ASSIGNMENT-8.2

Name-Anushka Voorugonda

H.no-2303A51176

Batch – 29

Task 1 – Test-Driven Development for Even/Odd Number Validator

Prompt :

Generate test cases for a function is_even(n) and then implement the function and print output using Assert test cases.

CODE:

```
Assignment_8.2.py > ...
1   #generate test cases for a function is_even(n) and then implement the function and print output using Assert test cases
2   def is_even(n):
3       return n % 2 == 0
4   # Test cases
5   assert is_even(4) == True, "Test case 1 failed"
6   assert is_even(7) == False, "Test case 2 failed"
7   assert is_even(0) == True, "Test case 3 failed"
8   assert is_even(-2) == True, "Test case 4 failed"
9   assert is_even(-3) == False, "Test case 5 failed"
10  print("All test cases passed!")
11
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding>  c:; cd 'c:\Users\hp\Or
13\python.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0
_Assited_Coding\Assignment_8.2.py'
All test cases passed!
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding>
```

JUSTIFICATION:

In this task, I used assert statements to test the is_even() function before finalizing the implementation. I checked positive numbers, negative numbers, and zero. The assert tests helped me verify correctness quickly. If any test failed, Python showed an error immediately. This made debugging simple and effective.

Task 2 – Test-Driven Development for String Case Converter

Prompt :

Generate test cases for two functions: to_uppercase(text) ,to_lowercase(text) using assert test cases.

CODE:

```python
#generate test cases for two functions: to_uppercase(text) ,to_lowercase(text) using assert test cases
def to_uppercase(text):
    return text.upper()
def to_lowercase(text):
    return text.lower()
# Test cases for to_uppercase
assert to_uppercase("hello") == "HELLO", "Test case 1 failed for to_uppercase"
assert to_uppercase("Python") == "PYTHON", "Test case 2 failed for to_uppercase"
assert to_uppercase("123") == "123", "Test case 3 failed for to_uppercase"
# Test cases for to_lowercase
assert to_lowercase("HELLO") == "hello", "Test case 1 failed for to_lowercase"
assert to_lowercase("Python") == "python", "Test case 2 failed for to_lowercase"
assert to_lowercase("123") == "123", "Test case 3 failed for to_lowercase"
print("All test cases passed for both functions!")
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding>  c:; cd 'c:\Users\hp\OneDrive\D
13\python.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x
_Assited_Coding\Assignment_8.2.py'
All test cases passed for both functions!
```

JUSTIFICATION:

I used assert statements to test both to_uppercase() and to_lowercase() functions. I tested normal strings, mixed-case strings, empty strings, and invalid inputs. The assertions ensured that the functions returned correct results. This approach helped me handle edge cases properly and improve code reliability.

Task 3 - Test-Driven Development for List Sum Calculator

Prompt :

Generate test cases for a function sum_list(numbers) that calculates the sum of list elements.

CODE:

```python
#generate test cases for a function sum_list(numbers) that calculates the sum of list elements.
def sum_list(numbers):
    return sum(numbers)
# Test cases
assert sum_list([1, 2, 3]) == 6, "Test case 1 failed"
assert sum_list([-1, 0, 1]) == 0, "Test case 2 failed"
assert sum_list([]) == 0, "Test case 3 failed"
assert sum_list([10, 20, 30]) == 60, "Test case 4 failed"
print("All test cases passed for sum_list function!")
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding>  c:; cd 'c:\Users\hp\OneDrive\Desk
13\python.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64'
_Assited_Coding\Assignment_8.2.py'
All test cases passed for sum_list function!
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding>
```

JUSTIFICATION:

For this task, I wrote assert test cases to check different scenarios like empty lists, negative numbers, and non-numeric values. The assertions confirmed whether the function calculated the correct sum. If any test failed, it showed an immediate error. This helped me build a more robust and safe function.

Task 4 - Test Cases for Student Result Class

Prompt :

Generate test cases for a StudentResult class with the following methods: add_marks(mark) ,calculate_average(),get_result() using assert test .

CODE:

```python
#Generate test cases for a StudentResult class with the following methods: add_marks(mark) ,calculate_average(),get_result() using assert
class StudentResult:
    def __init__(self):
        self.marks = []
    def add_marks(self, mark):
        self.marks.append(mark)
    def calculate_average(self):
        if not self.marks:
            return 0
        return sum(self.marks) / len(self.marks)
    def get_result(self):
        average = self.calculate_average()
        if average >= 90:
            return "A"
        elif average >= 80:
            return "B"
        elif average >= 70:
            return "C"
        else:
            return "F"
# Test cases
student = StudentResult()
student.add_marks(85)
student.add_marks(90)
assert student.calculate_average() == 87.5, "Test case 1 failed for calculate_average"
assert student.get_result() == "B", "Test case 1 failed for get_result"
student.add_marks(95)
assert student.calculate_average() == 90, "Test case 2 failed for calculate_average"
assert student.get_result() == "A", "Test case 2 failed for get_result"
student.add_marks(70)
assert student.calculate_average() == 88.33, "Test case 3 failed for calculate_average"
assert student.get_result() == "A", "Test case 3 failed for get_result"
print("All test cases passed for StudentResult class!")
```

OUTPUT:

```
13\python.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs
_Assited_Coding\Assignment_8.2.py'
All test cases passed for StudentResult class!
PS C:\Users\hp\OneDrive\Desktop\AI Assited Coding>
```

JUSTIFICATION:

I used assert statements to test average calculation and Pass/Fail logic. I verified that marks were within the valid range and handled invalid inputs. The test cases ensured that the class methods worked correctly. Using assertions made it easy to validate the logic step by step.

Task 5 - Test-Driven Development for Username Validator

Prompt :

Generate a python code to Test-Driven Development for Username Validator Requirements: Minimum length: 5 characters,No spaces allowed,Only alphanumeric characters using assert test cases

CODE:

```python
#Generate a python code to 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 5
# A username validation function that passes all AI-generated test cases using assert test cases.
def is_valid_username(username):
    if len(username) < 5:
        return False
    if " " in username:
        return False
    if not username.isalnum():
        return False
    return True
# Test cases
assert is_valid_username("user01") == True, "Test case 1 failed"
assert is_valid_username("ai") == False, "Test case 2 failed"
assert is_valid_username("user name") == False, "Test case 3 failed"
assert is_valid_username("user@123") == False, "Test case 4 failed"
assert is_valid_username("validUser") == True, "Test case 5 failed"
print("All test cases passed for is_valid_username function!")
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding>  c:; cd 'c:\Users\hp\OneDrive\D
13\python.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x
_Assited_Coding\Assignment_8.2.py'
All test cases passed for StudentResult class!
All test cases passed for is_valid_username function!
```

JUSTIFICATION:

In this task, I used assert statements to test username validation rules. I checked minimum length, no spaces, and only alphanumeric characters. The assertions helped confirm that valid usernames passed and invalid ones failed. This improved my understanding of input validation and testing logic.