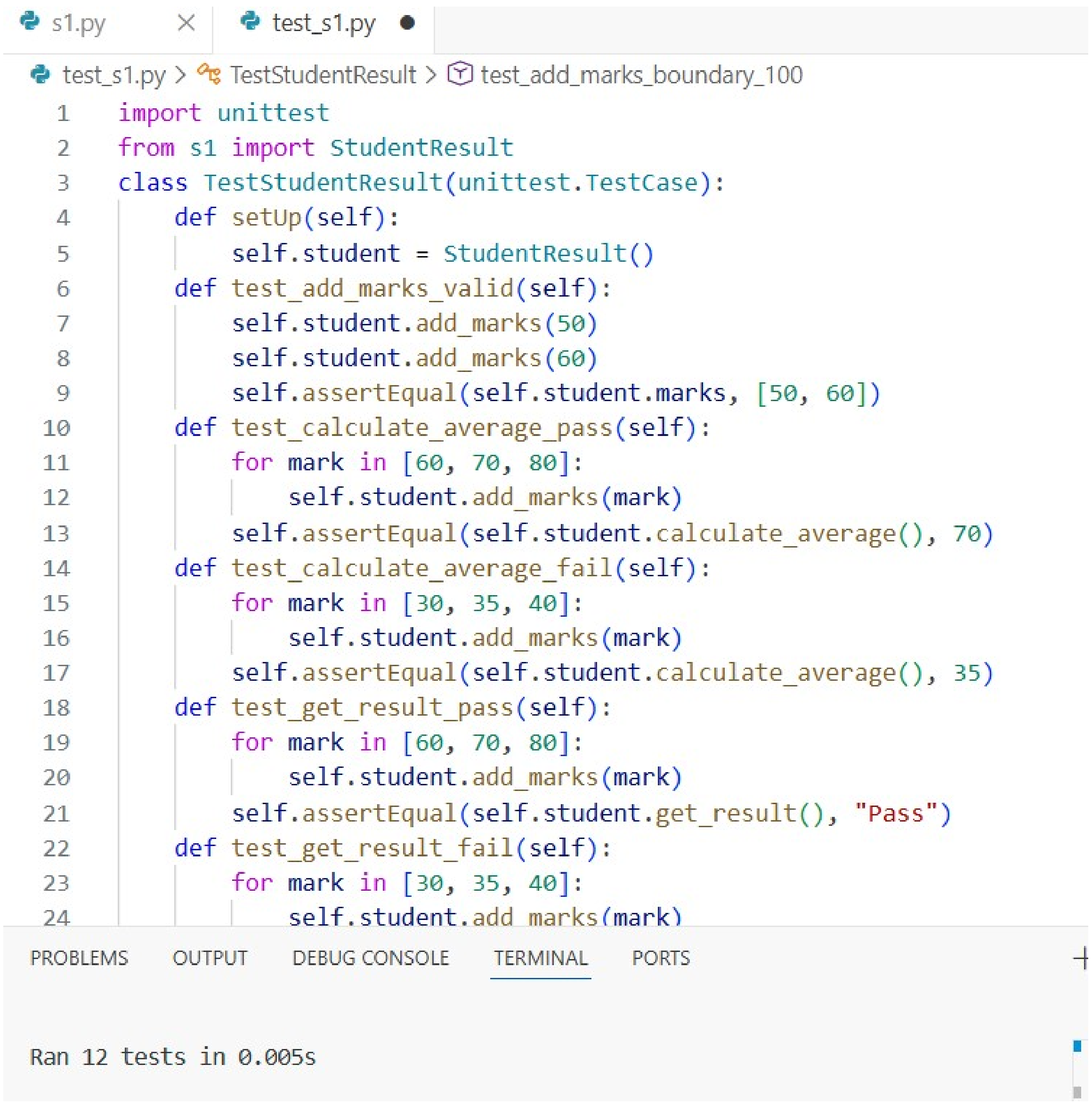
```
PS C:\AIAC>
Ran 3 tests in 0.000s

OK
```

Explanation:
AI-generated test cases were written first for the string case conversion functions. The functions were then implemented to pass all tests using unittest, ensuring correct behavior and safe handling of invalid inputs, following the Test-Driven Development approach.

Task Description-3
Test-Driven Development for List Sum Calculator
Expected Output:

A robust list-sum function validated using AI-generated test
cases.
Prompt:
Generate test cases for a Python function sum_list(numbers) that returns the sum of elements in a list.The function should handle empty lists, negative numbers, and safely ignore non-numeric values.
CODE AND OUTPUT:

```python
def sum_list(numbers):
    total = 0
    for item in numbers:
        if type(item) in (int, float):
            total += item
    return total
user_input = input("Enter list elements separated by space: ")
elements = []
for x in user_input.split():
    try:
        elements.append(int(x))
    except ValueError:
        elements.append(x)
print(sum_list(elements))
```
```
PS C:\AIAC>
& C:\Users\ajayr\AppData\Local\Programs\Python\Python313\python.exe c:/AIAC/s1.py
Enter list elements separated by space: 1 2 3
6
PS C:\AIAC> & C:\Users\ajayr\AppData\Local\Programs\Python\Python313\python.exe c
Enter list elements separated by space: 1 a 4
5
```
Explanation:
The program takes a list from the user, safely ignores non-numeric values, and returns the sum of valid numbers. It handles empty lists and negative numbers correctly.

Task Description-4
Test Cases for Student Result Class.
Expected Output:
A fully functional StudentResult class that passes all AI-
generated test.
Prompt:
Generate test cases for a Python class StudentResult with methods add_marks(mark), calculate_average(), and get_result().Marks should be between 0 and 100.
If average ≥ 40, result is Pass, otherwise Fail.Invalid marks should raise an error.
CODE AND OUTPUT:


s1.py | test_s1.py

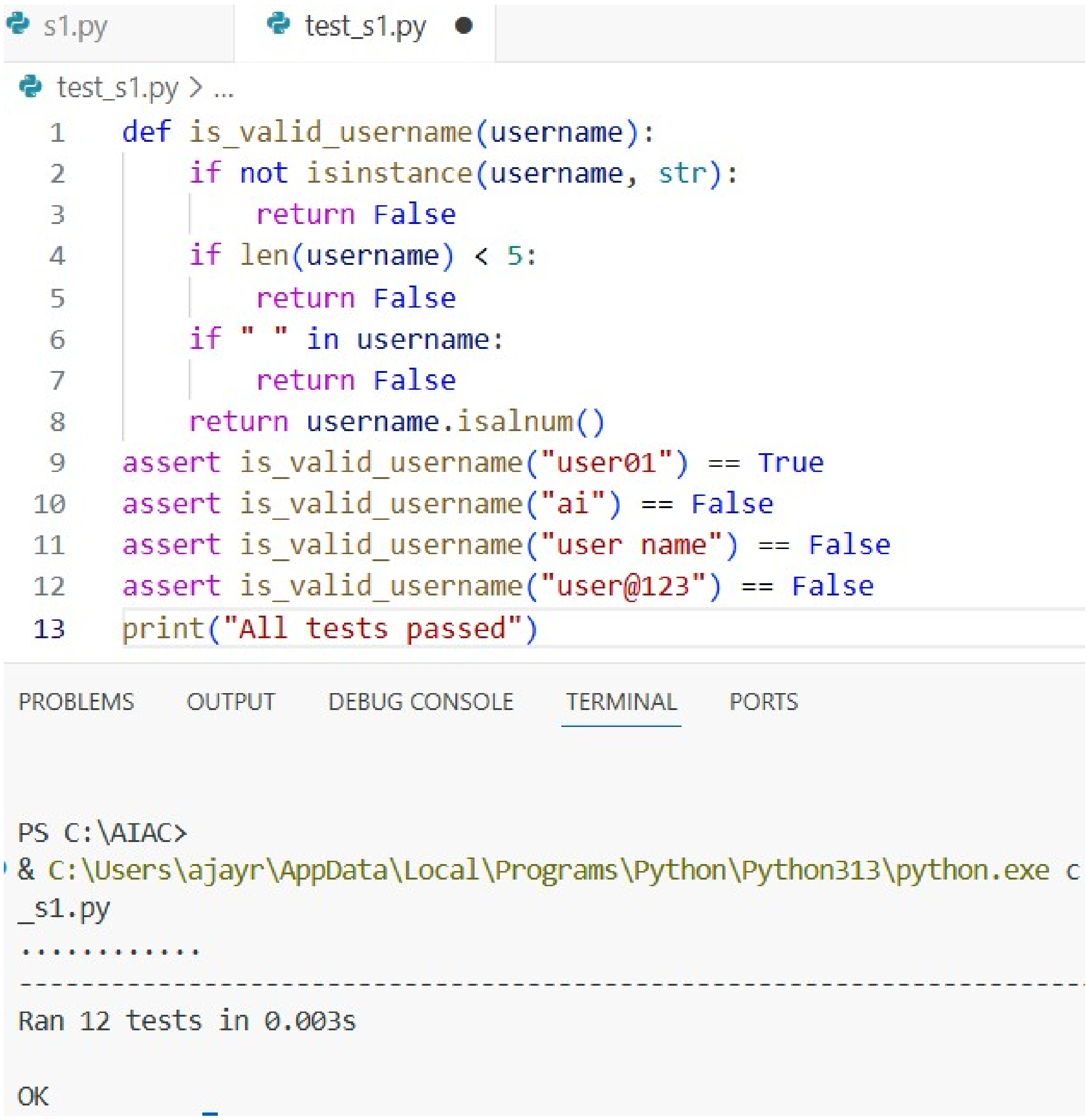
test_s1.py > TestStudentResult > test_add_marks_boundary_100

```python
import unittest
from s1 import StudentResult
class TestStudentResult(unittest.TestCase):
    def setUp(self):
        self.student = StudentResult()
    def test_add_marks_valid(self):
        self.student.add_marks(50)
        self.student.add_marks(60)
        self.assertEqual(self.student.marks, [50, 60])
    def test_calculate_average_pass(self):
        for mark in [60, 70, 80]:
            self.student.add_marks(mark)
        self.assertEqual(self.student.calculate_average(), 70)
    def test_calculate_average_fail(self):
        for mark in [30, 35, 40]:
            self.student.add_marks(mark)
        self.assertEqual(self.student.calculate_average(), 35)
    def test_get_result_pass(self):
        for mark in [60, 70, 80]:
            self.student.add_marks(mark)
        self.assertEqual(self.student.get_result(), "Pass")
    def test_get_result_fail(self):
        for mark in [30, 35, 40]:
            self.student.add_marks(mark)
```

```
Ran 12 tests in 0.005s
```

Explanation:
Tests are written first using assert. The class is built to pass those tests. Marks are validated, average is calculated, and result is decided as Pass or Fail.

Task Description-5
Test-Driven Development for Username Validator
Expected Output:
A username validation function that passes all AI-generated test cases.
Prompt:
Generate test cases for a Python function is_valid_username(username) with rules: minimum 5 characters, no spaces, only alphanumeric characters.Include both valid and invalid cases.
CODE AND OUTPUT:



```python
def is_valid_username(username):
    if not isinstance(username, str):
        return False
    if len(username) < 5:
        return False
    if " " in username:
        return False
    return username.isalnum()
assert is_valid_username("user01") == True
assert is_valid_username("ai") == False
assert is_valid_username("user name") == False
assert is_valid_username("user@123") == False
print("All tests passed")
```

```
PS C:\AIAC>
& C:\Users\ajayr\AppData\Local\Programs\Python\Python313\python.exe c
_s1.py
............
----------------------------------------------------------------------
Ran 12 tests in 0.003s

OK
```

Explanation:
First, AI helped create test cases that clearly define what a valid and invalid username looks like.Then the function was written to satisfy those tests by checking length, spaces, and characters.If all rules are followed, it returns True; otherwise, False.