

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`

`→ False` `is_even(0) → True`

`is_even(-4) → True`

`is_even(9) → False`

Expected Output

```
EXPLORER          8.2_ass.py ●
AIAC
1.2_ass.py
8.2_ass.py
app.db
ass.py
ASS1.PY
Assignment.py
Lab_exam.py
Mon.py
wed.py

8.2_ass.py > ...
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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
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

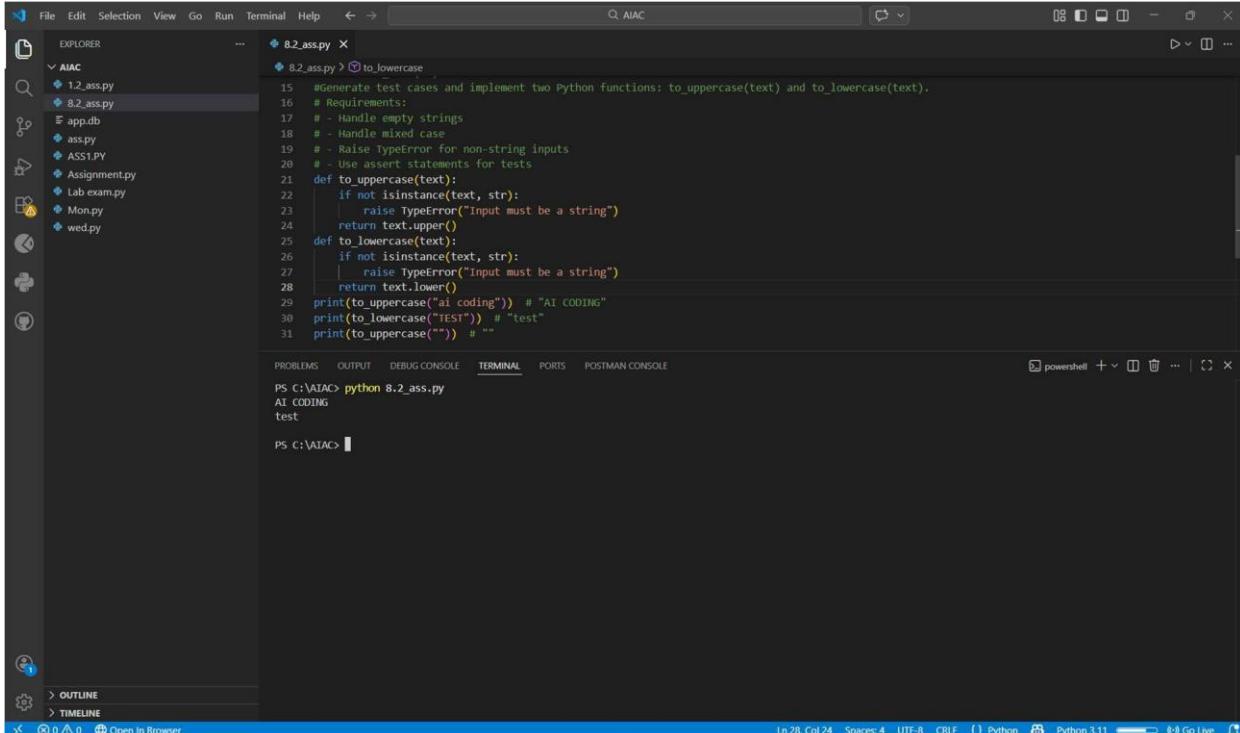
Task Description

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



```

File Edit Selection View Go Run Terminal Help ← → Q AIAC
EXPLORER ... 8.2_ass.py
AIAC
1.2_ass.py
8.2_ass.py
app.db
ass.py
ASS1.PY
Assignment.py
Lab exam.py
Mon.py
wed.py
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 with the following details:

- File Explorer:** Shows a folder named "AIAC" containing several Python files: 1.2_ass.py, 8.2_ass.py, app.db, ass.py, ASS1.PY, Assignment.py, Lab_exam.py, Mon.py, and wed.py.
- Code Editor:** The active file is 8.2_ass.py, which contains Python code for generating test cases for a sum_list function. The code includes comments explaining requirements like handling empty lists, negative numbers, and non-numeric values, and uses assert statements to validate the function's behavior.
- Terminal:** The terminal tab is active, showing the command "python 8.2_ass.py" being run in a PowerShell window. The output shows the results of the generated test cases: 6, -6, 6, 0, and an empty line.
- Status Bar:** Shows the current file path as "C:\VAIAC> 8.2_ass.py", line count as "Ln 47, Col 25", and other settings like "Spaces: 4", "UTF-8", and "Python 3.11".

- 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

```

File Edit Selection View Go Run Terminal Help ← →
AIAC
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\AIAC> python 8_2_ass.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 screenshot shows a code editor interface with a Python file named `8.2_ass.py`. The code defines a function `is_valid_username` that checks if a given string is a valid username based on specific requirements: at least 5 characters, no spaces, and only alphanumeric characters. The editor has a dark theme and includes a terminal window below the code editor showing the execution of the script and its output.

```
8.2_ass.py
98     # Requirements:
99     # - Minimum length: 5 characters
100    # - No spaces allowed
101    # - Only alphanumeric characters
102    # - Return True or False
103    # - Use assert statements
104
105   def is_valid_username(username):
106       if not isinstance(username, str):
107           raise TypeError(f"Method {isalnum} expected a string")
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
116   # Test cases
117   print(is_valid_username("user01")) # True
118   print(is_valid_username("ai")) # False
119   print(is_valid_username("user name")) # False
120   print(is_valid_username("user@123")) # False
121   print(is_valid_username("ai")) # False
122   print(is_valid_username("user name")) # False
123   print(is_valid_username("user@123")) # False
```

TERMINAL

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