

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 code editor interface with the following details:

- Explorer View:** Shows files in the AIAC folder, including `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 file `8.2_ass.py` 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
```

- Terminal:** The terminal window shows the command `python 8.2_ass.py` being run, followed by the output:

```
PS C:\AIAC> python 8.2_ass.py
True
False
True
True
False
PS C:\AIAC>
```
- Right Sidebar:** Contains a "powershell" section with three entries: "powershell", "powershell", and "powershell".

- 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

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

```

File Edt Selection View Go Run Terminal Help ← → ⌘ AIAC
EXPLORER 8.2_ass.py X
8.2_ass.py > to_lowercase
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 a code editor interface with a dark theme. The left sidebar has a tree view labeled 'EXPLORER' containing files like '1.2_ass.py', '8.2_ass.py', 'app.db', 'ass.py', 'ASS1.PY', 'Assignment.py', 'Lab exam.py', 'Mon.py', 'Mon.py', and 'wed.py'. The main area shows the content of '8.2_ass.py': a Python script that generates test cases for a function 'sum_list'. The script includes comments explaining requirements such as handling empty lists, negative numbers, non-numeric values, and empty lists. It uses assert statements to check the function's behavior for various inputs. Below the code editor is a terminal window showing the command 'python 8.2_ass.py' being run, followed by the output: '6', '-6', '6', '0'. At the bottom of the terminal window, status information is displayed: '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] → 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 < > Q AIAC
DOPLOER
AKM
B1_ass.py
B2_ass.py
ass.py
ASS1.PY
Assignment.py
Lab_exam.py
Mon.py
wed.py
8.2_ass.py X
8.2_ass.py > ...
47 # Complete test cases for a Python class StudentResult with methods:
48 # - add_marks(marks)
49 # - calculate_average()
50 # - get_result()
51 # - Result class
52 # - Results class
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 Result:
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
74 # Test cases
75 #Marks: [40, 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"
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\VSCode\python\8.2_ass.py
8.2_ass.py
Pass
35.0
Fail
True
True
True
PS C:\VSCode\]

```

- 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 window with a dark theme. The file being edited is `8.2_ass.py`. The code defines a function `is_valid_username` with the following logic:

```
def is_valid_username(username):
    if not isinstance(username, str):
        raise TypeError("method def isalnum() -> bool")
    if len(username) < 5:
        return False
    if ' ' in username:
        return False
    if not username.isalnum():
        return False
    return True
```

Test cases are provided at the bottom:

```
# Test cases
if is_valid_username("user01") == True:
    print("True")
else:
    print("False")
if is_valid_username("ai") == False:
    print("False")
else:
    print("True")
if is_valid_username("user name") == False:
    print("False")
else:
    print("True")
if is_valid_username("user@123") == False:
    print("False")
else:
    print("True")
if is_valid_username("user01") == True:
    print("True")
else:
    print("False")
if is_valid_username("ai") == False:
    print("False")
else:
    print("True")
if is_valid_username("user name") == False:
    print("False")
else:
    print("True")
if is_valid_username("user@123") == False:
    print("False")
else:
    print("True")
```

The terminal below shows the output of running the script:

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