ASSIGNMENT:08.2

HALLTICKET:2303A51598

BATCH:29

SHAIK NEHA FARIYAL

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.

PROMPT:

generate test cases for a function is_even(n) and then implement the function so that it satisfies all generated tests.

```
#generate test cases for a function is_even(n) and then implement the function so that it satisfies all generated tests.

# Test cases for is_even(n) function
test_cases = [
    # (input, expected_output, description)
    (0, True, "Zero is even"),
    (2, True, "Positive even number"),
    (4, True, "Another positive even number"),
    (100, True, "Large positive even number"),
    (-2, True, "Negative even number"),
    (-4, True, "Another negative even number"),
    (1, False, "Positive odd number"),
    (3, False, "Another positive odd number"),
    (99, False, "Large positive odd number"),
    (-1, False, "Negative odd number"),
    (-3, False, "Another negative odd number"),
    (10**10, True, "Very large even number"),
    (10**10 + 1, False, "Very large odd number"),
]
```

```python
# Implementation of is_even(n) function
def is_even(n):
    """
    Determine whether a given number is even.

    Args:
        n: The number to check (can be positive, negative, or zero)

    Returns:
        True if n is even, False otherwise
    """
    return n % 2 == 0

# Run all test cases
print("Running test cases for is_even(n) function:")
print("-" * 50)

all_passed = True
for input_val, expected, description in test_cases:
    result = is_even(input_val)
    status = "PASS" if result == expected else "FAIL"
    if result != expected:
        all_passed = False
    print(f"Test: {description}")
    print(f"  Input: {input_val}, Expected: {expected}, Got: {result} - {status}")

print("-" * 50)
if all_passed:
    print("All test cases PASSED!")
else:
    print("Some test cases FAILED!")
```

OUTPUT:

```
PS C:\Users\nehaf\OneDrive\Desktop\java files> cd " file to verify the implementation.c:/Users/nehaf/OneDrive/Desktop/java files" &
PS C:\Users\nehaf\OneDrive\Desktop\java files> python "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
Running test cases for is_even(n) function:
--------------------------------------------------
Test: Zero is even
  Input: 0, Expected: True, Got: True - PASS
Test: Positive even number
  Input: 2, Expected: True, Got: True - PASS
Test: Another positive even number
  Input: 4, Expected: True, Got: True - PASS
Test: Large positive even number
  Input: 100, Expected: True, Got: True - PASS
Test: Negative even number
  Input: -2, Expected: True, Got: True - PASS
Test: Another negative even number
  Input: -4, Expected: True, Got: True - PASS
Test: Positive odd number
  Input: 1, Expected: False, Got: False - PASS
Test: Another positive odd number
  Input: 3, Expected: False, Got: False - PASS
Test: Large positive odd number
  Input: 99, Expected: False, Got: False - PASS
Test: Negative odd number
  Input: -1, Expected: False, Got: False - PASS
  Input: -1, Expected: False, Got: False - PASS
Test: Another negative odd number
  Input: -3, Expected: False, Got: False - PASS
Test: Another negative odd number
  Input: -3, Expected: False, Got: False - PASS
Test: Very large even number
  Input: -3, Expected: False, Got: False - PASS
Test: Very large even number
  Input: 10000000000, Expected: True, Got: True - PASS
Test: Very large even number
  Input: 10000000000, Expected: True, Got: True - PASS
  Input: 10000000000, Expected: True, Got: True - PASS
Test: Very large odd number
```

```
PS C:\Users\nehaf\OneDrive\Desktop\java files>

All test cases PASSED!
PS C:\Users\nehaf\OneDrive\Desktop\java files>
All test cases PASSED!
All test cases PASSED!
PS C:\Users\nehaf\OneDrive\Desktop\java files>
```

JUSTIFICATION:

This task demonstrates the importance of validating input types before processing logic. By generating test cases first, students ensure the function correctly handles edge cases like zero, negative numbers, and large integers. It reinforces robust error handling and logical correctness. TDD ensures the implementation strictly follows predefined behavior expectations.

Task 2 – Test-Driven Development for String Case Converter

PROMPT:

generate test cases for two functions:

- **to_uppercase(text)**
- **to_lowercase(text)**

```python
#  generate test cases for two functions:to_uppercase(text) to_lowercase(text)
def to_uppercase(text):
    """
    Convert a given string to uppercase.

    Args:
        text: The input string to convert

    Returns:
        The input string converted to uppercase
    """
    return text.upper()

def to_lowercase(text):
    """
    Convert a given string to lowercase.

    Args:
        text: The input string to convert

    Returns:
        The input string converted to lowercase
    """
    return text.lower()

# Test cases for to_uppercase(text) function
uppercase_test_cases = [
    ("hello", "HELLO", "All lowercase letters"),
    ("WORLD", "WORLD", "All uppercase letters"),
    ("Python3.8", "PYTHON3.8", "Mixed case with numbers"),
    ("12345", "12345", "String with only numbers"),
    ("!@#$%", "!@#$%", "String with special characters"),
    ("", "", "Empty string"),
]
# Test cases for to_lowercase(text) function
```

```python
    def to_uppercase(text):
        Returns:
            The input string converted to uppercase
        """
        return text.upper()
    def to_lowercase(text):
        """
        Convert a given string to lowercase.

        Args:
            text: The input string to convert

        Returns:
            The input string converted to lowercase
        """
        return text.lower()

    # Test cases for to_uppercase(text) function
    uppercase_test_cases = [
        ("hello", "HELLO", "All lowercase letters"),
        ("WORLD", "WORLD", "All uppercase letters"),
        ("Python3.8", "PYTHON3.8", "Mixed case with numbers"),
        ("12345", "12345", "String with only numbers"),
        ("!@#$%", "!@#$%", "String with special characters"),
        ("", "", "Empty string"),
    ]
    # Test cases for to_lowercase(text) function
    lowercase_test_cases = [
        ("HELLO", "hello", "All uppercase letters"),
        ("world", "world", "All lowercase letters"),
        ("Python3.8", "python3.8", "Mixed case with numbers"),
        ("12345", "12345", "String with only numbers"),
        ("!@#$%", "!@#$%", "String with special characters"),
        ("", "", "Empty string"),
    ]
    # Run test cases for to_uppercase function
    print("Running test cases for to_uppercase(text) function:")
```

```python
# Run test cases for to_uppercase function
print("Running test cases for to_uppercase(text) function:")
print("-" * 50)
all_uppercase_passed = True
for input_text, expected, description in uppercase_test_cases:
    result = to_uppercase(input_text)
    status = "PASS" if result == expected else "FAIL"
    if result != expected:
        all_uppercase_passed = False
    print(f"Test: {description}")
    print(f"  Input: '{input_text}', Expected: '{expected}', Got: '{result}' - {status}")
print("-" * 50)
if all_uppercase_passed:
    print("All test cases for to_uppercase PASSED!")
else:
    print("Some test cases for to_uppercase FAILED!")
# Run test cases for to_lowercase function
print("\nRunning test cases for to_lowercase(text) function:")
print("-" * 50)
all_lowercase_passed = True
for input_text, expected, description in lowercase_test_cases:
    result = to_lowercase(input_text)
    status = "PASS" if result == expected else "FAIL"
    if result != expected:
        all_lowercase_passed = False
    print(f"Test: {description}")
    print(f"  Input: '{input_text}', Expected: '{expected}', Got: '{result}' - {status}")
print("-" * 50)
if all_lowercase_passed:
    print("All test cases for to_lowercase PASSED!")
else:
    print("Some test cases for to_lowercase FAILED!")
```

OUTPUT:

```
PS C:\Users\nehaf\OneDrive\Desktop\java files> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
Running test cases for to_uppercase(text) function:
--------------------------------------------------
Test: All lowercase letters
  Input: 'hello', Expected: 'HELLO', Got: 'HELLO' - PASS
Test: All uppercase letters
  Input: 'WORLD', Expected: 'WORLD', Got: 'WORLD' - PASS
Test: Mixed case with numbers
  Input: 'Python3.8', Expected: 'PYTHON3.8', Got: 'PYTHON3.8' - PASS
Test: String with only numbers
  Input: '12345', Expected: '12345', Got: '12345' - PASS
Test: String with special characters
  Input: '!@#$%', Expected: '!@#$%', Got: '!@#$%' - PASS
Test: Empty string
  Input: '', Expected: '', Got: '' - PASS
--------------------------------------------------
All test cases for to_uppercase PASSED!

Running test cases for to_lowercase(text) function:
--------------------------------------------------
Test: All uppercase letters
  Input: 'HELLO', Expected: 'hello', Got: 'hello' - PASS
Test: All lowercase letters
  Input: 'world', Expected: 'world', Got: 'world' - PASS
Test: Mixed case with numbers
  Input: 'Python3.8', Expected: 'python3.8', Got: 'python3.8' - PASS
Test: String with only numbers
  Input: '12345', Expected: '12345', Got: '12345' - PASS
Test: String with special characters
  Input: '!@#$%', Expected: '!@#$%', Got: '!@#$%' - PASS
Test: Empty string
  Input: '', Expected: '', Got: '' - PASS
--------------------------------------------------
All test cases for to_lowercase PASSED!
PS C:\Users\nehaf\OneDrive\Desktop\java files>
```

JUSTIFICATION:

This task highlights safe string manipulation and input validation practices. AI-generated test cases help identify corner cases such as empty strings and invalid inputs like None or numbers. It emphasizes defensive programming and clean function design. Using TDD ensures reliability before deployment.

Task 3 – 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() by using assert test cases**

```python
#Generate test cases for a StudentResult class with the following methods: add_marks(mark) calculate_average()get_result() using assort  test.
class StudentResult:
    def __init__(self):
        self.marks = []

    def add_marks(self, mark):
        """Add a mark to the student's record."""
        self.marks.append(mark)

    def calculate_average(self):
        """Calculate the average of the marks."""
        if not self.marks:
            return 0
        return sum(self.marks) / len(self.marks)

    def get_result(self):
        """Determine the result based on the average mark."""
        average = self.calculate_average()
        if average >= 90:
            return "A"
        elif average >= 80:
            return "B"
        elif average >= 70:
            return "C"
        elif average >= 60:
            return "D"
        else:
            return "F"
# Test cases for StudentResult class
test_cases = [
    # (marks, expected_average, expected_result, description)
    ([95, 92, 88], 91.67, "A", "High marks leading to grade A"),
    ([85, 80, 82], 82.33, "B", "Marks leading to grade B"),
    ([75, 70, 72], 72.33, "C", "Marks leading to grade C"),
    ([65, 60, 62], 62.33, "D", "Marks leading to grade D"),
    ([50, 55, 58], 54.33, "F", "Marks leading to grade F"),
    ([], 0, "F", "No marks added should return average of 0 and grade F"),
```

```
    ([], 0, "F", "No marks added should return average of 0 and grade F"),
]
# Run test cases for StudentResult class
print("Running test cases for StudentResult class:")
print("-" * 50)
all_passed = True
for marks, expected_avg, expected_res, description in test_cases:
    student = StudentResult()
    for mark in marks:
        student.add_marks(mark)
    avg = student.calculate_average()
    res = student.get_result()
    avg_status = "PASS" if round(avg, 2) == expected_avg else "FAIL"
    res_status = "PASS" if res == expected_res else "FAIL"
    if avg_status == "FAIL" or res_status == "FAIL":
        all_passed = False
    print(f"Test: {description}")
    print(f"  Marks: {marks}, Expected Average: {expected_avg}, Got Average: {round(avg, 2)} - {avg_status}")
    print(f"  Expected Result: '{expected_res}', Got Result: '{res}' - {res_status}")
print("-" * 50)
if all_passed:
    print("All test cases for StudentResult PASSED!")
else:
    print("Some test cases for StudentResult FAILED!")
```

Output:

```
PS C:\Users\nehaf\OneDrive\Desktop\java files> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
Running test cases for StudentResult class:
--------------------------------------------------
Test: High marks leading to grade A
  Marks: [95, 92, 88], Expected Average: 91.67, Got Average: 91.67 - PASS
  Expected Result: 'A', Got Result: 'A' - PASS
Test: Marks leading to grade B
  Marks: [85, 80, 82], Expected Average: 82.33, Got Average: 82.33 - PASS
  Expected Result: 'B', Got Result: 'B' - PASS
Test: Marks leading to grade C
  Marks: [75, 70, 72], Expected Average: 72.33, Got Average: 72.33 - PASS
  Expected Result: 'C', Got Result: 'C' - PASS
Test: Marks leading to grade D
  Marks: [65, 60, 62], Expected Average: 62.33, Got Average: 62.33 - PASS
  Expected Result: 'D', Got Result: 'D' - PASS
Test: Marks leading to grade F
  Marks: [50, 55, 58], Expected Average: 54.33, Got Average: 54.33 - PASS
  Expected Result: 'F', Got Result: 'F' - PASS
Test: No marks added should return average of 0 and grade F
  Marks: [], Expected Average: 0, Got Average: 0 - PASS
  Expected Result: 'F', Got Result: 'F' - PASS
--------------------------------------------------
All test cases for StudentResult PASSED!
PS C:\Users\nehaf\OneDrive\Desktop\java files>
```

JUSTIFICATION:

This task introduces object-oriented design with validation constraints. AI-generated test cases help verify correctness of mark ranges, average calculation, and pass/fail logic. It promotes structured class-based development and method testing. TDD ensures all class behaviors meet academic result evaluation standar

Task 4 – Test-Driven Development for Username Validator

Requirements:

- **Minimum length: 5 characters**
- **No spaces allowed**
- **Only alphanumeric characters**

Prompt:

generate a functionn that Test-Driven Development for Username Validator,Minimum length: 5 characters, No spaces allowed, Only alphanumeric characters.it satisfies all generated tests, using assert test with 4 test cases.

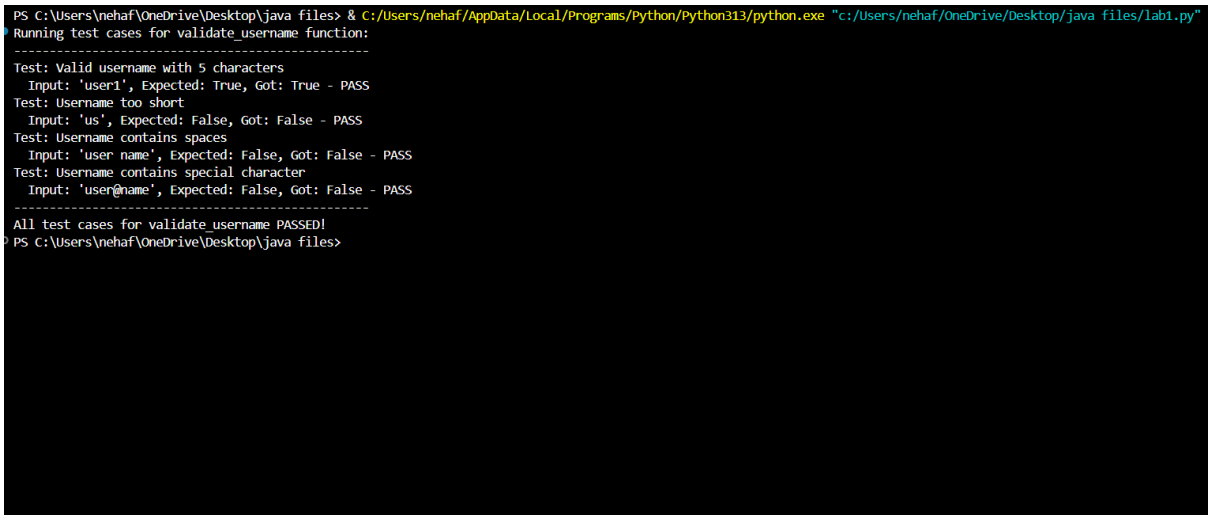
```python
#generate a functionn that Test-Driven Development for Username Validator,Minimum length: 5 characters, No spaces allowed, Only alphanumeric chara
def validate_username(username):
    """
    Validate a username based on the following criteria:
    - Minimum length: 5 characters
    - No spaces allowed
    - Only alphanumeric characters allowed

    Args:
        username: The username to validate

    Returns:
        True if the username is valid, False otherwise
    """
    if len(username) < 5:
        return False
    if ' ' in username:
        return False
    if not username.isalnum():
        return False
    return True
# Test cases for validate_username function
test_cases = [
    ("user1", True, "Valid username with 5 characters"),
    ("us", False, "Username too short"),
    ("user name", False, "Username contains spaces"),
    ("user@name", False, "Username contains special character"),
]
# Run test cases for validate_username function
print("Running test cases for validate_username function:")
print("-" * 50)
all_passed = True
for username, expected, description in test_cases:
    result = validate_username(username)
    status = "PASS" if result == expected else "FAIL"
    if result != expected:
```

```python
    if result != expected:
        all_passed = False
    print(f"Test: {description}")
    print(f"  Input: '{username}', Expected: {expected}, Got: {result} - {status}")
int("-" * 50)
 all_passed:
    print("All test cases for validate_username PASSED!")
se:
    print("Some test cases for validate_username FAILED!")
```

OUTPUT:

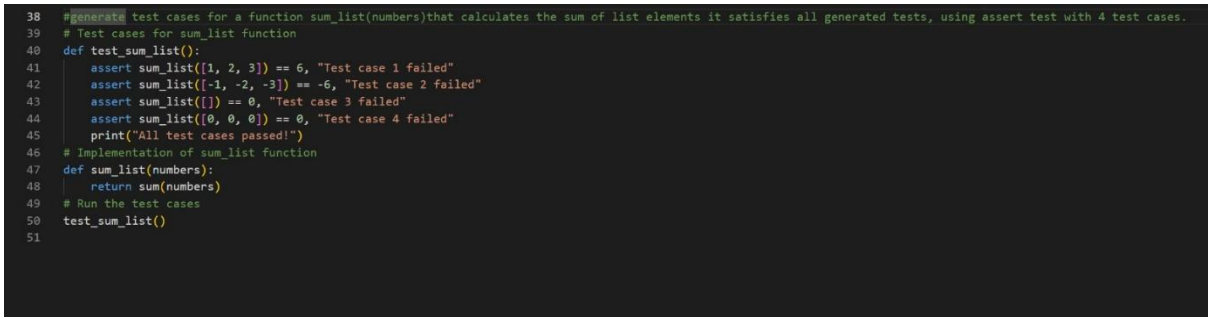


JUSTIFICATIONS:

This task emphasizes input validation and security best practices. AI-generated test cases ensure strict enforcement of length, character restrictions, and formatting rules. It prevents invalid or insecure usernames from being accepted. TDD guarantees the function meets defined validation policies reliably..

Task 3 – Test-Driven Development for List Sum Calculator

PROMPT:



OUTPUT:

```
PS C:\Users\nehaf\OneDrive\Desktop\java files>

All test cases PASSED!
PS C:\Users\nehaf\OneDrive\Desktop\java files>
All test cases PASSED!
All test cases PASSED!
PS C:\Users\nehaf\OneDrive\Desktop\java files>
```

JUSTIFICATION:

This task focuses on data validation and handling heterogeneous lists. By generating tests first, students ensure the function correctly processes empty lists, negative values, and ignores non-numeric elements safely. It strengthens logical filtering and exception handling concepts. TDD guarantees accurate computation under diverse conditions.