

AI Assisted Coding

Assignment – 8.2

K.VAMSHIDHAR || 2303A510H7 || Batch:- 8

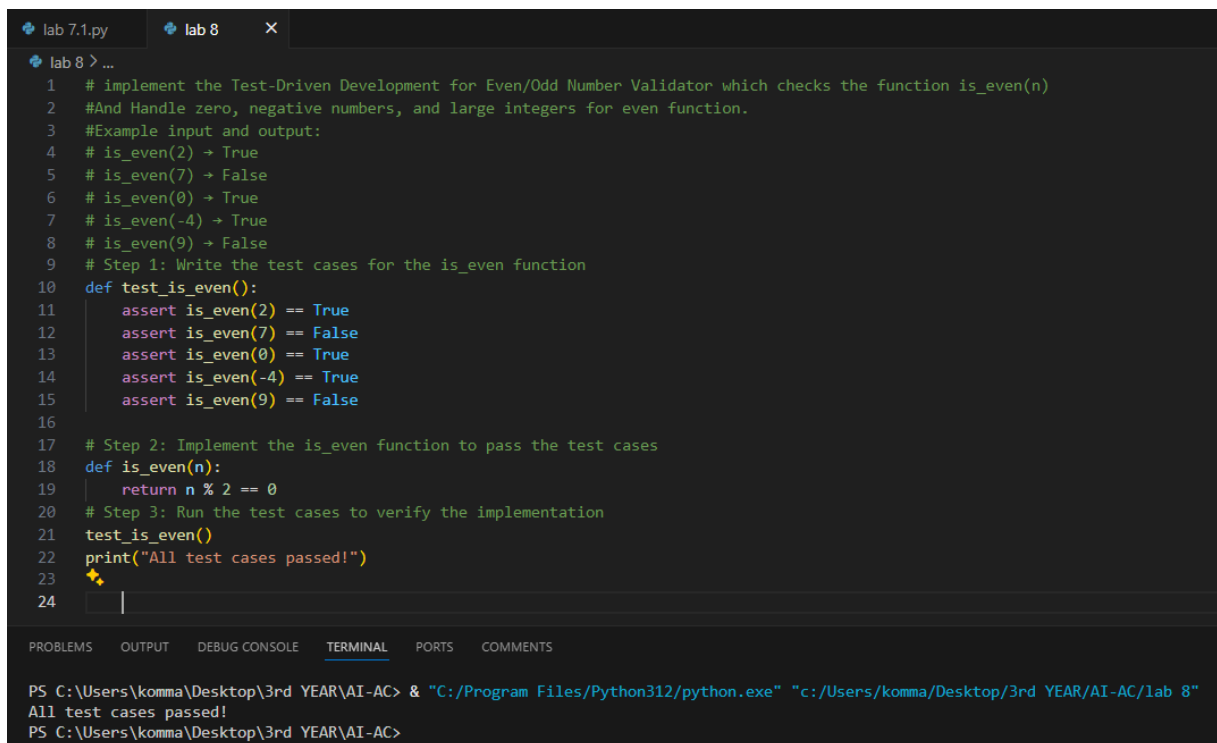
Task 1 – Test-Driven Development for Even/Odd Number Validator

- Use AI tools to first generate test cases for a function `is_even(n)`

and then implement the function so that it satisfies all generated tests.

Requirements:

- Input must be an integer
- Handle zero, negative numbers, and large integers



```
lab 8 > ...
1  # implement the Test-Driven Development for Even/Odd Number Validator which checks the function is_even(n)
2  #And Handle zero, negative numbers, and large integers for even function.
3  #Example input and output:
4  # is_even(2) → True
5  # is_even(7) → False
6  # is_even(0) → True
7  # is_even(-4) → True
8  # is_even(9) → False
9  # Step 1: Write the test cases for the is_even function
10 def test_is_even():
11     assert is_even(2) == True
12     assert is_even(7) == False
13     assert is_even(0) == True
14     assert is_even(-4) == True
15     assert is_even(9) == False
16
17 # Step 2: Implement the is_even function to pass the test cases
18 def is_even(n):
19     return n % 2 == 0
20 # Step 3: Run the test cases to verify the implementation
21 test_is_even()
22 print("All test cases passed!")
23 ✨
24 |

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

PS C:\Users\komma\Desktop\3rd YEAR\AI-AC> & "C:/Program Files/Python312/python.exe" "c:/Users/komma/Desktop/3rd YEAR/AI-AC/lab 8"
All test cases passed!
PS C:\Users\komma\Desktop\3rd YEAR\AI-AC>
```

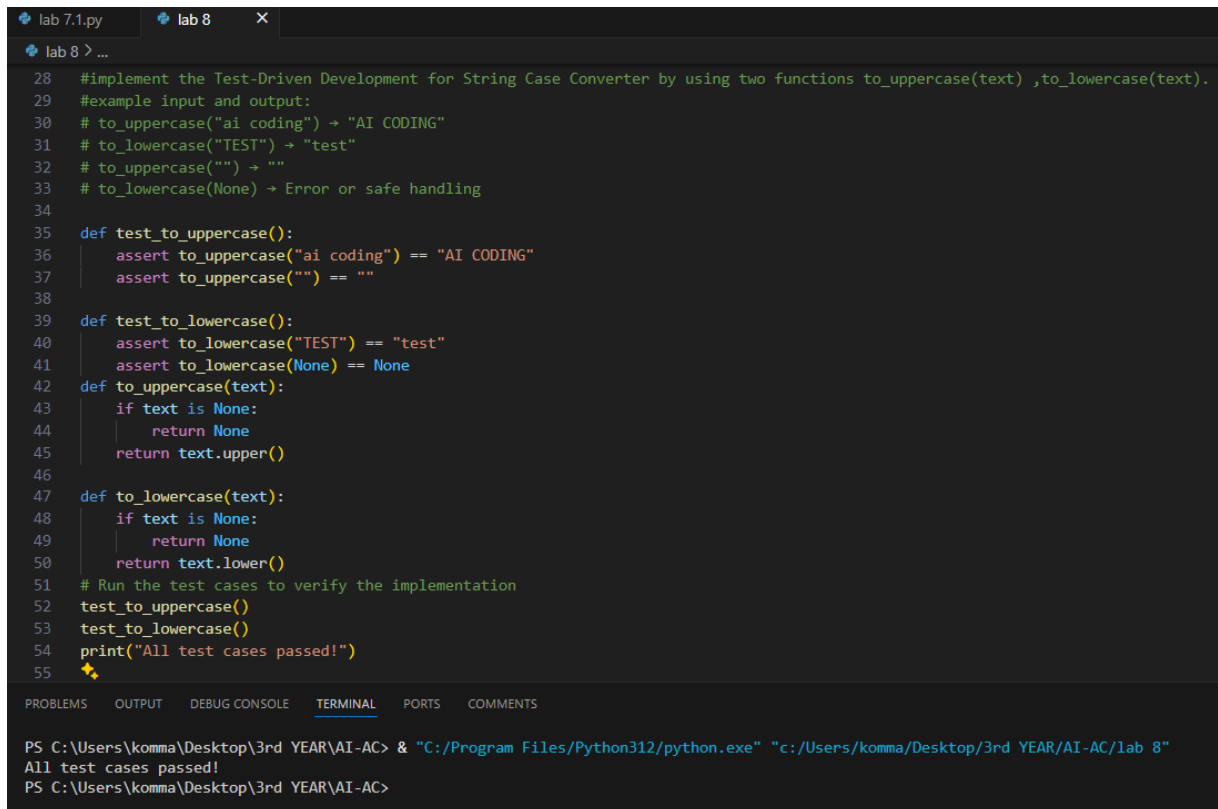
Task 2 – Test-Driven Development for String Case Converter

- Ask AI to generate test cases for two functions:
- `to_uppercase(text)`

- to_lowercase(text)

Requirements:

- Handle empty strings
- Handle mixed-case input
- Handle invalid inputs such as numbers or None



```

28 #implement the Test-Driven Development for String Case Converter by using two functions to_uppercase(text) ,to_lowercase(text).
29 #example input and output:
30 # to_uppercase("ai coding") → "AI CODING"
31 # to_lowercase("TEST") → "test"
32 # to_uppercase("") → ""
33 # to_lowercase(None) → Error or safe handling
34
35 def test_to_uppercase():
36     assert to_uppercase("ai coding") == "AI CODING"
37     assert to_uppercase("") == ""
38
39 def test_to_lowercase():
40     assert to_lowercase("TEST") == "test"
41     assert to_lowercase(None) == None
42 def to_uppercase(text):
43     if text is None:
44         return None
45     return text.upper()
46
47 def to_lowercase(text):
48     if text is None:
49         return None
50     return text.lower()
51 # Run the test cases to verify the implementation
52 test_to_uppercase()
53 test_to_lowercase()
54 print("All test cases passed!")
55

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```

PS C:\Users\komma\Desktop\3rd YEAR\AI-AC> & "C:/Program Files/Python312/python.exe" "c:/Users/komma/Desktop/3rd YEAR/AI-AC/lab 8"
All test cases passed!
PS C:\Users\komma\Desktop\3rd YEAR\AI-AC>

```

Task 3 – Test-Driven Development for List Sum Calculator

- Use AI to generate test cases for a function sum_list(numbers) that calculates the sum of list elements.

Requirements:

- Handle empty lists
- Handle negative numbers
- Ignore or safely handle non-numeric values

```
lab 7.1.py lab 8 X
lab 8 > ...
58
59 # Generate code for implementing the Test-Driven Development for List Sum Calculator using function sum_list(numbers).
60 # example input and output:
61 # sum_list([1, 2, 3]) → 6
62 # sum_list([]) → 0
63 # sum_list([-1, 5, -4]) → 0
64 # sum_list([2, "a", 3]) → 5
65 def test_sum_list():
66     assert sum_list([1, 2, 3]) == 6
67     assert sum_list([]) == 0
68     assert sum_list([-1, 5, -4]) == 0
69     assert sum_list([2, "a", 3]) == 5
70
71 def sum_list(numbers):
72     total = 0
73     for num in numbers:
74         if isinstance(num, (int, float)):
75             total += num
76     return total
77 # Run the test cases to verify the implementation
78 test_sum_list()
79 print("All test cases passed!")
80 ✨
81
82
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
PS C:\Users\komma\Desktop\3rd YEAR\AI-AC> & "C:/Program Files/Python312/python.exe" "c:/Users/komma/Desktop/3rd YEAR/AI-AC/lab 8"
All test cases passed!
PS C:\Users\komma\Desktop\3rd YEAR\AI-AC>
```

Task 4 – Test Cases for Student Result Class

- 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

```
lab 7.1.py  lab 8  •
lab 8 > ...
83 # Generate code for checking Test Cases for Student Result Class by using functions add_marks(mark), calculate_average(), get_result().
84 #Average ≥ 40 → Pass, otherwise Fail.
85 #example input and output:
86 # Marks: [60, 70, 80] → Average: 70 → Result: Pass
87 # Marks: [30, 35, 40] → Average: 35 → Result: Fail
88 # Marks: [-10] → Error
89 class StudentResult:
90     def __init__(self):
91         self.marks = []
92
93     def add_marks(self, mark):
94         if mark < 0:
95             raise ValueError("Marks cannot be negative")
96         self.marks.append(mark)
97
98     def calculate_average(self):
99         if not self.marks:
100             return 0
101         return sum(self.marks) / len(self.marks)
102
103     def get_result(self):
104         average = self.calculate_average()
105         return "Pass" if average >= 40 else "Fail"
106 # Test cases
107 def test_student_result():
108     student = StudentResult()
109     student.add_marks(60)
110     student.add_marks(70)
111     student.add_marks(80)
112     assert student.calculate_average() == 70
113     assert student.get_result() == "Pass"
114
115     student = StudentResult()
116     student.add_marks(30)
117     student.add_marks(35)
118     student.add_marks(40)
119     assert student.calculate_average() == 35
120     assert student.get_result() == "Fail"
121
122     student = StudentResult()
123     student.add_marks(-10)
124     assert student.get_result() == "Error"
125
126 if __name__ == '__main__':
127     test_student_result()
128     print("All test cases passed!")
129
130 PS C:\Users\komma\Desktop\3rd YEAR\AI-AC>
```

Task 5 – Test-Driven Development for Username Validator

Requirements:

- Minimum length: 5 characters
- No spaces allowed
- Only alphanumeric characters

```

132
133 #Generate Code for Test-Driven Development for Username Validator by following the rules:
134 # Minimum length: 5 characters
135 # No spaces allowed
136 # Only alphanumeric characters
137 #Example input and output:
138 # is_valid_username("user01") → True
139 # is_valid_username("ai") → False
140 # is_valid_username("user name") → False
141 # is_valid_username("user@123") → False
142 def test_is_valid_username():
143     assert is_valid_username("user01") == True
144     assert is_valid_username("ai") == False
145     assert is_valid_username("user name") == False
146     assert is_valid_username("user@123") == False
147
148 def is_valid_username(username):
149     if len(username) < 5:
150         return False
151     if " " in username:
152         return False
153     if not username.isalnum():
154         return False
155     return True
156 # Run the test cases to verify the implementation
157 test_is_valid_username()
158 print("All test cases passed!")
159
160

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

All test cases passed!
PS C:\Users\komma\Desktop\3rd YEAR\AI-AC>