

## 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:

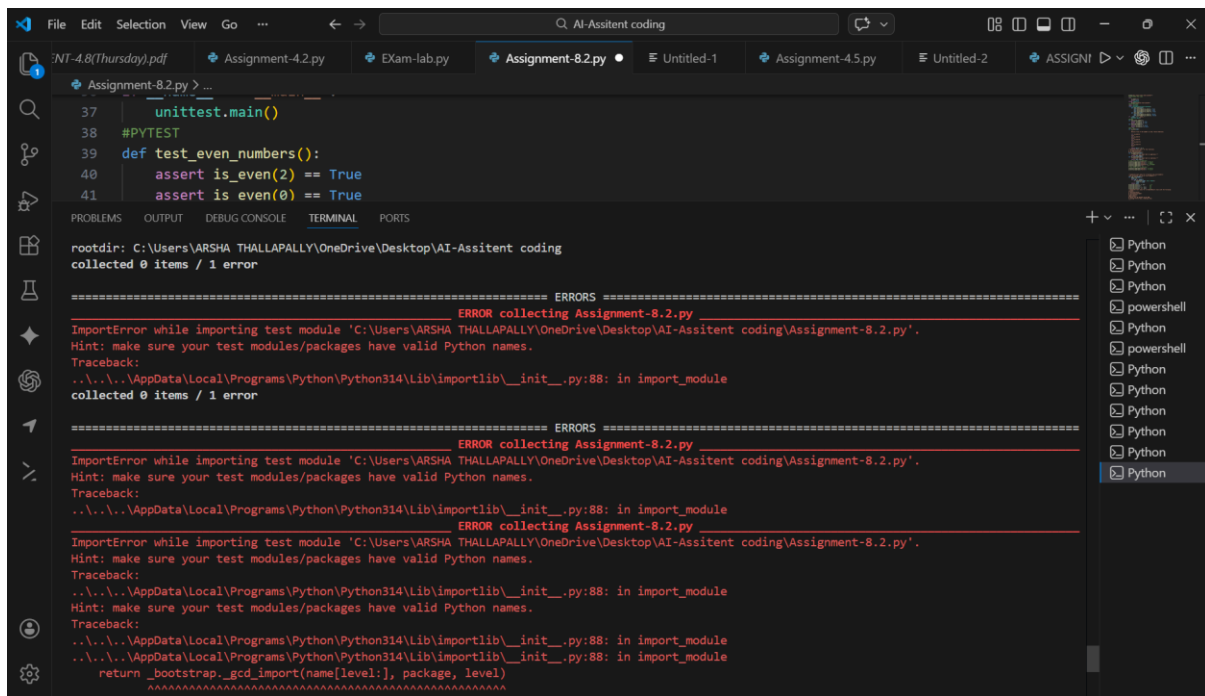
```
File Edit Selection View ... AI-Assitent coding
Assignment-8.2.py > ...
6 # Test cases
7 print(is_even(2)) # True
8 print(is_even(3)) # False
9 print(
10 (function) def is_even(num: Any) -> Any
11 print( Returns True if the number is even, False otherwise.
12 print(is_even(10)) # True
13 print(is_even(11)) # False
14 print(is_even(100)) # True
15 #ASSERT TEST CASES
16 assert is_even(2) == True
17 assert is_even(3) == False
18 assert is_even(0) == True
19 print("All assert test cases passed!")
20 #EXCEPTIONAL TEST CASE
21 try:
22     is_even("a")
23     assert False
24 except TypeError:
25     print("Exception test passed")
26 #UNIT TESTCASES
27 import unittest
28 class TestIsEven(unittest.TestCase):
29     def test_even(self):
30         self.assertEqual(is_even(2), True)
31         self.assertEqual(is_even(0), True)
32         self.assertEqual(is_even(-4), True)
33     def test_odd(self):
34         self.assertEqual(is_even(3), False)
35         self.assertEqual(is_even(11), False)
36 if __name__ == "__main__":
37     unittest.main()
38 #PYTEST
39 def test_even_numbers():
40     assert is_even(2) == True
41     assert is_even(0) == True
42 def test_odd_numbers():
43     assert is_even(3) == False
44     assert is_even(11) == False
45 #DOC TEST
46 def is_even(num):
47     """
48     Returns True if the number is even, False otherwise.
```

```
Assignment-8.2.py > ...
1 """Write a function is_even() generate test cases for finding if a number is even or odd"""
2 def is_even(num):
3     """Returns True if the number is even, False otherwise."""
4     return num % 2 == 0
5 # Test cases
6 print(is_even(2)) # True
7 print(is_even(3)) # False
8 print(is_even(0)) # True
9 print(is_even(-4)) # True
10 print(is_even(-5)) # False
11 print(is_even(10)) # True
12 print(is_even(11)) # False
13 print(is_even(100)) # True
14

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:**

## .Doctest:

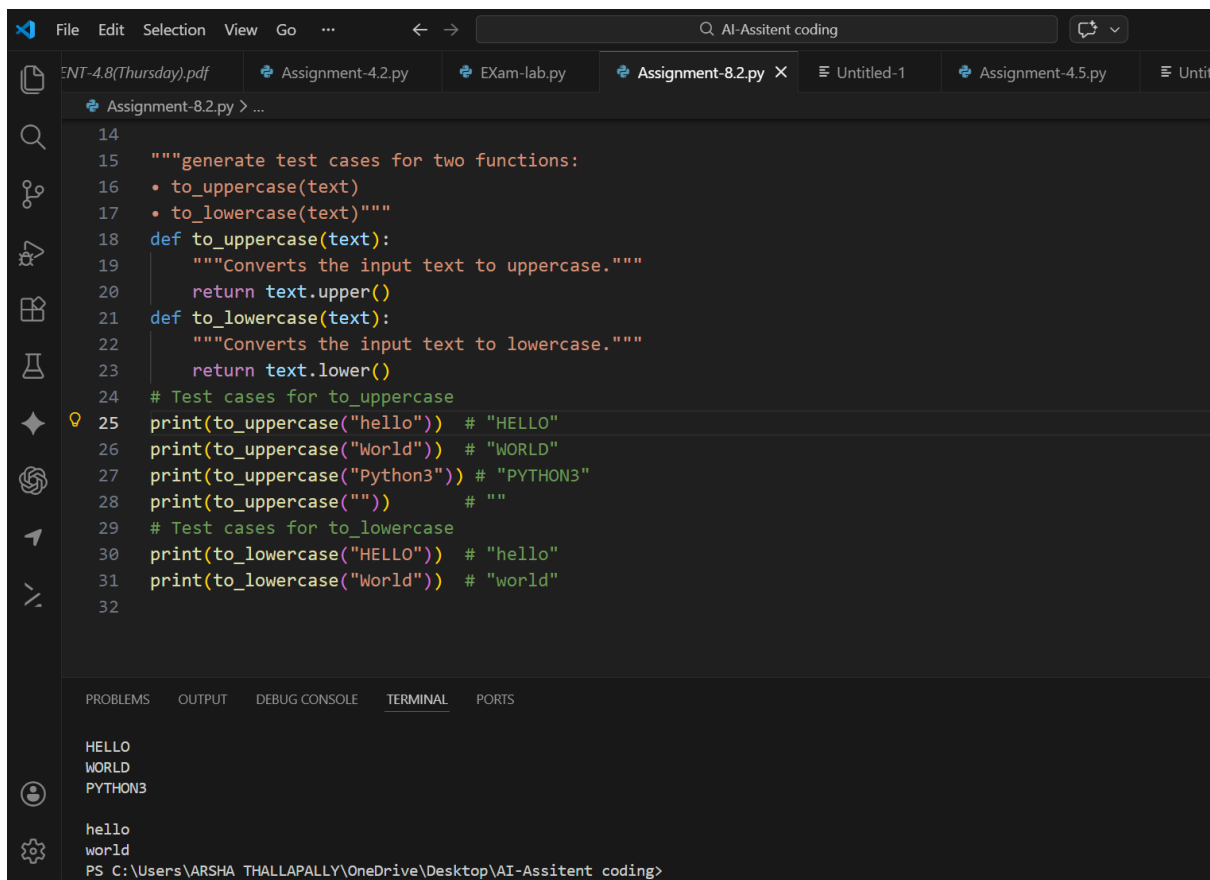
I added examples inside the function's docstring and ran doctest to automatically verify the outputs.

## TASK-2

PROMPT: GENERATE TEST CASES FOR A FUNCTION  
SUM\_LIST(NUMBERS)

THAT CALCULATES THE SUM OF LIST ELEMENTS

CODE:



```
14
15 """generate test cases for two functions:
16 • to_uppercase(text)
17 • to_lowercase(text)"""
18 def to_uppercase(text):
19     """Converts the input text to uppercase."""
20     return text.upper()
21 def to_lowercase(text):
22     """Converts the input text to lowercase."""
23     return text.lower()
24 # Test cases for to_uppercase
25 print(to_uppercase("hello")) # "HELLO"
26 print(to_uppercase("World")) # "WORLD"
27 print(to_uppercase("Python3")) # "PYTHON3"
28 print(to_uppercase("")) # ""
29 # Test cases for to_lowercase
30 print(to_lowercase("HELLO")) # "hello"
31 print(to_lowercase("World")) # "world"
32
```

HELLO  
WORLD  
PYTHON3  
  
hello  
world  
PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding>

OBSERVATION:

The function correctly sums a list of positive numbers.

For an empty list, it returns 0 as expected.

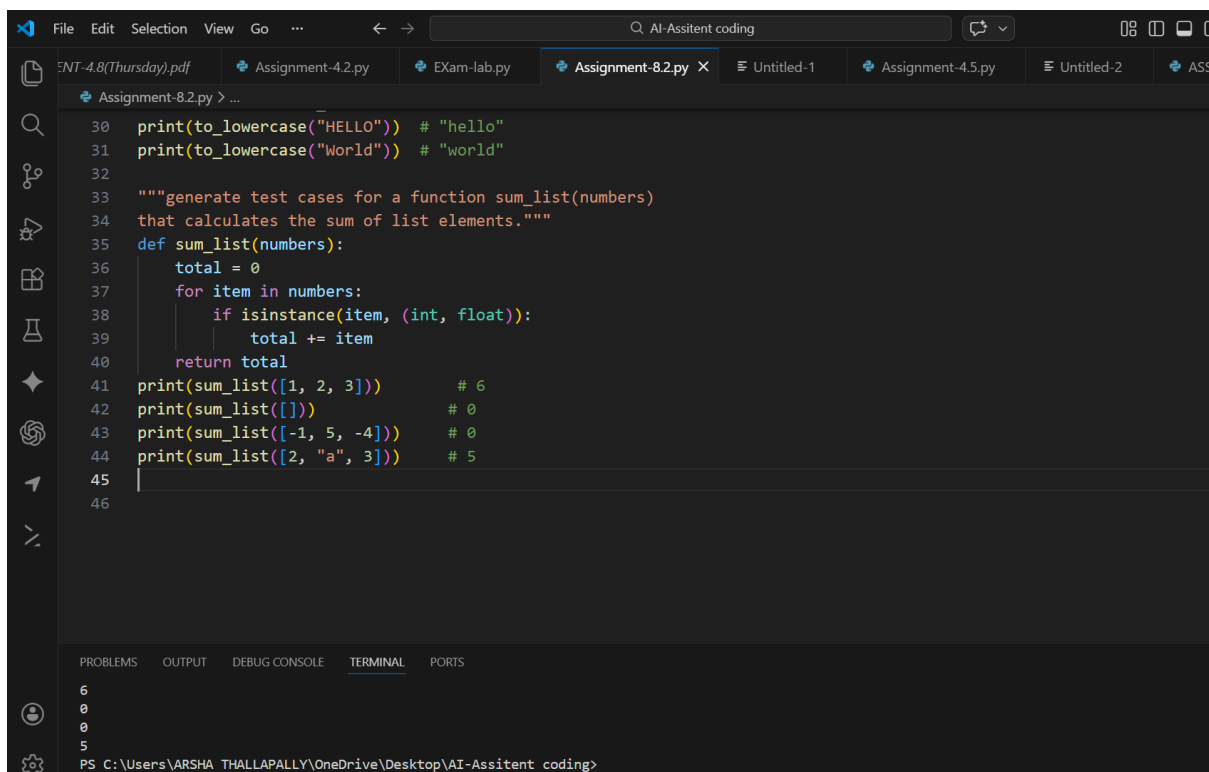
It handles a mix of negative and positive numbers properly

## TASK-3

### PROMPT:

generate test cases for a function `sum_list(numbers)` that calculates the sum of list elements.

### CODE:



```
30 print(to_lowercase("HELLO")) # "hello"
31 print(to_lowercase("World")) # "world"
32
33 """generate test cases for a function sum_list(numbers)
34 that calculates the sum of list elements."""
35 def sum_list(numbers):
36     total = 0
37     for item in numbers:
38         if isinstance(item, (int, float)):
39             total += item
40     return total
41 print(sum_list([1, 2, 3])) # 6
42 print(sum_list([])) # 0
43 print(sum_list([-1, 5, -4])) # 0
44 print(sum_list([2, "a", 3])) # 5
45
46
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
6
0
0
5
```

PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding>

### 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:

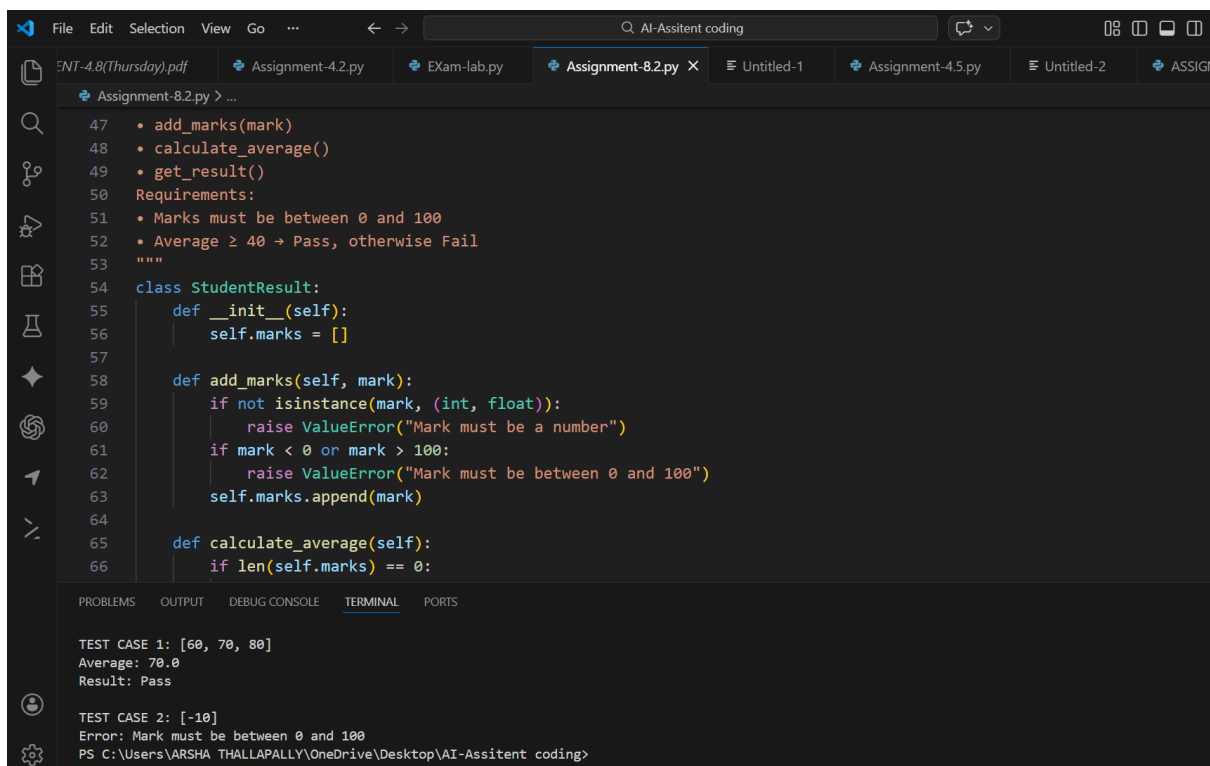
Generate test cases for a StudentResult class with the following methods:

- add\_marks(mark)
- calculate\_average()
- get\_result()

Requirements:

- Marks must be between 0 and 100
- Average  $\geq 40 \rightarrow$  Pass, otherwise Fail

### CODE:



```
File Edit Selection View Go ... AI-Assitent coding
Assignment-8.2.py x
Assignment-8.2.py > ...
47 • add_marks(mark)
48 • calculate_average()
49 • get_result()
50 Requirements:
51 • Marks must be between 0 and 100
52 • Average  $\geq 40 \rightarrow$  Pass, otherwise Fail
53 """
54 class StudentResult:
55     def __init__(self):
56         self.marks = []
57
58     def add_marks(self, mark):
59         if not isinstance(mark, (int, float)):
60             raise ValueError("Mark must be a number")
61         if mark < 0 or mark > 100:
62             raise ValueError("Mark must be between 0 and 100")
63         self.marks.append(mark)
64
65     def calculate_average(self):
66         if len(self.marks) == 0:
67
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
TEST CASE 1: [60, 70, 80]
Average: 70.0
Result: Pass
TEST CASE 2: [-10]
Error: Mark must be between 0 and 100
PS C:\Users\ARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding>
```

## OBSERVATION:

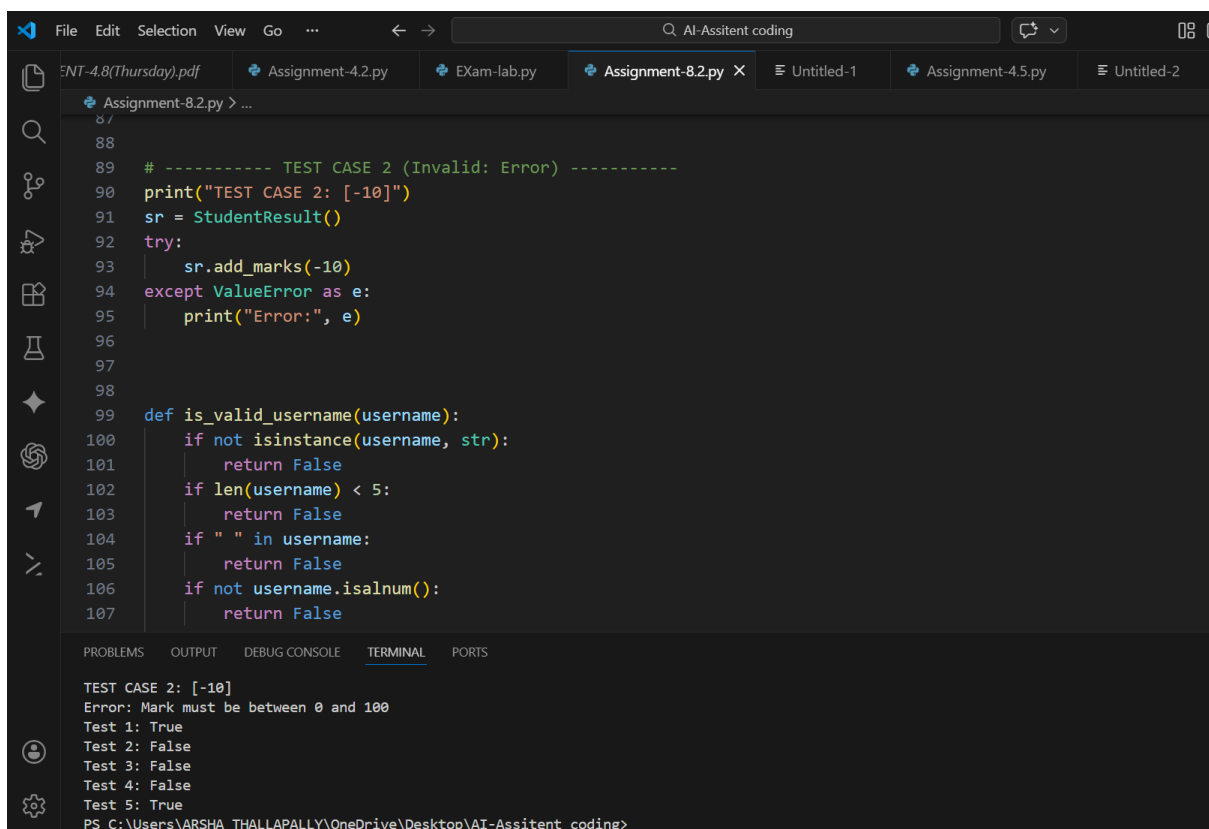
The StudentResult class correctly calculates averages and returns Pass for averages  $\geq 40$  and Fail for averages  $< 40$ .

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:



```
88
89 # ----- TEST CASE 2 (Invalid: Error) -----
90 print("TEST CASE 2: [-10]")
91 sr = StudentResult()
92 try:
93     sr.add_marks(-10)
94 except ValueError as e:
95     print("Error:", e)
96
97
98
99 def is_valid_username(username):
100     if not isinstance(username, str):
101         return False
102     if len(username) < 5:
103         return False
104     if " " in username:
105         return False
106     if not username.isalnum():
107         return False

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

TEST CASE 2: [-10]
Error: Mark must be between 0 and 100
Test 1: True
Test 2: False
Test 3: False
Test 4: False
Test 5: True
PS C:\Users\VARSHA THALLAPALLY\OneDrive\Desktop\AI-Assitent coding>
```

**OBSERVATION:** The is\_valid\_username function correctly checks that usernames are at least 5 characters long, contain no spaces, and are alphanumeric.