

ASSIGNMENT-8.2

Name: B.Shravya

HT. No: 2303A51492

Batch: 08

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

Task Description

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 Example

Test Scenarios:

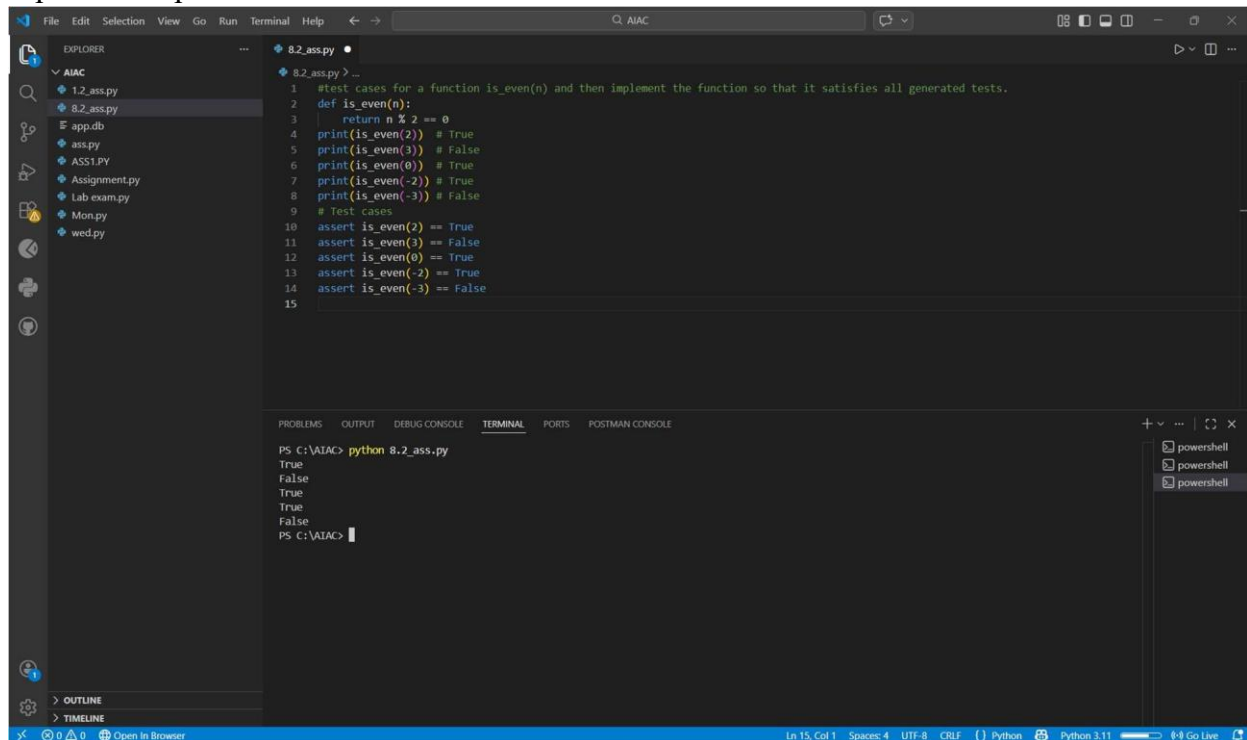
`is_even(2)` → True `is_even(7)`

→ False `is_even(0)` → True

`is_even(-4)` → True

`is_even(9)` → False

Expected Output



```
1 #test cases for a function is_even(n) and then implement the function so that it satisfies all generated tests.
2 def is_even(n):
3     return n % 2 == 0
4     print(is_even(2)) # True
5     print(is_even(3)) # False
6     print(is_even(0)) # True
7     print(is_even(-2)) # True
8     print(is_even(-3)) # False
9 # Test cases
10 assert is_even(2) == True
11 assert is_even(3) == False
12 assert is_even(0) == True
13 assert is_even(-2) == True
14 assert is_even(-3) == False
15
```

```
PS C:\AIAC> python 8.2_ass.py
True
False
True
True
False
False
PS C:\AIAC>
```

- A correctly implemented `is_even()` function that passes all AI-generated test cases

Task Description

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 Example Test Scenarios: `to_uppercase("ai coding")` → "AI CODING"
 - `to_lowercase("TEST")` → "test" `to_uppercase("")` → ""
 - `to_lowercase(None)` → Error or safe handling
- Expected Output

```

15 #Generate test cases and implement two Python functions: to_uppercase(text) and to_lowercase(text).
16 # Requirements:
17 # - Handle empty strings
18 # - Handle mixed case
19 # - Raise TypeError for non-string inputs
20 # - Use assert statements for tests
21 def to_uppercase(text):
22     if not isinstance(text, str):
23         raise TypeError("Input must be a string")
24     return text.upper()
25 def to_lowercase(text):
26     if not isinstance(text, str):
27         raise TypeError("Input must be a string")
28     return text.lower()
29 print(to_uppercase("ai coding")) # "AI CODING"
30 print(to_lowercase("TEST")) # "test"
31 print(to_uppercase("")) # ""

```

```

PS C:\AIAC> python 8.2_ass.py
AI CODING
test
PS C:\AIAC>

```

- Two string conversion functions that pass all AI-generated test cases with safe input handling.

Task Description

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 Example Test Scenarios: `sum_list([1, 2, 3])` → 6 `sum_list([])` → 0 `sum_list([-1, 5, -4])` → 0
- `sum_list([2, "a", 3])` → 5

Expected Output

The screenshot shows a Visual Studio Code editor with a file named `8.2_ass.py` open. The file contains a function `sum_list(numbers, list)` and several test cases. The terminal window at the bottom shows the command `python 8.2_ass.py` being executed, and the output is displayed.

```
32 #Generate test cases for a function sum_list(numbers).
33 # Requirements:
34 # - Handle empty lists
35 # - Handle negative numbers
36 # - Ignore non-numeric values
37 # - Return 0 for empty list
38 # - Use assert statements
39 def sum_list(numbers, list):
40     if not isinstance(numbers, list):
41         raise TypeError("Input must be a list")
42     return sum(num for num in numbers if isinstance(num, (int, float)))
43 # Test cases
44 print(sum_list([1, 2, 3])) # 6
45 print(sum_list([-1, -2, -3])) # -6
46 print(sum_list([1, 'a', 2, None, 3])) # 6
47 print(sum_list([])) # 0
```

```
PS C:\AIAC> python 8.2_ass.py
AI CODING
test
PS C:\AIAC> python 8.2_ass.py
6
-6
6
0
PS C:\AIAC>
```

- A robust list-sum function validated using AI-generated test cases.

Task Description

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:

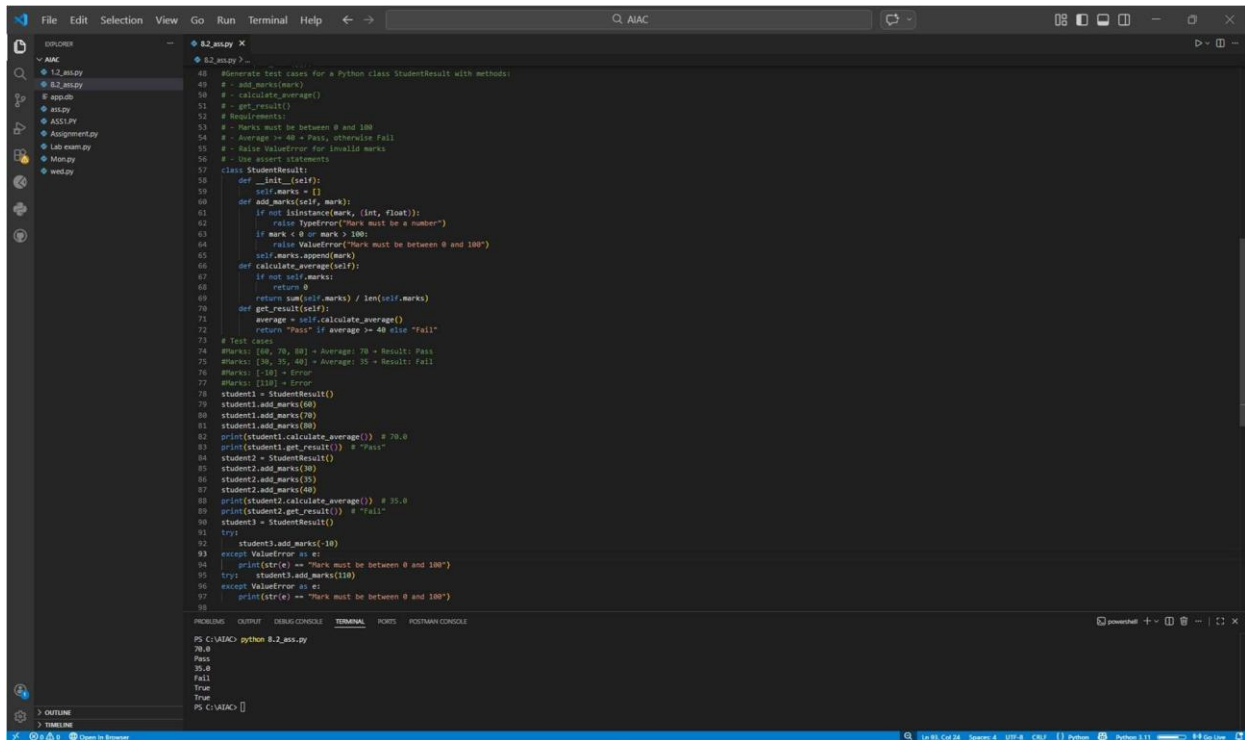
- Marks must be between 0 and 100 • Average $\geq 40 \rightarrow$ Pass, otherwise Fail

Marks: [60, 70, 80] \rightarrow Average: 70 \rightarrow Result: Pass

Marks: [30, 35, 40] \rightarrow Average: 35 \rightarrow Result: Fail

Marks: [-10] \rightarrow Error

Expected Output



```
40 #Generate test cases for a Python class StudentResult with methods:
41 # - add_marks(marks)
42 # - calculate_average()
43 # - get_result()
44 # Requirements:
45 # - marks must be between 0 and 100
46 # - Average >= 40 = Pass, otherwise Fail
47 # - Raise ValueError for invalid marks
48 # - Use assert statements
49
50 class StudentResult:
51     def __init__(self):
52         self.marks = []
53
54     def add_marks(self, marks):
55         if not isinstance(marks, (list, tuple)):
56             raise ValueError("Mark must be a number")
57         if mark < 0 or mark > 100:
58             raise ValueError("Mark must be between 0 and 100")
59         self.marks.append(marks)
60
61     def calculate_average(self):
62         if not self.marks:
63             return 0
64         return sum(self.marks) / len(self.marks)
65
66     def get_result(self):
67         average = self.calculate_average()
68         return "Pass" if average >= 40 else "Fail"
69
70 # Test cases
71 # Marks: [80, 70, 90] = Average: 80 = Result: Pass
72 # Marks: [30, 35, 40] = Average: 35 = Result: Fail
73 # Marks: [10] = Error
74 # Marks: [150] = Error
75
76 student1 = StudentResult()
77 student1.add_marks(80)
78 student1.add_marks(80)
79
80 print(student1.calculate_average()) # 70.0
81 print(student1.get_result()) # "Pass"
82
83 student2 = StudentResult()
84 student2.add_marks(30)
85 student2.add_marks(35)
86 student2.add_marks(40)
87
88 print(student2.calculate_average()) # 35.0
89 print(student2.get_result()) # "Fail"
90
91 student3 = StudentResult()
92 try:
93     student3.add_marks(10)
94 except ValueError as e:
95     print(str(e) == "Mark must be between 0 and 100")
96
97 try:
98     student3.add_marks(150)
99 except ValueError as e:
100     print(str(e) == "Mark must be between 0 and 100")
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
PS C:\AIAC> python 8.2_assignment.py
70.0
Pass
35.0
Fail
True
True
PS C:\AIAC>
```

- A fully functional StudentResult class that passes all AI-generated test

Task Description

Task 5 – Test-Driven Development for Username Validator

Requirements:

- Minimum length: 5 characters
- No spaces allowed
- Only alphanumeric characters Example Test Scenarios:

is_valid_username("user01") → True

is_valid_username("ai") → False

is_valid_username("user name") → False

is_valid_username("user@123") → False

Expected Output

The image shows a Visual Studio Code editor window with a Python file named `8.2_ass.py`. The script defines a function `is_valid_username` and includes test cases. A tooltip is visible over the function definition. Below the editor is a terminal window showing the execution of the script.

```
File Edit Selection View Go Run Terminal Help
8.2_ass.py
8.2_ass.py > is_valid_username
98 #Generate test cases for a function is_valid_username(username).
99 # Requirements:
100 # - Minimum length: 5 characters
101 # - No spaces allowed
102 # - Only alphanumeric characters
103 # - Return True or False
104 # - Use assert statements
105 def is_valid_username(username):
106     if not isinstance(username, str):
107         raise TypeError(method def isalnum() -> bool)
108     if len(username) < 5:
109         return False
110     if ' ' in username:
111         return False
112     if not username.isalnum():
113         return False
114     return True
115 # Test cases
116 # is_valid_username("user01") -> True
117 # is_valid_username("a") -> False
118 # is_valid_username("user name") -> False
119 # is_valid_username("user@123") -> False
120 print(is_valid_username("user01")) # True
121 print(is_valid_username("a")) # False
122 print(is_valid_username("user name")) # False
123 print(is_valid_username("user@123")) # False

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\AIAC> python 8.2_ass.py
True
False
False
False
PS C:\AIAC>
```

Ln 107, Col 39 Spaces: 4 UTF-8 CRLF Python Python 3.11 Go Live