

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

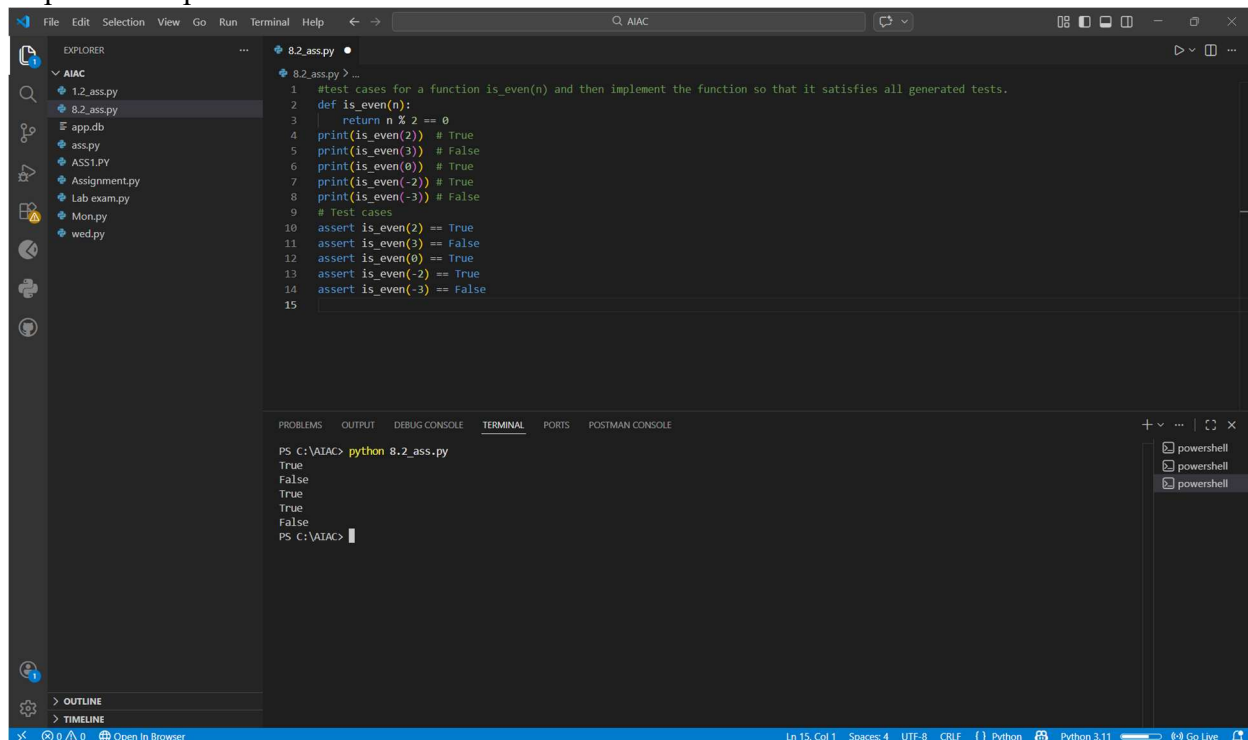
`is_even(7)` → False

`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
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("")) # ""
  
```

Terminal Output:

```

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:

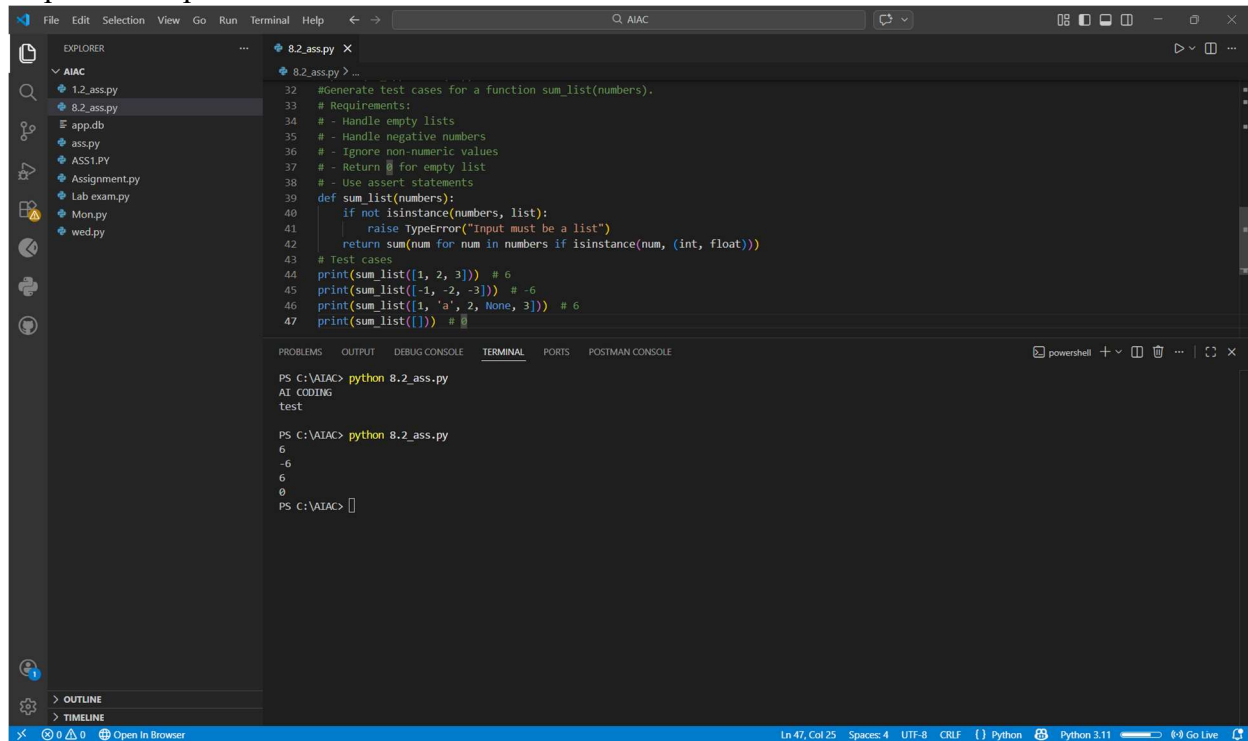
`sum_list([1, 2, 3])` → 6

`sum_list([])` → 0

`sum_list([-1, 5, -4])` → 0

`sum_list([2, "a", 3]) → 5`

Expected Output



```
File Edit Selection View Go Run Terminal Help
8.2_ass.py X
#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 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_result()`

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

```
48 #Generate test cases for a Python class StudentResult with methods:
49 # - add_marks(marks)
50 # - calculate_average()
51 # - get_result()
52 # Requirements:
53 # - Marks must be between 0 and 100
54 # - Average >= 40 - Pass, otherwise Fail
55 # - Raise ValueError for invalid marks
56 # - Use assert statements
57 class StudentResult:
58     def __init__(self):
59         self.marks = []
60     def add_marks(self, mark):
61         if not isinstance(mark, (int, float)):
62             raise TypeError("Mark must be a number")
63         if mark < 0 or mark > 100:
64             raise ValueError("Mark must be between 0 and 100")
65         self.marks.append(mark)
66     def calculate_average(self):
67         if not self.marks:
68             return 0
69         return sum(self.marks) / len(self.marks)
70     def get_result(self):
71         average = self.calculate_average()
72         return "Pass" if average >= 40 else "Fail"
73 # Test cases
74 #marks: [60, 70, 80] - Average: 70 - Result: Pass
75 #marks: [30, 35, 40] - Average: 35 - Result: Fail
76 #marks: [-10] - Error
77 #marks: [110] - 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")
99
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

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

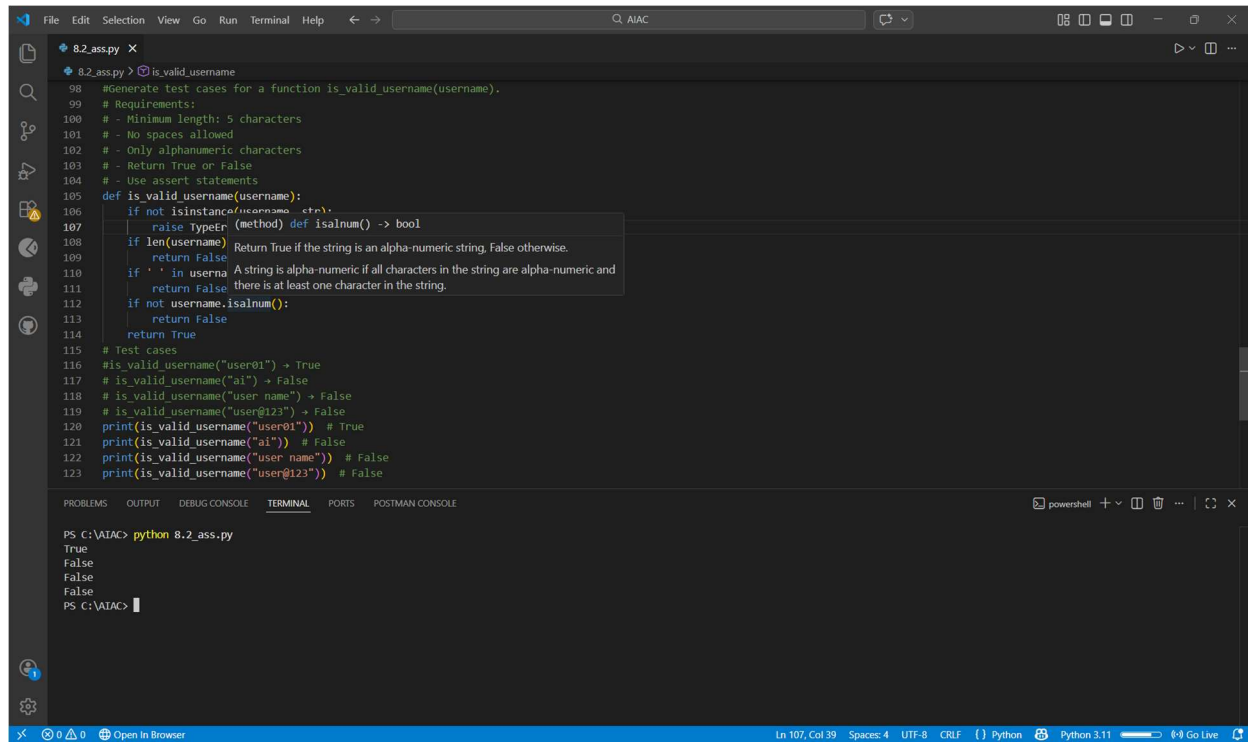
is_valid_username("user01") → True

is_valid_username("ai") → False

is_valid_username("user name") → False

is_valid_username("user@123") → False

Expected Output



```
File Edit Selection View Go Run Terminal Help
8.2_ass.py x
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("ai") -> 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("ai")) # 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