

ASSIGNMENT-8.2

Name: V.Pranavith

HT. No: 2303A51488

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 the Visual Studio Code interface. The code editor displays a Python file named `8.2_ass.py` with the following content:

```
#test cases for a function is_even(n) and then implement the function so that it satisfies all generated tests.
def is_even(n):
    if n % 2 == 0:
        return True
    else:
        return False

# Test cases
assert is_even(2) == True
assert is_even(3) == False
assert is_even(0) == True
assert is_even(-2) == True
assert is_even(-3) == False
```

The terminal below shows the output of running the script:

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

- 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

The screenshot shows the Visual Studio Code interface. The code editor displays a Python file named `8.2_ass.py` with the following content:

```

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("")) # ""

```

The terminal below shows the execution of the script:

```

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

```

- 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 Explorer sidebar on the left lists 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'. The code editor window displays '8.2_ass.py' with Python code for a 'sum_list' function. The terminal window at the bottom shows the command 'python 8.2_ass.py' being run, followed by the output: '6 -6 6 0'. The status bar at the bottom right indicates the file is saved, the current line is 47, column 25, and the encoding is UTF-8.

```
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([])) # []
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\AIAC> python 8.2_ass.py
AI CODING
test
6
-6
6
0
PS C:\AIAC> []

Ln 47, Col 25 Spaces: 4 UFT-8 CRLF 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

```

6.2_ass.py
43 #import test class for a Python class StudentResult with methods:
44 # - add_marks(marks)
45 # - calculate_average()
46 # - get_result()
47 # - Result class
48 # Marks must be between 0 and 100
49 # - Average >= 40 = Pass, otherwise Fail
50 # - Raise ValueError for invalid marks
51 # - Add assert statements
52 class StudentResult:
53     def __init__(self):
54         self.marks = []
55     def add_marks(self, mark):
56         if not isinstance(mark, (int, float)):
57             raise TypeError("Mark must be a number")
58         if mark < 0 or mark > 100:
59             raise ValueError("Mark must be between 0 and 100")
60         self.marks.append(mark)
61     def calculate_average(self):
62         if not self.marks:
63             return None
64         return sum(self.marks) / len(self.marks)
65     def get_result(self):
66         average = self.calculate_average()
67         if average >= 40:
68             return "Pass"
69         else:
70             return "Fail"
71
72 # Test cases
73 #Marks: [60, 70, 80] + Average: 70 = Result: Pass
74 #Marks: [30, 15, 40] + Average: 35 = Result: Fail
75 #Marks: [-10] = Error
76 #Marks: [100, 100, 100] = Pass
77 student1 = StudentResult()
78 student1.add_marks(60)
79 student1.add_marks(70)
80 student1.add_marks(80)
81 print(student1.calculate_average()) # 70.0
82 print(student1.get_result()) # "Pass"
83 student2 = StudentResult()
84 student2.add_marks(30)
85 student2.add_marks(15)
86 student2.add_marks(40)
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(e) # Mark must be between 0 and 100
95 try:
96     student3.add_marks(110)
97 except ValueError as e:
98     print(e) # Mark must be between 0 and 100
99

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\AIAC> python 6.2_ass.py
6.0
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 dark-themed instance of Visual Studio Code (VS Code) with a Python file named `8.2_ass.py` open. The code defines a function `is_valid_username` that checks if a given string is a valid username based on specific requirements: it must be alpha-numeric, have a minimum length of 5 characters, and contain no spaces. The code includes several test cases at the bottom to demonstrate its functionality.

```
8.2_ass.py X
8.2_ass.py 1 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
106    def is_valid_username(username):
107        if not isinstance(username, str):
108            raise TypeError("method def isalnum() -> bool")
109        if len(username) < 5:
110            return False
111        if ' ' in username:
112            return False
113        if not username.isalnum():
114            return False
115        return True
116
117    #is_valid_username("user01") -> True
118    #is_valid_username("ai") -> False
119    #is_valid_username("user name") -> False
120    #is_valid_username("user@123") -> False
121    print(is_valid_username("user01")) # True
122    print(is_valid_username("ai")) # False
123    print(is_valid_username("user name")) # False
124    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>
```