ASSIGNMENT 8.2

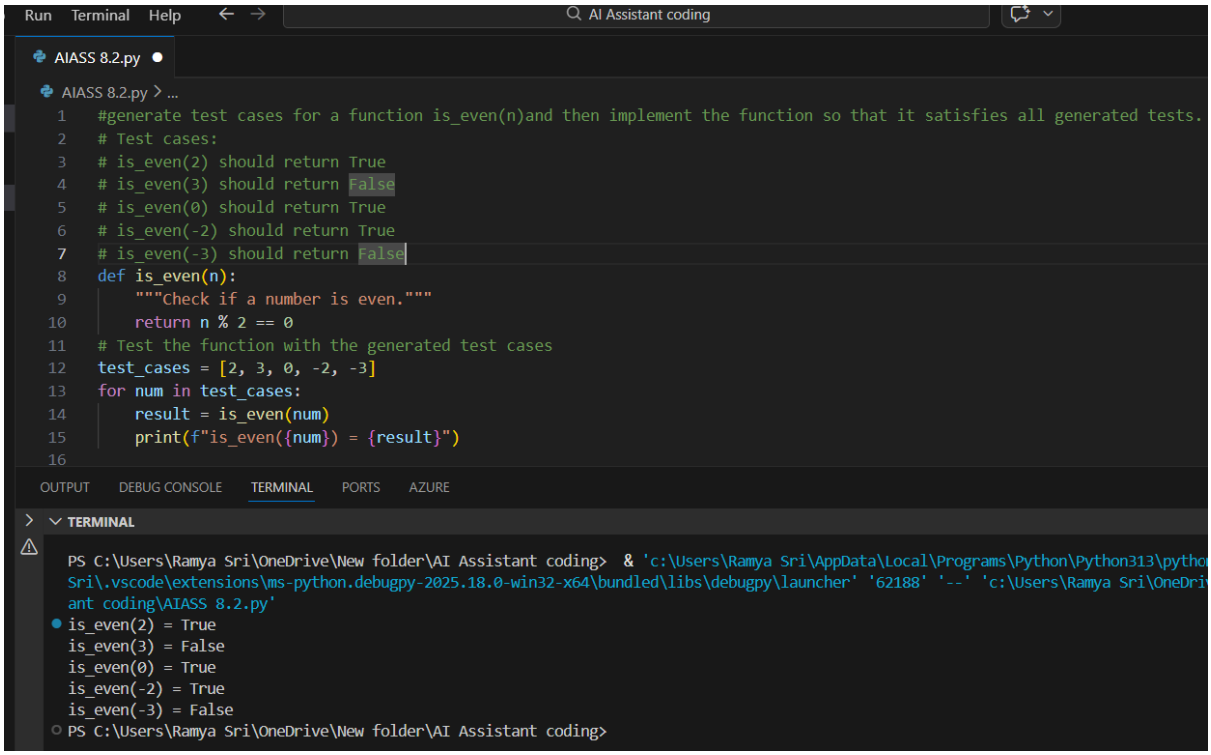
A.RAMYASRI

2303A510i9

BATCH:30

TASK 1: Test-Driven Development for Even/Odd Number Validator

CODE & OUTPUT:



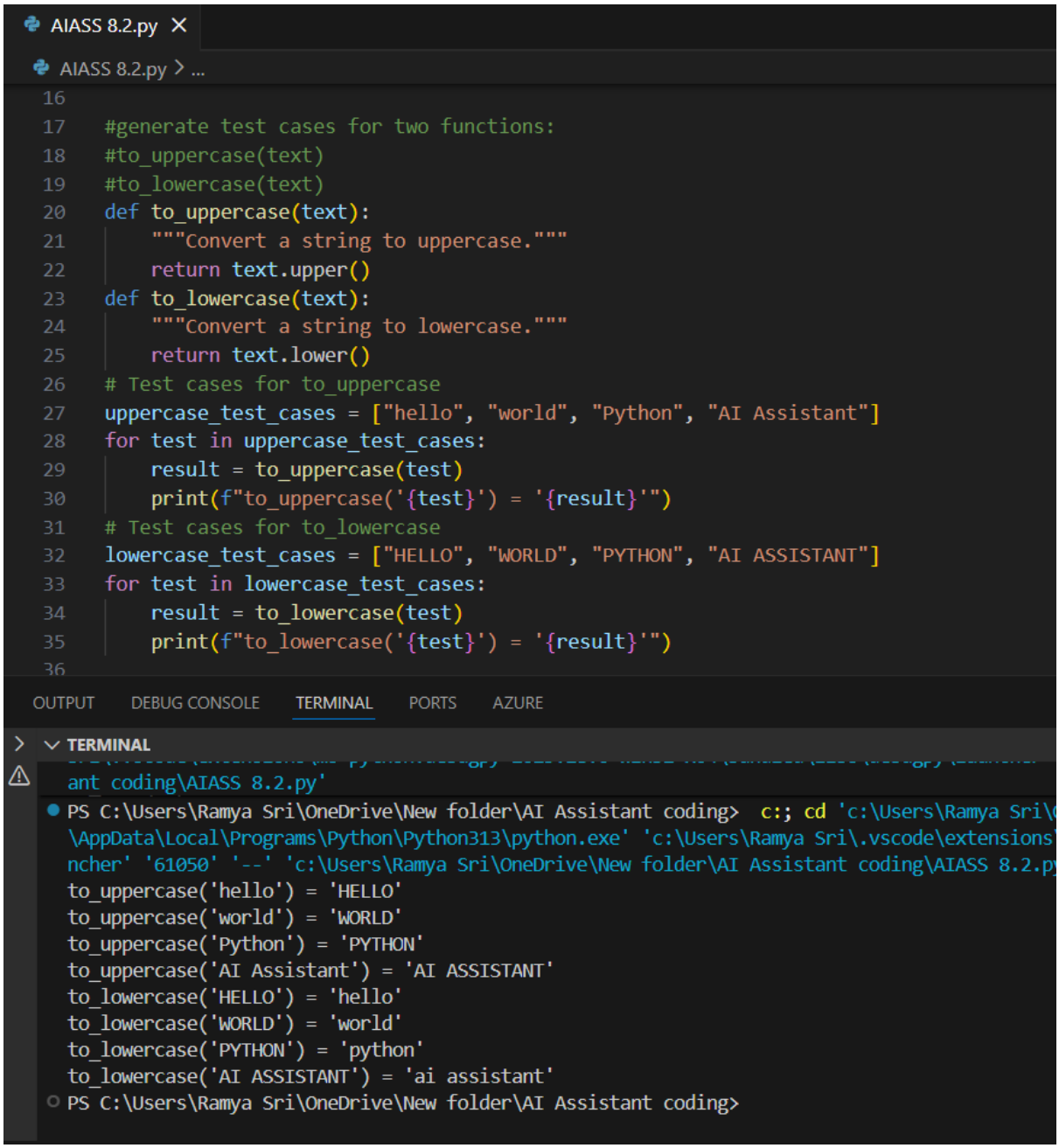
JUSTIFICATION:

Even/Odd Number Validator

The test cases ensure correctness across edge cases like zero, negatives, and large integers. By defining tests first, the is_even() function is forced to handle all scenarios without assumptions. This prevents logical gaps and guarantees robustness in mathematical validation.

TASK2: Test-Driven Development for String Case Converter

CODE & OUTPUT:



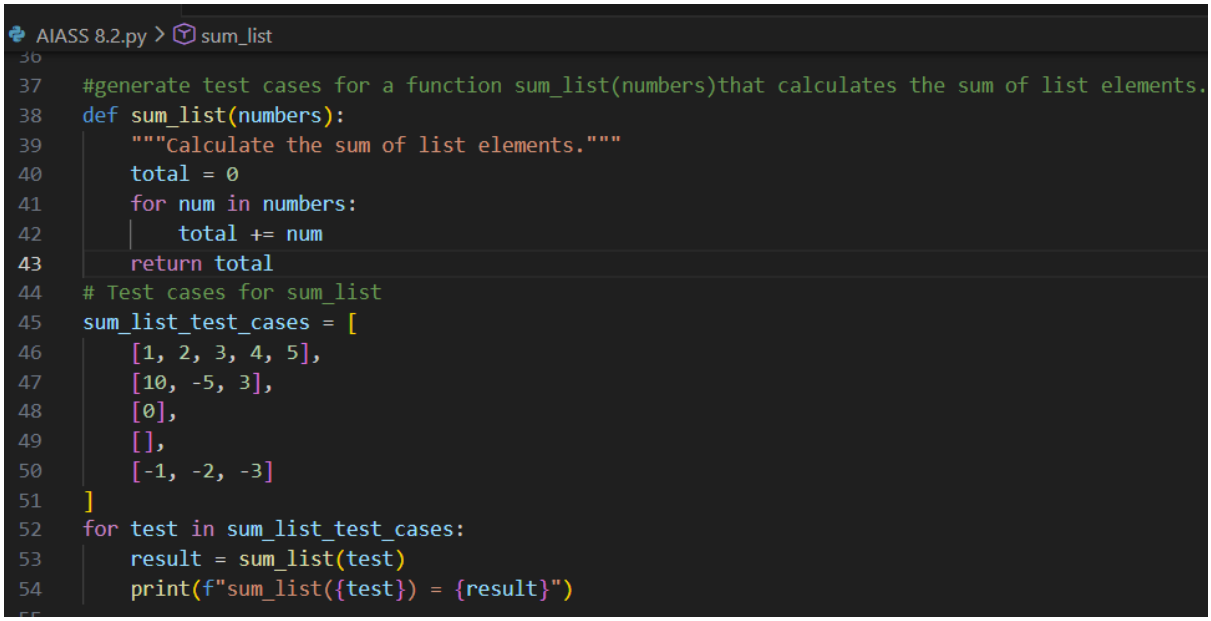
JUSTIFICATION:

String Case Converter

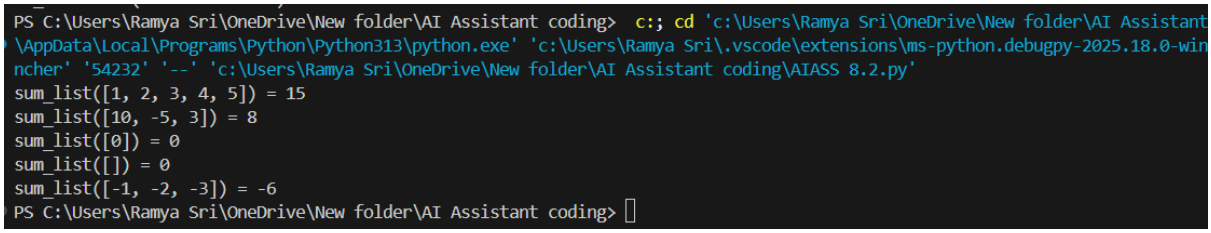
Test cases cover empty strings, mixed-case input, and invalid inputs like numbers or None. Writing tests upfront ensures the functions handle both normal and exceptional cases safely. This enforces defensive programming and avoids runtime errors in real-world usage.

TASK 3: Test-Driven Development for List Sum Calculator

CODE & OUTPUT:



```
AIASS 8.2.py > sum_list
#generate test cases for a function sum_list(numbers)that calculates the sum of list elements.
def sum_list(numbers):
    """Calculate the sum of list elements."""
    total = 0
    for num in numbers:
        total += num
    return total
# Test cases for sum_list
sum_list_test_cases = [
    [1, 2, 3, 4, 5],
    [10, -5, 3],
    [0],
    [],
    [-1, -2, -3]
]
for test in sum_list_test_cases:
    result = sum_list(test)
    print(f"sum_list({test}) = {result}")
```



```
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>  c:; cd 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant
\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Ramya Sri\.vscode\extensions\ms-python.debugpy-2025.18.0-win
ncher' '54232' '--' 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding\AIASS 8.2.py'
sum_list([1, 2, 3, 4, 5]) = 15
sum_list([10, -5, 3]) = 8
sum_list([0]) = 0
sum_list([]) = 0
sum_list([-1, -2, -3]) = -6
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

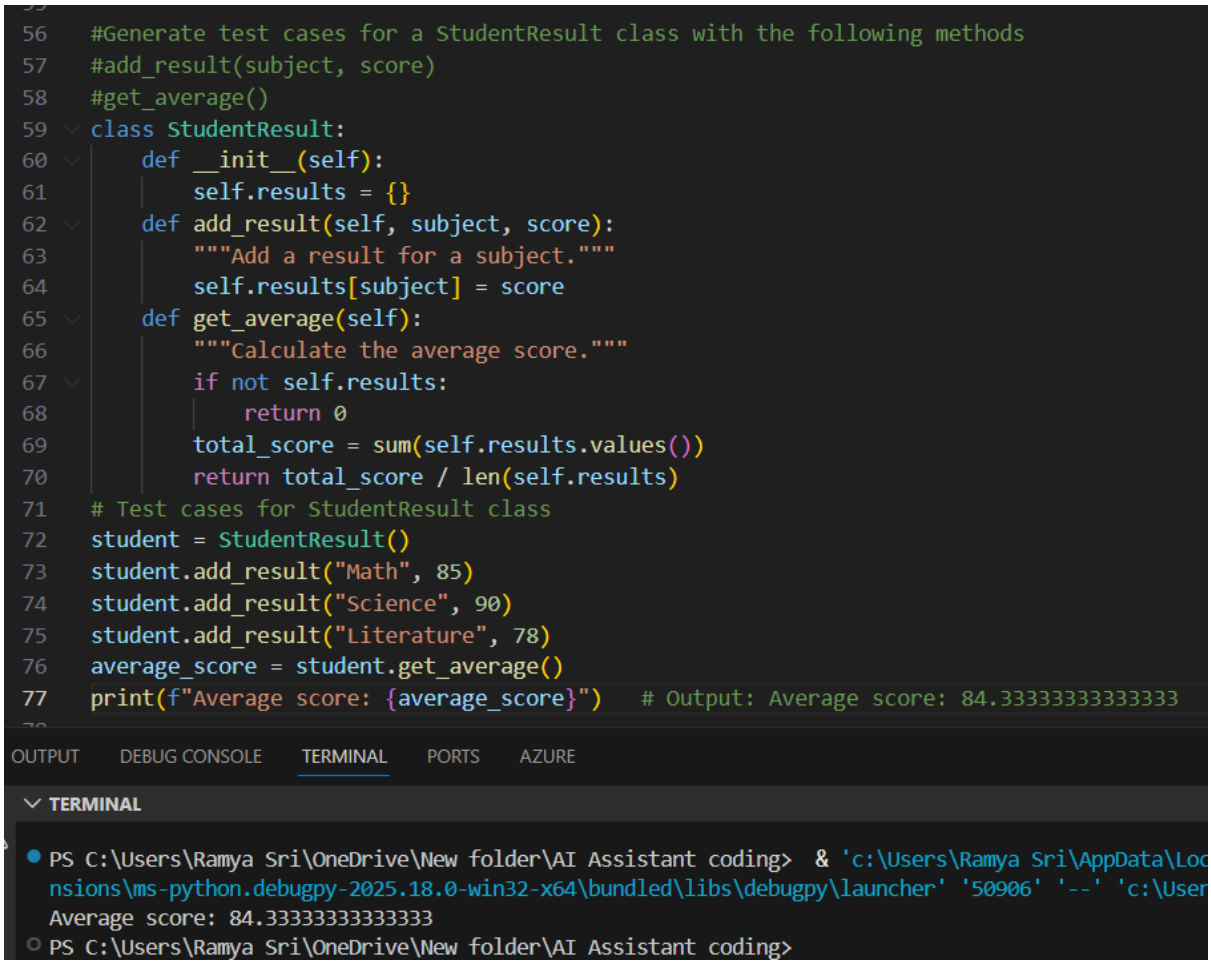
JUSTIFICATION:

List Sum Calculator

The tests include empty lists, negatives, and mixed data types. This ensures the sum_list() function is resilient and avoids crashes when encountering non-numeric values. TDD here guarantees predictable behaviour and safe handling of irregular inputs.

TASK 4: Test Cases for Student Result Class

CODE & OUTPUT:



```python
#Generate test cases for a StudentResult class with the following methods
#add_result(subject, score)
#get_average()
class StudentResult:
    def __init__(self):
        self.results = {}
    def add_result(self, subject, score):
        """Add a result for a subject."""
        self.results[subject] = score
    def get_average(self):
        """Calculate the average score."""
        if not self.results:
            return 0
        total_score = sum(self.results.values())
        return total_score / len(self.results)
# Test cases for StudentResult class
student = StudentResult()
student.add_result("Math", 85)
student.add_result("Science", 90)
student.add_result("Literature", 78)
average_score = student.get_average()
print(f"Average score: {average_score}")   # Output: Average score: 84.33333333333333
```

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE
TERMINAL
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>  & 'c:\Users\Ramya Sri\AppData\Loc
nsions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50906' '--' 'c:\User
Average score: 84.33333333333333
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

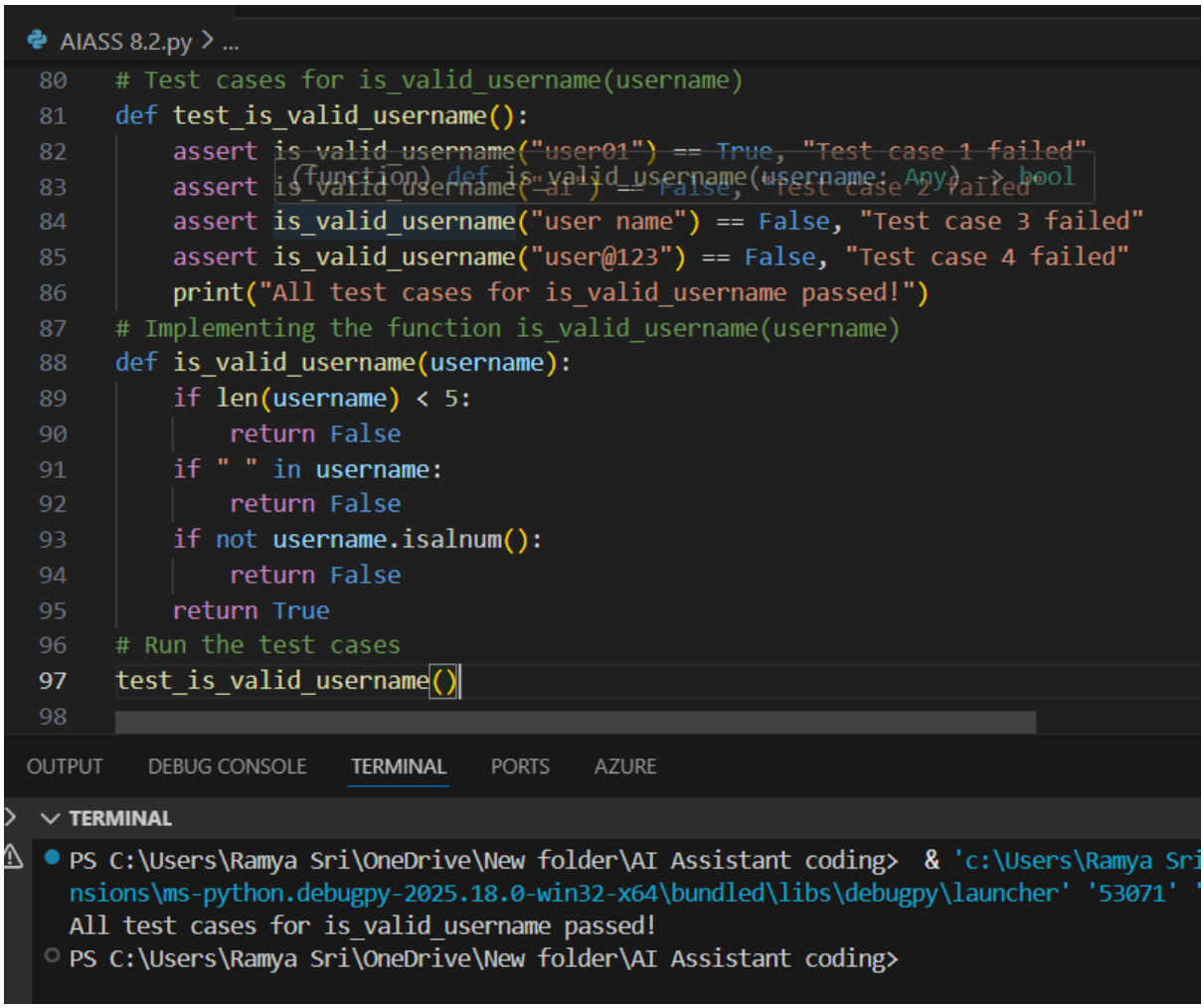
JUSTIFICATION:

Student Result Class

By generating tests for valid marks, invalid marks, and pass/fail thresholds, the class is validated against both expected and erroneous inputs. TDD ensures the class enforces rules (0–100 range) and produces consistent results, making it reliable for academic evaluation systems.

TASK 5: Test-Driven Development for Username Validator

CODE & OUTPUT:



JUSTIFICATION:

Username Validator

The tests enforce rules like minimum length, no spaces, and alphanumeric-only input. Writing these tests first ensures the validator is strict, consistent, and secure. TDD prevents weak or invalid usernames from passing through, which is critical for authentication systems.