

ASSIGNMENT-8.2

Name: Y. Poojitha

HT. No: 2303A51499

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)`

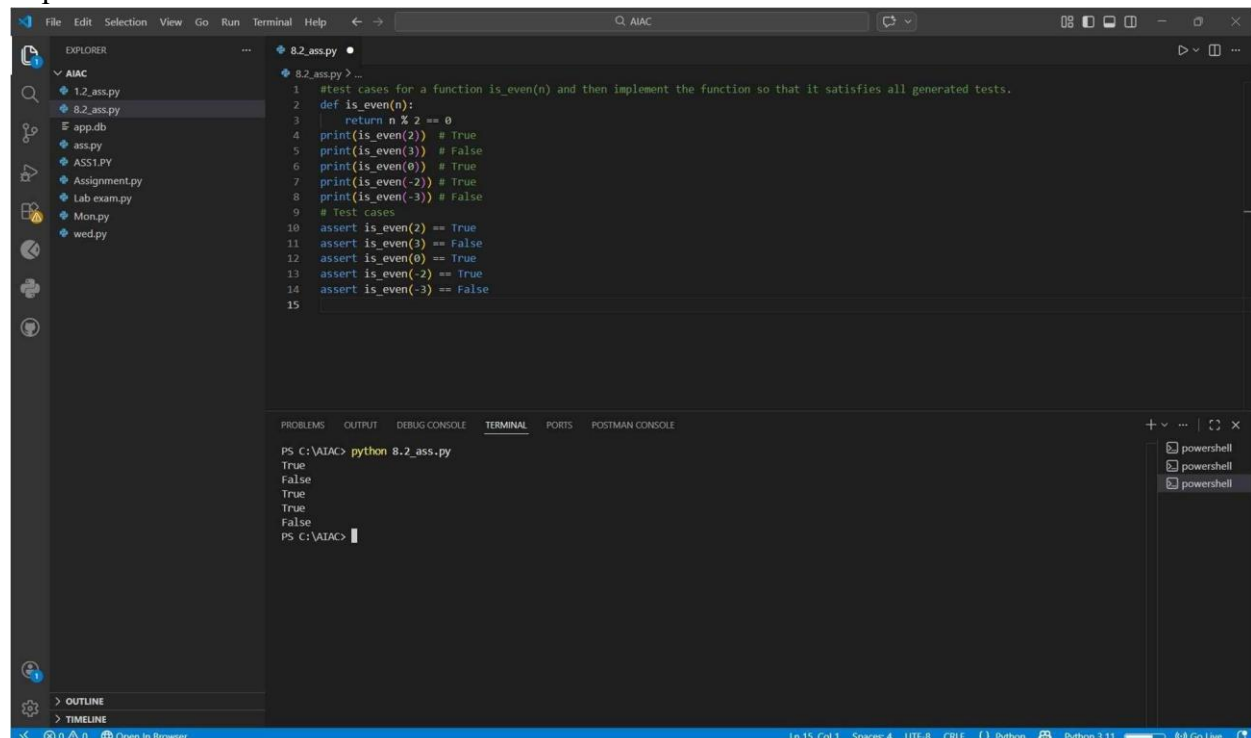
→ False `is_even(0)` → True

`is_even(-4)` →

True `is_even(9)` → False

Expected

Output



The screenshot shows a Visual Studio Code editor window with a file named `8.2_ass.py` open. The file contains the following Python code:

```
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
```

The terminal at the bottom shows the output of running `python 8.2_ass.py`:

```
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_lowercas

e(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:

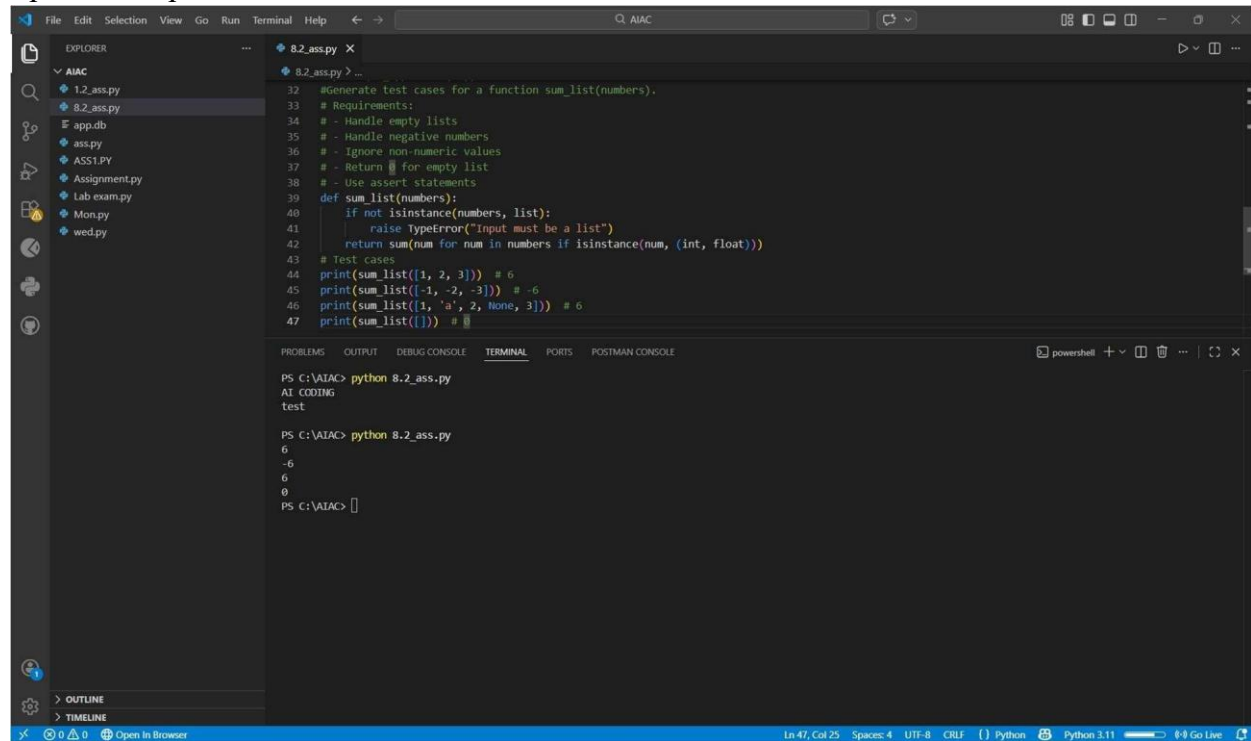
- Handle empty lists
 - Handle negative numbers
 - Ignore or safely handle non-numeric values
- Example Test Scenarios:

sum_list([1, 2, 3]) → 6

$\text{sum_list}([]) \rightarrow 0$ $\text{sum_list}([-1,$
 $5, -4]) \rightarrow 0$

`sum_list([2, "a", 3]) → 5`

Expected Output



```
File Edit Selection View Go Run Terminal Help
8.2_ass.py X
8.2_ass.py > ...
#Generate test cases for a function sum_list(numbers).
# Requirements:
# - Handle empty lists
# - Handle negative numbers
# - Ignore non-numeric values
# - Return 0 for empty list
# - Use assert statements
def sum_list(numbers):
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")
    return sum(num for num in numbers if isinstance(num, (int, float)))
# Test cases
print(sum_list([1, 2, 3])) # 6
print(sum_list([-1, -2, -3])) # -6
print(sum_list([1, 'a', 2, None, 3])) # 6
print(sum_list([])) # 0

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
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>

Ln 47, Col 25 Spaces: 4 UTF-8 CRLF Python Python 3.11 8:9 Go Live
```

- 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_re`

sult()

Requirements

:

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

otherwise Fail Example Test Scenarios:

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, mark):
55         if not isinstance(mark, (int, float)):
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(mark)
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: [40, 70, 80] = Average: 70 = Result: Pass
72 @marks: [30, 35, 40] = Average: 35 = Result: Fail
73 @marks: [-10] = Error
74 @marks: [110] = Error
75
76 student1 = StudentResult()
77 student1.add_marks(80)
78 student1.add_marks(70)
79 student1.calculate_average() # 75.0
80 print(student1.get_result()) # "Pass"
81
82 student2 = StudentResult()
83 student2.add_marks(90)
84 student2.add_marks(35)
85 student2.add_marks(40)
86 print(student2.calculate_average()) # 55.0
87 print(student2.get_result()) # "Fail"
88
89 student3 = StudentResult()
90 try:
91     student3.add_marks(-10)
92 except ValueError as e:
93     print(str(e) == "Mark must be between 0 and 100")
94 try:
95     student3.add_marks(110)
96 except ValueError as e:
97     print(str(e) == "Mark must be between 0 and 100")
98

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```

PS C:\VAIA> python s2_ass.py
70.0
Pass
35.0
Fail
True
True
PS C:\VAIA>

```

- 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 file named `8.2_ass.py`. The editor is displaying a Python function `is_valid_username` and its test cases. A tooltip is visible over the `isalnum()` method call, showing its signature: `(method) def isalnum() -> bool` and a description: "Return True if the string is an alpha-numeric string, False otherwise. A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string."

```
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
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
```

The bottom panel shows the terminal output of running `python 8.2_ass.py` in a PowerShell shell:

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

The status bar at the bottom indicates the current cursor position is at line 107, column 39, with 4 spaces, in UTF-8 encoding, using the Python interpreter.