ASSIGNMENT-8.2

ROLL-NO:2303A51600

NAME: ARSHA VARDHINI

BATCH-29

Types of test cases:

1.Assert test case- An assert test case uses the assert keyword to automatically verify expected output.

2.normal test case- A normal test case checks output using print() and requires manual verification.

3. Exception Test Case: An exception test case checks whether a function correctly raises an error for invalid input.

4. unit test Test Case: A unittest test case uses Python's unittest module to test functions inside a class structure.

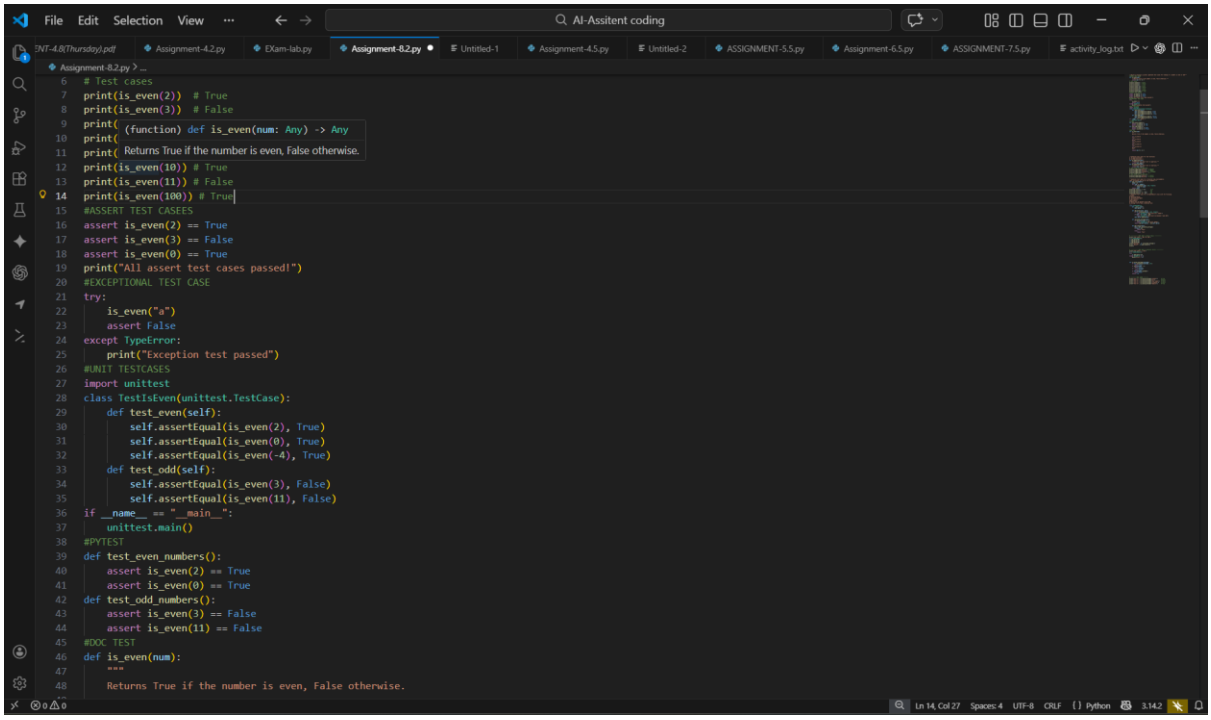
5. pytest Test Case: A pytest test case uses the pytest framework and relies on simple assert statements.

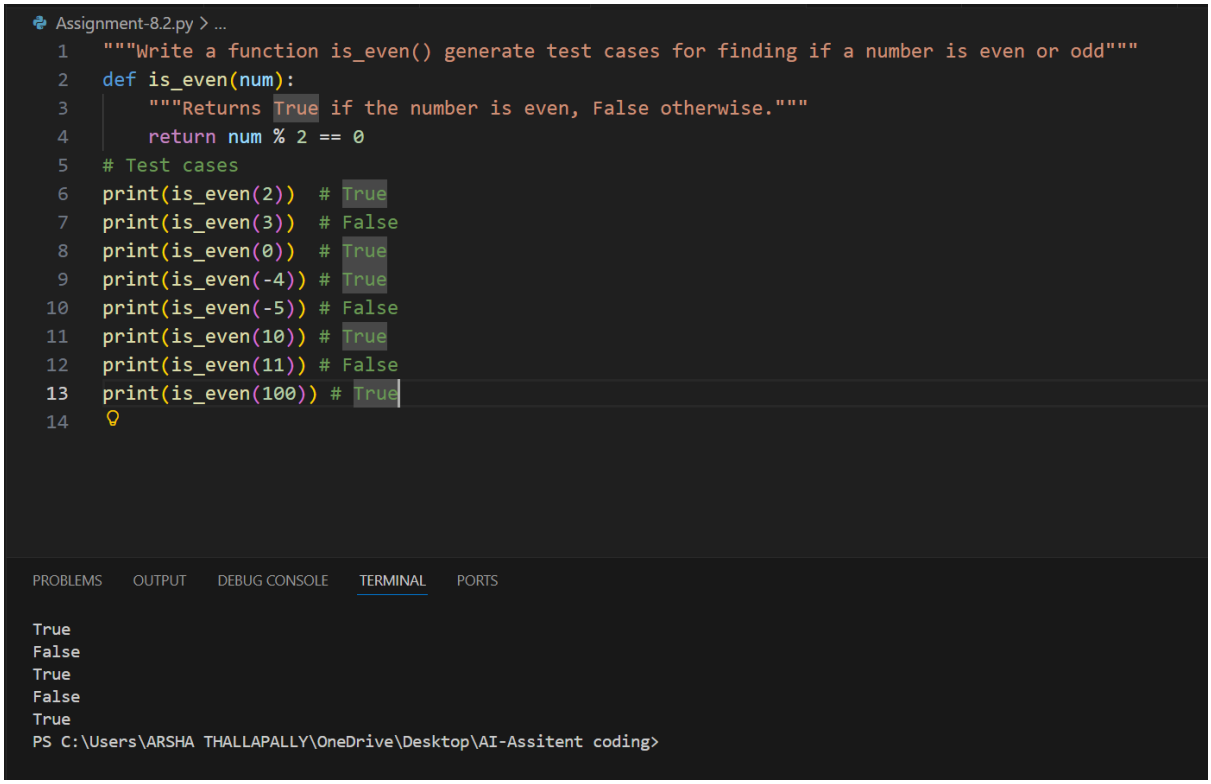
6. Doctest: A doctest is written inside a function's docstring and looks like an interactive Python session.

TASK-1:

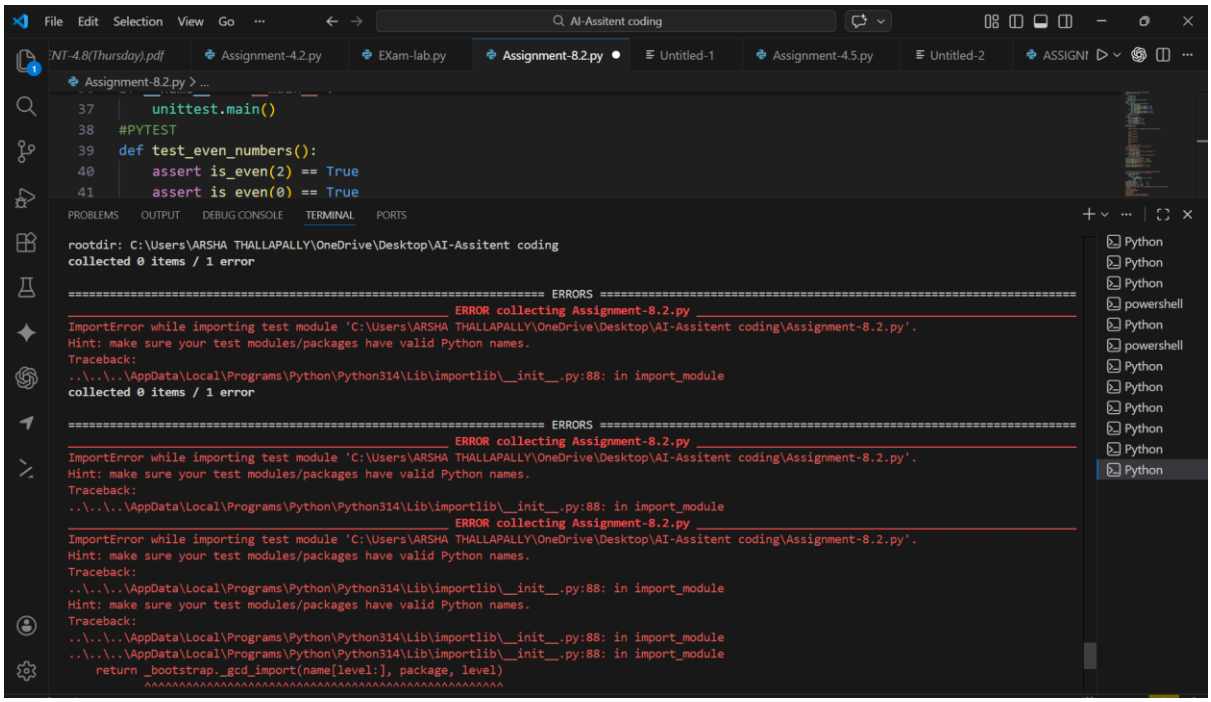
PROMPT: Write a function is_even() generate test cases for finding if a number is even or odd

CODE:


File Edit Selection View
AI-Assitent coding
Assignment-8.2.py
Assignment-4.2.py
Exam-lab.py
Untitled-1
Assignment-4.5.py
Untitled-2
ASSIGNMENT-5.5.py
Assignment-6.5.py
ASSIGNMENT-7.5.py
activity_log.txt
(function) def is_even(num: Any) -> Any
Returns True if the number is even, False otherwise.
Test cases
print(is_even(2)) # True
print(is_even(3)) # False
print(is_even(10)) # True
print(is_even(11)) # False
print(is_even(100)) # True
#ASSERT TEST CASEES
assert is_even(2) == True
assert is_even(3) == False
assert is_even(0) == True
print("All assert test cases passed!")
#EXCEPTIONAL TEST CASE
try:
is_even("a")
assert False
except TypeError:
print("Exception test passed")
#UNIT TESTCASES
import unittest
class TestIsEven(unittest.TestCase):
def test_even(self):
self.assertEqual(is_even(2), True)
self.assertEqual(is_even(0), True)
self.assertEqual(is_even(-4), True)
def test_odd(self):
self.assertEqual(is_even(3), False)
self.assertEqual(is_even(11), False)
if __name__ == "__main__":
unittest.main()
#PYTEST
def test_even_numbers():
assert is_even(2) == True
assert is_even(0) == True
def test_odd_numbers():
assert is_even(3) == False
assert is_even(11) == False
#DOC TEST
def is_even(num):
"""
Returns True if the number is even, False otherwise.


Assignment-8.2.py > ...
"""Write a function is_even() generate test cases for finding if a number is even or odd"""
def is_even(num):
"""Returns True if the number is even, False otherwise."""
return num % 2 == 0
Test cases
print(is_even(2)) # True
print(is_even(3)) # False
print(is_even(0)) # True
print(is_even(-4)) # True
print(is_even(-5)) # False
print(is_even(10)) # True
print(is_even(11)) # False
print(is_even(100)) # True
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
True
False
True
False
True
PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding>

PYTEST:



OBSERVATION :

1. Normal test case:

I printed the output of is_even() and checked manually if the numbers are even or odd.

2. Assert test case:

I used assert to automatically check if is_even() returned True for even numbers and False for odd numbers.

3. Exception test case:

I tested is_even() with invalid input like a string to see if it raises an error properly.

4. unittest test case:

I wrote a unittest class to test multiple even and odd numbers in an organized way.

5. pytest test case:

I used simple assert statements in pytest to check the function for different inputs.**Doctest:**

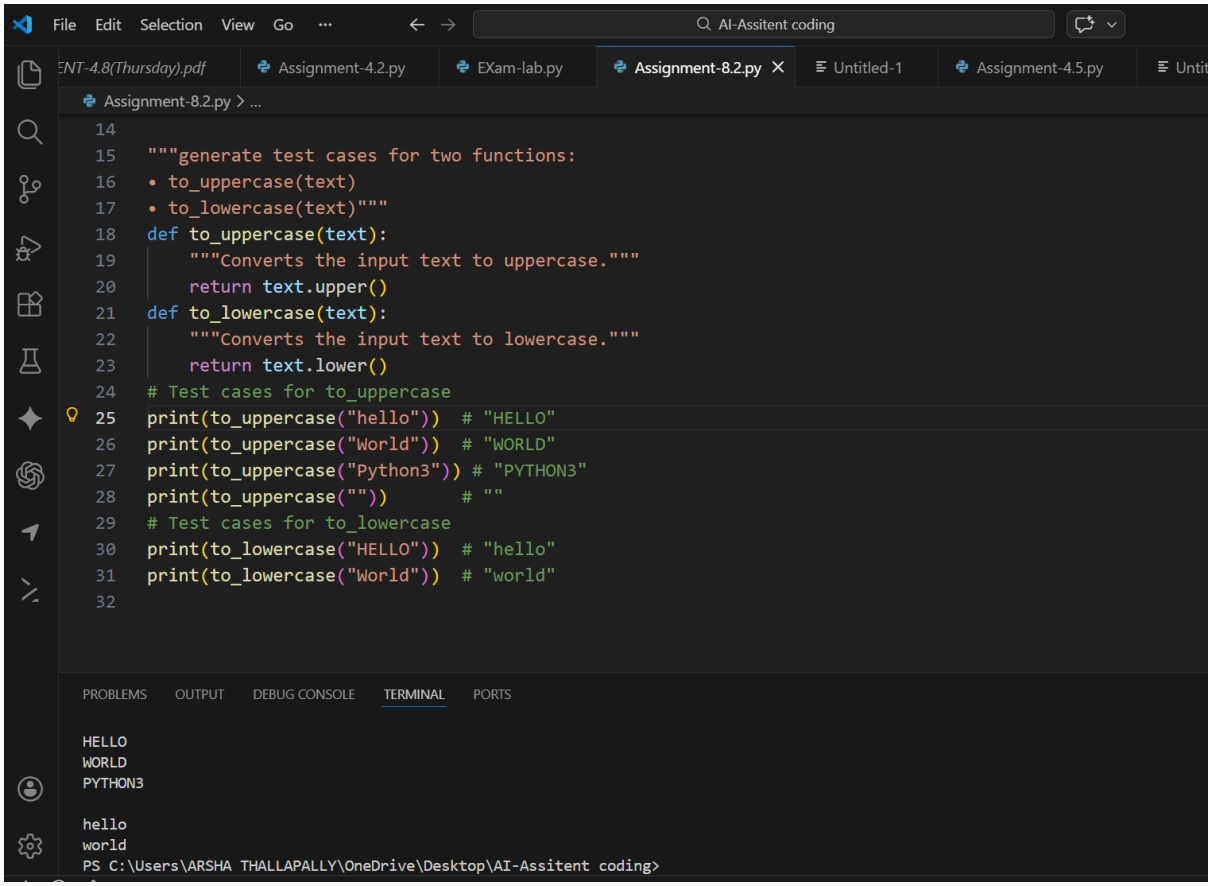
I added examples inside the function's docstring and ran doctest to automatically verify the outputs**.**

TASK-2

PROMPT: GENERATE TEST CASES FOR A FUNCTION SUM_LIST(NUMBERS)

THAT CALCULATES THE SUM OF LIST ELEMENTS

CODE:



OBSERVATION:

The function correctly sums a list of positive numbers.

For an empty list, it returns 0 as expected.

It handles a mix of negative and positive numbers properly

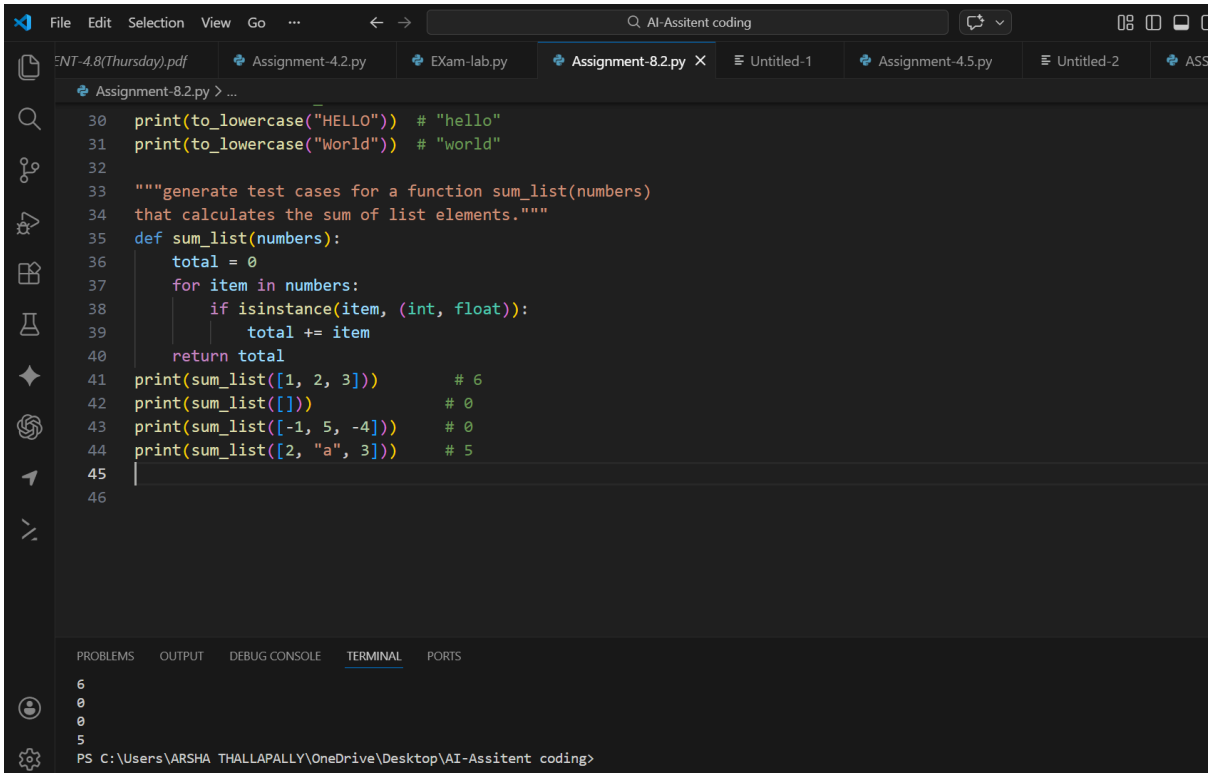
TASK-3

PROMPT:

generate test cases for a function sum_list(numbers)

that calculates the sum of list elements.

CODE:



OBSERVATION:

Correctly calculates average and returns **Pass** for all marks above 40.

Returns **Fail** when average is below 40.

Passes at the boundary where average equals 40.

Raises error for marks below 0.

Raises error for marks above 100.

TASK-4:

PROMPT:

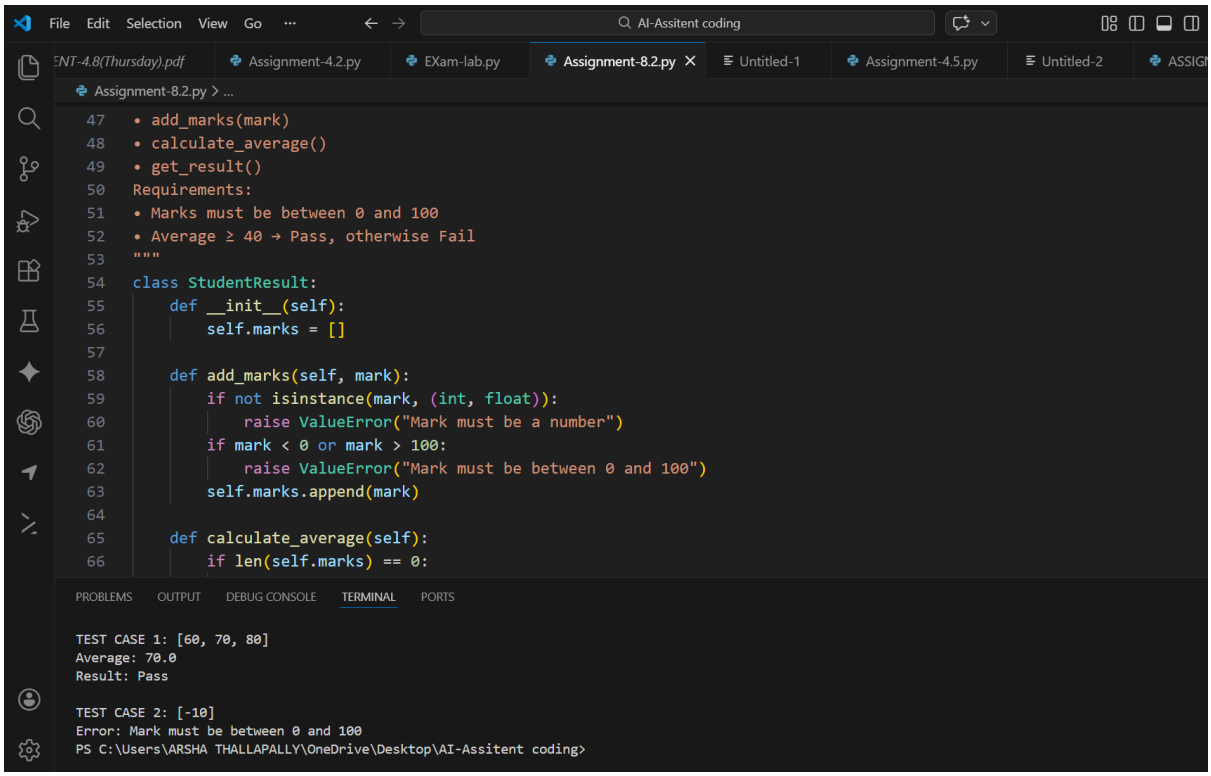
Generate test cases for a StudentResult class with the following methods:

- add_marks(mark)
- calculate_average()
- get_result()

Requirements:

- Marks must be between 0 and 100
- Average ≥ 40 → Pass, otherwise Fail

CODE:



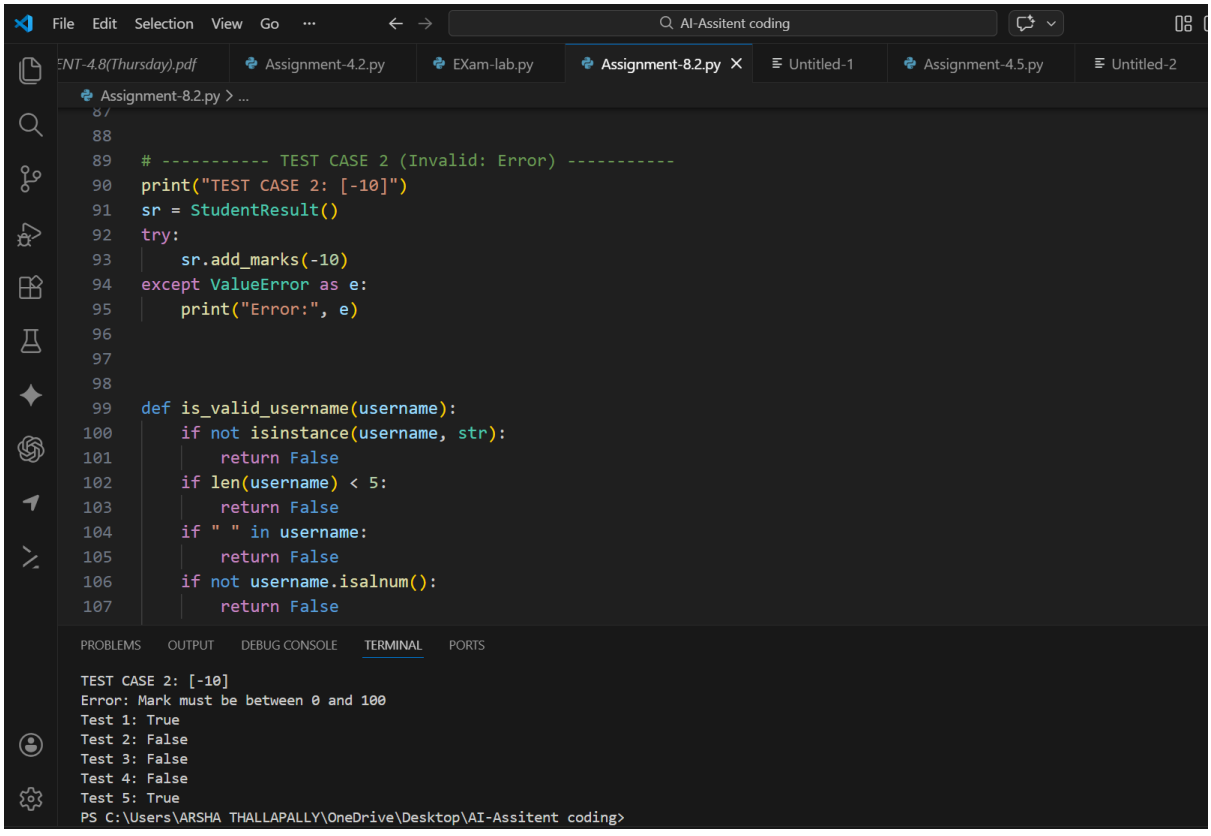
OBSERVATION:

The StudentResult class correctly calculates averages and returns Pass for averages ≥ 40 and Fail for averages < 40.
It also raises errors for invalid marks (<0 or >100) or when no marks are added

TASK-5

PROMPT: Generate code for Minimum length: 5 characters,No spaces allowed, Only alphanumeric characters include all types of test cases mentioned

CODE:



OBSERVATION: The is_valid_username function correctly checks that usernames are at least 5 characters long, contain no spaces, and are alphanumeric.