

ASSIGNMENT-8.2

Name: J.Vyshnavi

HT. No: 2303A51895

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
```

PS C:\AIAC> python 8.2_ass.py
True
False
True
True
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("")) # ""

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

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 the Visual Studio Code interface. The left sidebar has a tree view with 'AIAC' expanded, showing files like 1.2_ass.py, 8.2_ass.py, app.db, ass.py, ASS1.PY, Assignment.py, Lab exam.py, Mon.py, and wed.py. The main editor area shows the code for 8.2_ass.py:

```
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 [] for empty list
38 # - Use assert statements
39 def sum_list(numbers):
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([])) # []
```

The bottom terminal tab shows the execution of the script:

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

PS C:\AIAC> python 8.2_ass.py
6
-6
6
0
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

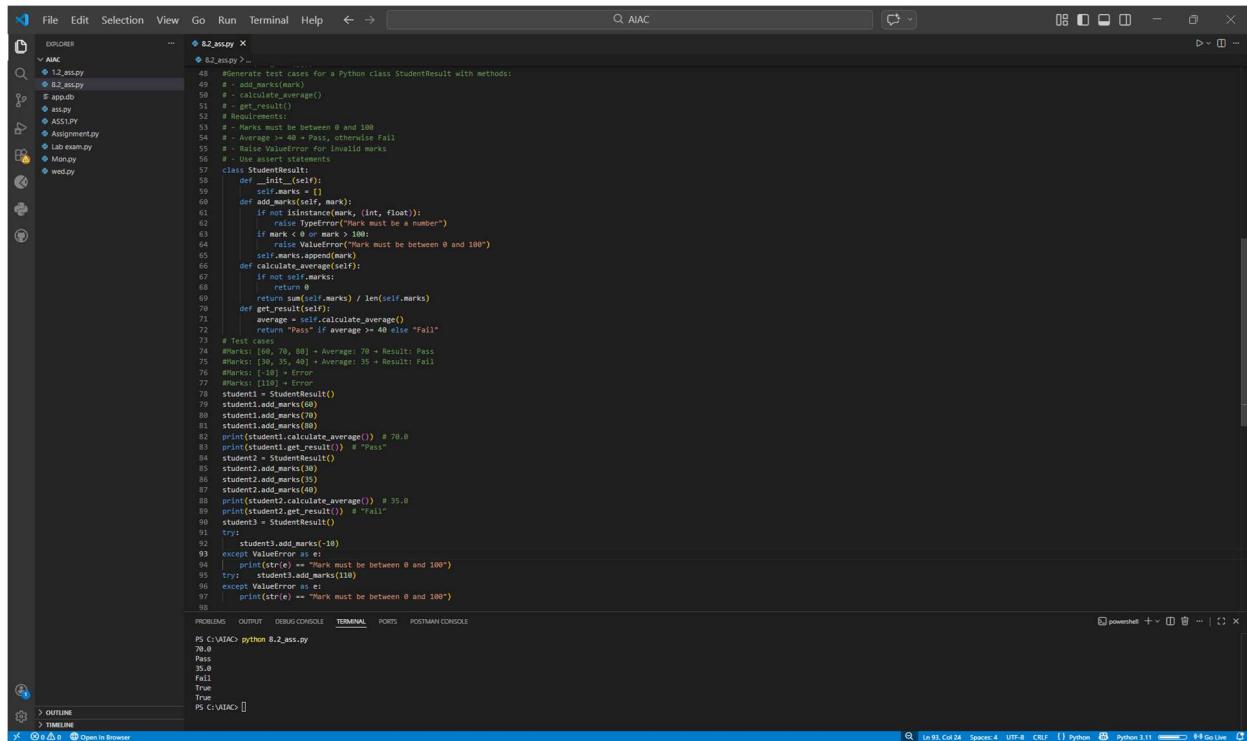
Example Test Scenarios:

Marks: [60, 70, 80] → Average: 70 → Result: Pass

Marks: [30, 35, 40] → Average: 35 → Result: Fail

Marks: [-10] → Error

Expected Output



```

8.2_ass.py
...
49     def __init__(self):
50         self.marks = []
51
52     def add_marks(self, mark):
53         if not isinstance(mark, (int, float)):
54             raise TypeError("Mark must be a number")
55         if mark < 0 or mark > 100:
56             raise ValueError("Mark must be between 0 and 100")
57         self.marks.append(mark)
58
59     def calculate_average(self):
60         if not self.marks:
61             return 0
62         return sum(self.marks) / len(self.marks)
63
64     def get_result(self):
65         average = self.calculate_average()
66
67         if average >= 40:
68             return "Pass"
69         else:
70             return "Fail"
71
72     def __str__(self):
73         return f"Student Result: {self.get_result()}"
74
75 @marks: [60, 70, 80] + Average: 70 + Result: Pass
76 @marks: [30, 35, 40] + Average: 35 + Result: Fail
77 @marks: [-10] + Error
78 student1 = StudentResult()
79 student1.add_marks(60)
80 student1.add_marks(70)
81 student1.add_marks(80)
82 print(student1.calculate_average()) # 70.0
83 print(student1.get_result()) # "Pass"
84 student2 = StudentResult()
85 student2.add_marks(30)
86 student2.add_marks(35)
87 student2.add_marks(40)
88 print(student2.calculate_average()) # 35.0
89 print(student2.get_result()) # "Fail"
90 student3 = StudentResult()
91 try:
92     student3.add_marks(10)
93 except ValueError as e:
94     print(str(e)) # "Mark must be between 0 and 100"
95 try:
96     student3.add_marks(110)
97 except ValueError as e:
98     print(str(e)) # "Mark must be between 0 and 100"

```

- 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 screenshot shows a Python script named `8.2_ass.py` open in a code editor. The code defines a function `is_valid_username` that checks if a given string is a valid username. It uses assertions to ensure the string has at least 5 characters, contains no spaces, and consists only of alphanumeric characters. The script then tests this function with several strings, printing the results to the terminal.

```
File Edit Selection View Go Run Terminal Help ← → 🔍 AIAC
```

```
8.2_ass.py x
8.2_ass.py > ⚙️ is_valid_username
98     #Generate test cases for a function is_valid_username(username).
99
100    # Requirements:
101    # - Minimum length: 5 characters
102    # - No spaces allowed
103    # - Only alphanumeric characters
104    # - Return True or False
105    # - Use assert statements
106
107    def is_valid_username(username):
108        if not isinstance(username, str):
109            raise TypeError(f"method def isalnum() -> bool")
110        if len(username) < 5: Return True if the string is an alpha-numeric string. False otherwise.
111        if ' ' in username: A string is alpha-numeric if all characters in the string are alpha-numeric and
112        return False there is at least one character in the string.
113        if not username.isalnum():
114            return False
115        return True
116
117    # Test cases
118    # is_valid_username("user01") -> True
119    # is_valid_username("ai") -> False
120    # is_valid_username("user name") -> False
121    # is_valid_username("user@123") -> False
122    print(is_valid_username("user01")) # True
123    print(is_valid_username("ai")) # False
124    print(is_valid_username("user name")) # False
125    print(is_valid_username("user@123")) # False
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

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