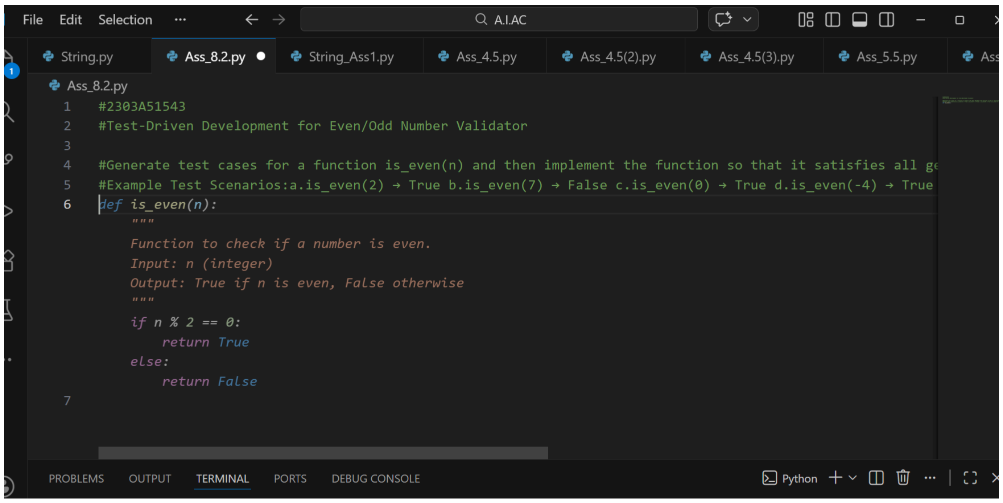
AI Assisted Coding Assignment-8.2

> 2303A51543 BT: 29 VELDI.HRUTHIKA 17.02.2026

💡 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 -1
• A correctly implemented is_even() function that passes all AI-generated test cases

```
#2303A51543
#Task-1:Test-Driven Development for Even/Odd Number Validator
#Generate test cases for a function is_even(n) and then implement the function so that it satisfies all generated tests. Requirements:i.Input->integer ii.Handle 0,negative numbers,large integers.
#Example Test Scenarios:a.is_even(2) → True b.is_even(7) → False c.is_even(0) → True d.is_even(-4) → True e.is_even(9) → False
def is_even(n):
    """
    Function to check if a number is even.
    Input: n (integer)
    Output: True if n is even, False otherwise
    """
    if n % 2 == 0:
        return True
    else:
        return False
# Test cases
print(is_even(2))   # Expected: True
print(is_even(7))   # Expected: False
print(is_even(0))   # Expected: True
print(is_even(-4))  # Expected: True
print(is_even(9))   # Expected: False
```



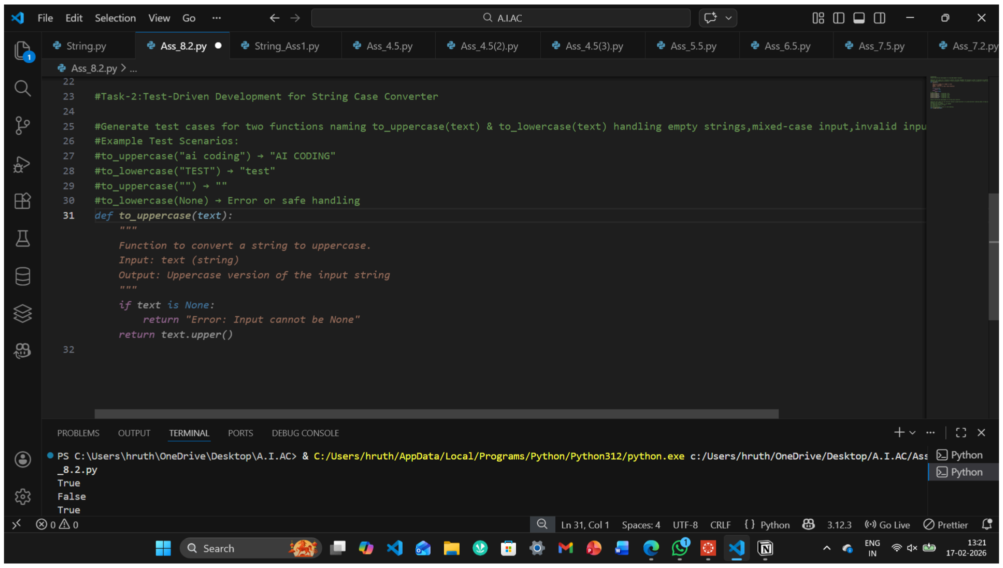
```
#2303A51543
#Test-Driven Development for Even/Odd Number Validator

#Generate test cases for a function is_even(n) and then implement the function so that it satisfies all ge
#Example Test Scenarios:a.is_even(2) → True b.is_even(7) → False c.is_even(0) → True d.is_even(-4) → True
def is_even(n):
    """
    Function to check if a number is even.
    Input: n (integer)
    Output: True if n is even, False otherwise
    """
    if n % 2 == 0:
        return True
    else:
        return False
```

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 -2
• Two string conversion functions that pass all AI-generated test cases with safe input handling.

```
#Task-2:Test-Driven Development for String Case Converter
#Generate test cases for two functions naming to_uppercase(text) & to_lowercase(text) han
dling empty strings,mixed-case input,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
def to_uppercase(text):
    """
    Function to convert a string to uppercase.
    Input: text (string)
    Output: Uppercase version of the input string
    """
    if text is None:
        return "Error: Input cannot be None"
    return text.upper()
```

```
def to_lowercase(text):
    """
Function to convert a string to lowercase.
Input: text (string)
Output: Lowercase version of the input string
    """
    if text is None:
        return "Error: Input cannot be None"
    return text.lower()
# Test cases
print(to_uppercase("ai coding"))  # Expected: "AI CODING"
print(to_lowercase("TEST"))       # Expected: "test"
print(to_uppercase(""))           # Expected: ""
print(to_lowercase(None))        # Expected: "Error: Input cannot be None"
```



```
#Task-2:Test-Driven Development for String Case Converter

#Generate test cases for two functions naming to_uppercase(text) & to_lowercase(text) handling empty strings,mixed-case input,invalid inpu
#Example Test Scenarios:
#to_uppercase("ai coding") → "AI CODING"
#to_lowercase("TEST") → "test"
#to_uppercase("") → ""
#to_lowercase(None) → Error or safe handling
def to_uppercase(text):
    """
    Function to convert a string to uppercase.
    Input: text (string)
    Output: Uppercase version of the input string
    """
    if text is None:
        return "Error: Input cannot be None"
    return text.upper()

```

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hruth/OneDrive/Desktop/A.I.AC/Ass
_8.2.py
True
False
True
```

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

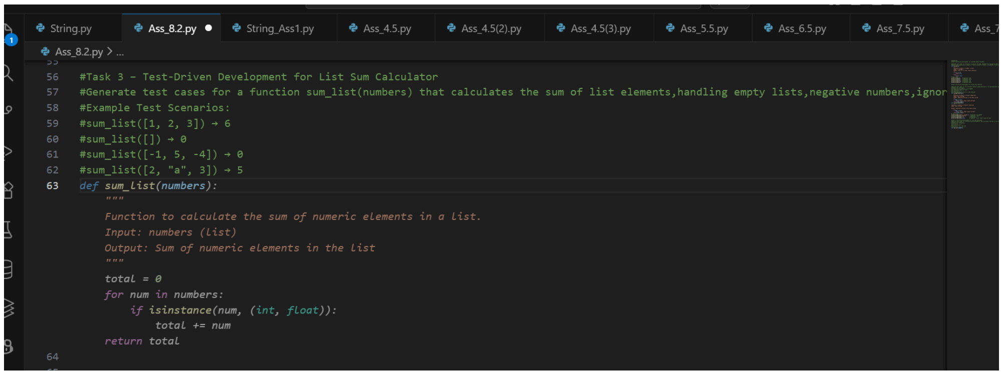
Example Test Scenarios:

sum_list([1, 2, 3]) → 6
sum_list([]) → 0
sum_list([-1, 5, -4]) → 0
sum_list([2, "a", 3]) → 5

Expected Output 3

• A robust list-sum function validated using AI-generated test cases.

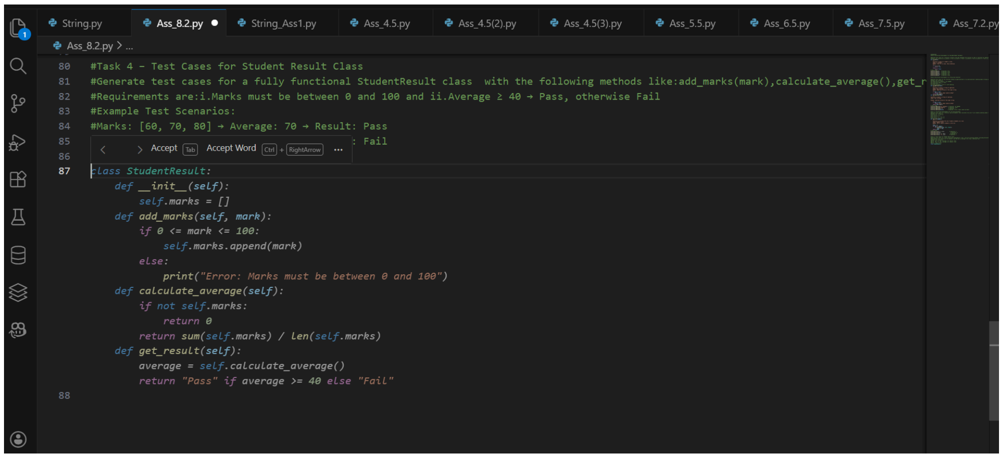
```
#Task 3 – Test-Driven Development for List Sum Calculator
#Generate test cases for a function sum_list(numbers) that calculates the sum of list ele
ments,handling empty lists,negative numbers,ignore non-numeric values.
#Example Test Scenarios:
#sum_list([1, 2, 3]) → 6
#sum_list([]) → 0
#sum_list([-1, 5, -4]) → 0
#sum_list([2, "a", 3]) → 5
def sum_list(numbers):
    """
    Function to calculate the sum of numeric elements in a list.
    Input: numbers (list)
    Output: Sum of numeric elements in the list
    """
    total = 0
    for num in numbers:
        if isinstance(num, (int, float)):
            total += num
    return total
# Test cases
print(sum_list([1, 2, 3]))          # Expected: 6
print(sum_list([]))                   # Expected: 0
print(sum_list([-1, 5, -4]))         # Expected: 0
print(sum_list([2, "a", 3]))        # Expected: 5
```



```
#Task 3 – Test-Driven Development for List Sum Calculator
#Generate test cases for a function sum_list(numbers) that calculates the sum of list elements,handling empty lists,negative numbers,ignor
#Example Test Scenarios:
#sum_list([1, 2, 3]) → 6
#sum_list([]) → 0
#sum_list([-1, 5, -4]) → 0
#sum_list([2, "a", 3]) → 5
def sum_list(numbers):
    """
    Function to calculate the sum of numeric elements in a list.
    Input: numbers (list)
    Output: Sum of numeric elements in the list
    """
    total = 0
    for num in numbers:
        if isinstance(num, (int, float)):
            total += num
    return total
```

💡 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 ≥ 40 → 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 -4
• A fully functional StudentResult class that passes all AI-generated test

```
#Task 4 – Test Cases for Student Result Class
#Generate test cases for a fully functional StudentResult class  with the following metho
ds like:add_marks(mark),calculate_average(),get_result()
#Requirements are:i.Marks must be between 0 and 100 and ii.Average ≥ 40 → 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
class StudentResult:
    def __init__(self):
        self.marks = []
    def add_marks(self, mark):
        if 0 <= mark <= 100:
            self.marks.append(mark)
        else:
            print("Error: Marks must be between 0 and 100")
    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()
        return "Pass" if average >= 40 else "Fail"
# Test cases
student = StudentResult()
student.add_marks(60)
student.add_marks(70)
student.add_marks(80)
print("Average:", student.calculate_average())  # Expected: 70.0
print("Result:", student.get_result())          # Expected: Pass
student = StudentResult()
student.add_marks(30)
student.add_marks(35)
student.add_marks(40)
print("Average:", student.calculate_average())  # Expected: 35.0
print("Result:", student.get_result())          # Expected: Fail
student = StudentResult()
```



```
#Task 4 – Test Cases for Student Result Class
#Generate test cases for a fully functional StudentResult class  with the following methods like:add_marks(mark),calculate_average(),get_r
#Requirements are:i.Marks must be between 0 and 100 and ii.Average ≥ 40 → Pass, otherwise Fail
#Example Test Scenarios:
#Marks: [60, 70, 80] → Average: 70 → Result: Pass
                                              : Fail

class StudentResult:
    def __init__(self):
        self.marks = []
    def add_marks(self, mark):
        if 0 <= mark <= 100:
            self.marks.append(mark)
        else:
            print("Error: Marks must be between 0 and 100")
    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()
        return "Pass" if average >= 40 else "Fail"
```

💡 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

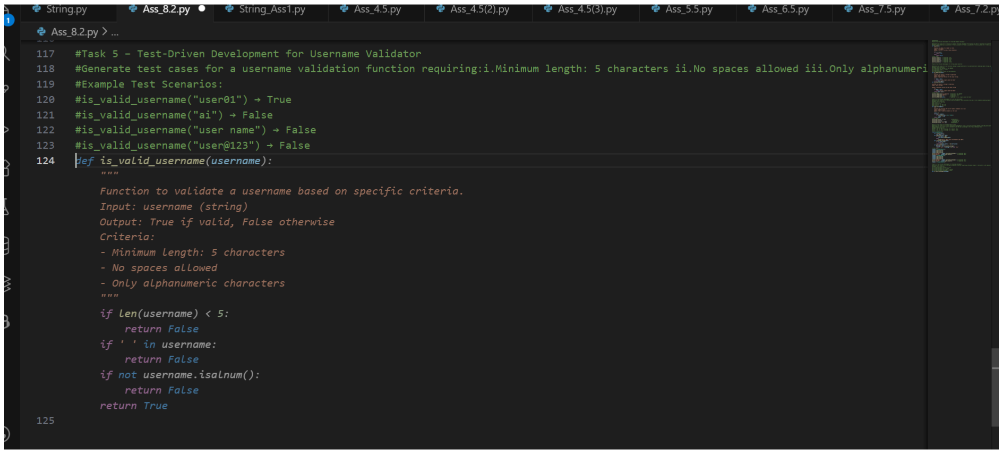
is_valid_username("user name") → False

is_valid_username("user@123") → False

Expected Output 5

A username validation function that passes all AI-generated test cases.

```
#Task 5 – Test-Driven Development for Username Validator
#Generate test cases for a username validation function requiring:i.Minimum length: 5 cha
racters ii.No spaces allowed iii.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
def is_valid_username(username):
    """
    Function to validate a username based on specific criteria.
    Input: username (string)
    Output: True if valid, False otherwise
    Criteria:
    - Minimum length: 5 characters
    - No spaces allowed
    - Only alphanumeric characters
    """
    if len(username) < 5:
        return False
    if ' ' in username:
        return False
    if not username.isalnum():
        return False
    return True
# Test cases
print(is_valid_username("user01"))     # Expected: True
print(is_valid_username("ai"))         # Expected: False
print(is_valid_username("user name"))  # Expected: False
print(is_valid_username("user@123"))  # Expected: False
```



```
#Task 5 - Test-Driven Development for Username Validator
#Generate test cases for a username validation function requiring:i.Minimum length: 5 characters ii.No spaces allowed iii.Only alphanumeri
#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
def is_valid_username(username):
    """
    Function to validate a username based on specific criteria.
    Input: username (string)
    Output: True if valid, False otherwise
    Criteria:
    - Minimum length: 5 characters
    - No spaces allowed
    - Only alphanumeric characters
    """
    if len(username) < 5:
        return False
    if ' ' in username:
        return False
    if not username.isalnum():
        return False
    return True
```

💡 DOCTEST:

```
python -m doctest -v Ass_8_2.py
True
False
True
True
False
AI CODING
test
Error: Input cannot be None
6
0
0
5
Average: 70.0
Result: Pass
Average: 35.0
Result: Fail
True
False
False
False
All tests passed successfully.
Trying:
is_even(2)
Expecting:
True
ok
Trying:
is_even(7)
Expecting:
False
ok
Trying:
is_even(0)
Expecting:
True
ok
Trying:
is_even(-4)
Expecting:
True
ok
Trying:
is_even(9)
Expecting:
False
ok
10 items had no tests:
Ass_8_2
```

```
Ass_8_2.StudentResult
Ass_8_2.StudentResult.init
Ass_8_2.StudentResult.add_marks
Ass_8_2.StudentResult.calculate_average
Ass_8_2.StudentResult.get_result
Ass_8_2.is_valid_username
Ass_8_2.sum_list
Ass_8_2.to_lowercase
Ass_8_2.to_uppercase
1 items passed all tests:
5 tests in Ass_8_2.is_even
5 tests in 11 items.
5 passed and 0 failed.
Test passed.
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
```

```
import pytest
from Ass_8_2 import is_even, to_uppercase, to_lowercase, sum_list, StudentResult, is_vali
d_username

# Test cases for Task 1 - Even/Odd Number Validator
def test_is_even():
    assert is_even(2) == True
    assert is_even(7) == False
    assert is_even(0) == True
    assert is_even(-4) == True
    assert is_even(9) == False

# Test cases for Task 2 - String Case Converter
def test_to_uppercase():
    assert to_uppercase("ai coding") == "AI CODING"
    assert to_uppercase("") == ""
    assert to_uppercase(None) == "Error: Input cannot be None"
    assert to_uppercase("Test") == "TEST"

def test_to_lowercase():
    assert to_lowercase("TEST") == "test"
    assert to_lowercase("") == ""
    assert to_lowercase(None) == "Error: Input cannot be None"
    assert to_lowercase("Test") == "test"

# Test cases for Task 3 - List Sum Calculator
def test_sum_list():
    assert sum_list([1, 2, 3]) == 6
    assert sum_list([]) == 0
    assert sum_list([-1, 5, -4]) == 0
    assert sum_list([2, "a", 3]) == 5
    assert sum_list([2, 3, -3, "a", 4]) == 6
    assert sum_list([100, -50, 20]) == 70
```

```python
# Test cases for Task 4 - StudentResult Class
def test_student_result():
    student = StudentResult()
    student.add_marks(60)
    student.add_marks(70)
    student.add_marks(80)
    assert student.calculate_average() == 70.0
    assert student.get_result() == "Pass"

    student = StudentResult()
    student.add_marks(30)
    student.add_marks(35)
    student.add_marks(40)
    assert student.calculate_average() == 35.0
    assert student.get_result() == "Fail"

# Test cases for Task 5 - Username Validator
def test_is_valid_username():
    assert is_valid_username("user01") == True
    assert is_valid_username("ai") == False
    assert is_valid_username("user name") == False
    assert is_valid_username("user@123") == False
    assert is_valid_username("validUser") == True
    assert is_valid_username("us") == False
```

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> pytest Ass_8_2.py
====================================================== test session starts
======================================================
platform win32 -- Python 3.12.3, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\hruth\OneDrive\Desktop\A.I.AC
rootdir: C:\Users\hruth\OneDrive\Desktop\A.I.AC
collected 6 items
Ass_8_2.py ......                                                    [100%]
======================================================= 6 passed in 0.09s
=======================================================
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> ^C
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
```