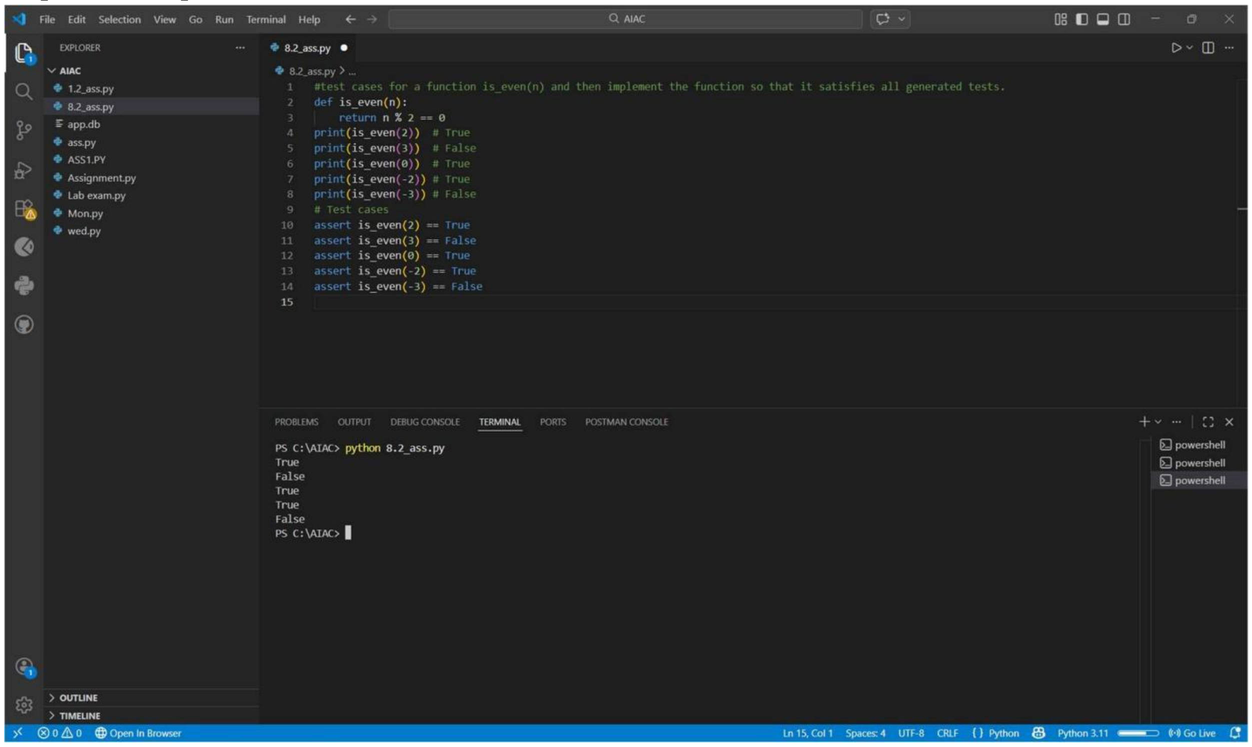
ASSIGNMENT-8.2

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

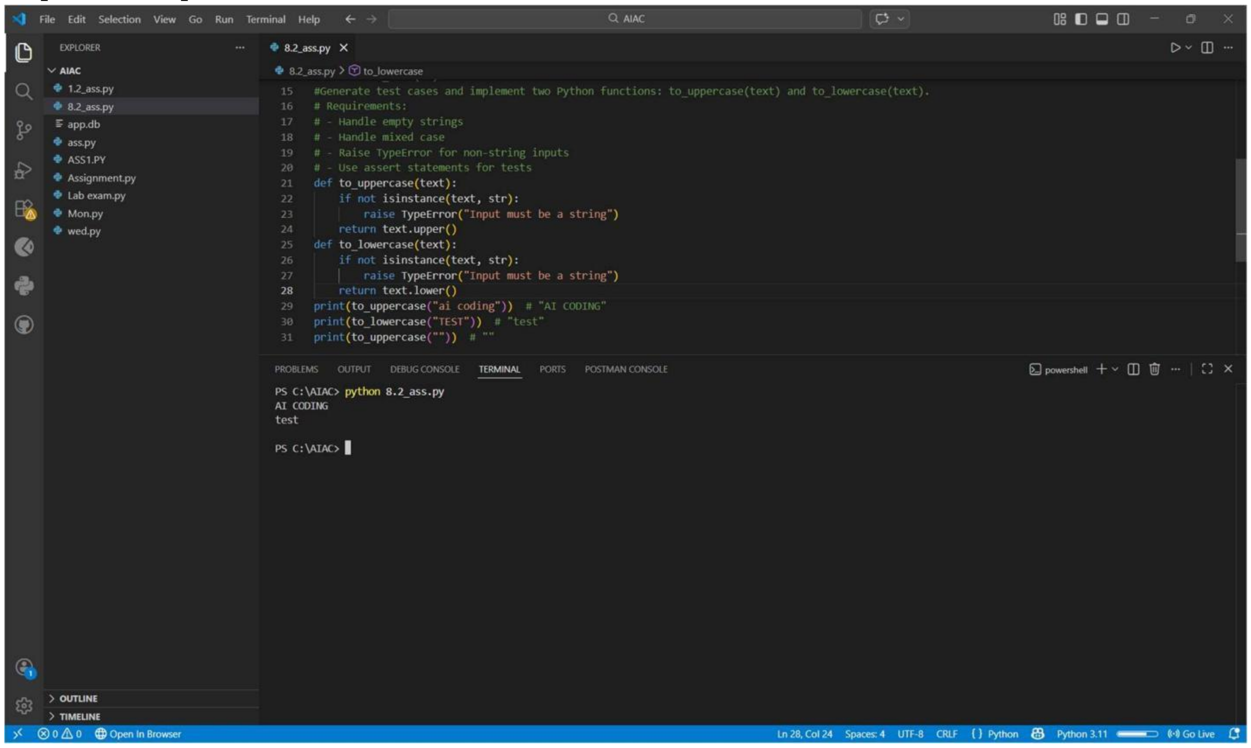


```python
#test cases for a function is_even(n) and then implement the function so that it satisfies all generated tests.
def is_even(n):
    return n % 2 == 0
print(is_even(2))  # True
print(is_even(3))  # False
print(is_even(0))  # True
print(is_even(-2)) # True
print(is_even(-3)) # False
# Test cases
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
```

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

• 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



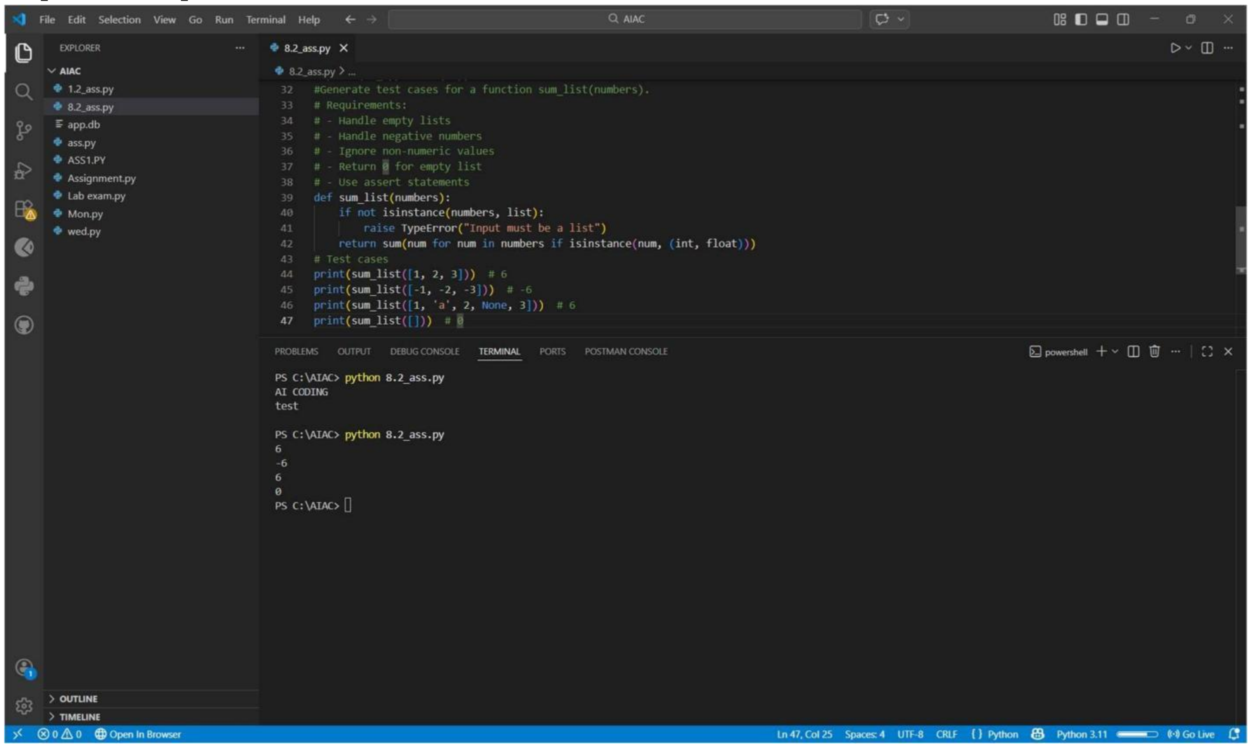
```python
#Generate test cases and implement two Python functions: to_uppercase(text) and to_lowercase(text).
# Requirements:
# - Handle empty strings
# - Handle mixed case
# - Raise TypeError for non-string inputs
# - Use assert statements for tests
def to_uppercase(text):
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return text.upper()
def to_lowercase(text):
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return text.lower()
print(to_uppercase("ai coding"))  # "AI CODING"
print(to_lowercase("TEST"))  # "test"
print(to_uppercase(""))  # ""
```

```
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



```python
#Generate test cases for a function sum_list(numbers).
# Requirements:
# - Handle empty lists
# - Handle negative numbers
# - Ignore non-numeric values
# - Return 0 for empty list
# - Use assert statements
def sum_list(numbers):
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")
    return sum(num for num in numbers if isinstance(num, (int, float)))
# Test cases
print(sum_list([1, 2, 3]))  # 6
print(sum_list([-1, -2, -3]))  # -6
print(sum_list([1, 'a', 2, None, 3]))  # 6
print(sum_list([]))  # 0
```

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

PS C:\AIAC> python 8.2_ass.py
6
-6
6
0
PS C:\AIAC>
```

• 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()

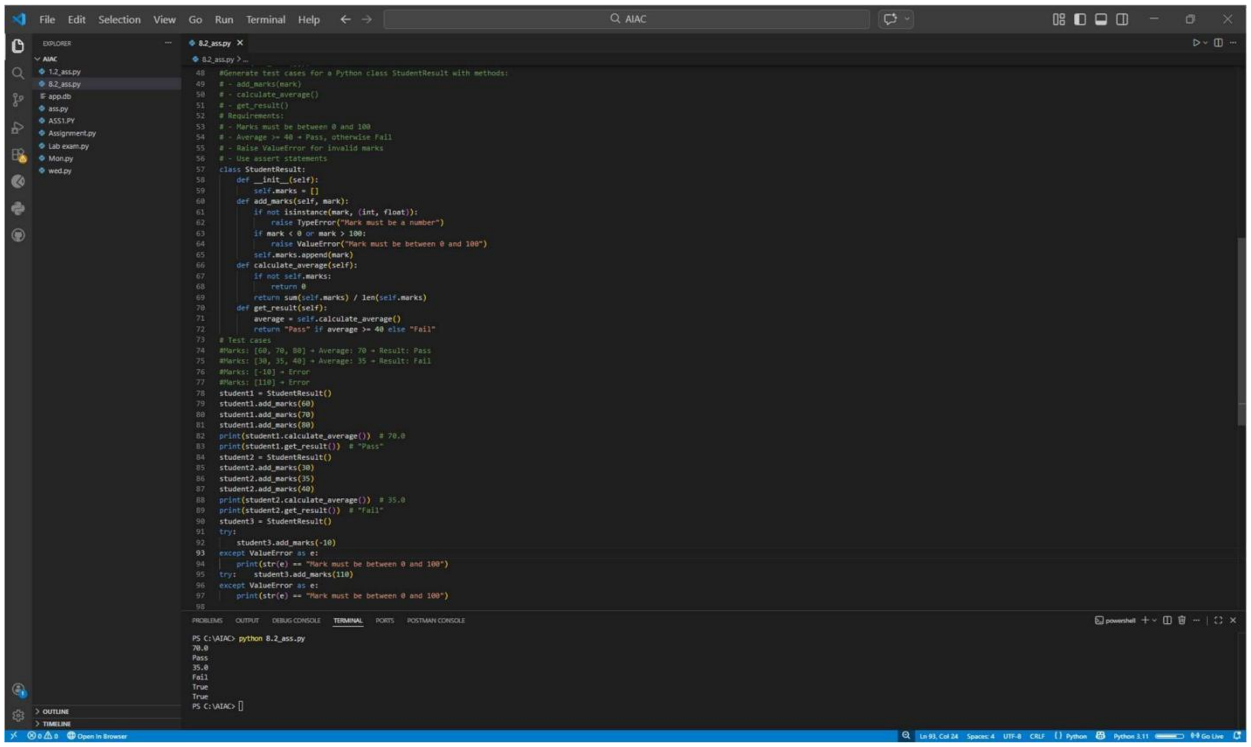
Requirements:

• Marks must be between 0 and 100
• Average ≥ 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



```python
#Generate test cases for a Python class StudentResult with methods:
# - add_marks(mark)
# - calculate_average()
# - get_result()
# Requirements:
# - Marks must be between 0 and 100
# - Average >= 40 → Pass, otherwise Fail
# - Raise ValueError for invalid marks
# - Use assert statements
class StudentResult:
    def __init__(self):
        self.marks = []
    def add_marks(self, mark):
        if not isinstance(mark, (int, float)):
            raise TypeError("Mark must be a number")
        if mark < 0 or mark > 100:
            raise ValueError("Mark must be between 0 and 100")
        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()
        return "Pass" if average >= 40 else "Fail"
# Test cases
#Marks: [60, 70, 80] → Average: 70 → Result: Pass
#Marks: [30, 35, 40] → Average: 35 → Result: Fail
#Marks: [-10] → Error
#Marks: [110] → Error
student1 = StudentResult()
student1.add_marks(60)
student1.add_marks(70)
student1.add_marks(80)
print(student1.calculate_average())  # 70.0
print(student1.get_result())  # "Pass"
student2 = StudentResult()
student2.add_marks(30)
student2.add_marks(35)
student2.add_marks(40)
print(student2.calculate_average())  # 35.0
print(student2.get_result())  # "Fail"
student3 = StudentResult()
try:
    student3.add_marks(-10)
except ValueError as e:
    print(str(e) == "Mark must be between 0 and 100")
try:    student3.add_marks(110)
except ValueError as e:
    print(str(e) == "Mark must be between 0 and 100")
```

```
PS C:\AIAC> python 8.2_ass.py
70.0
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:

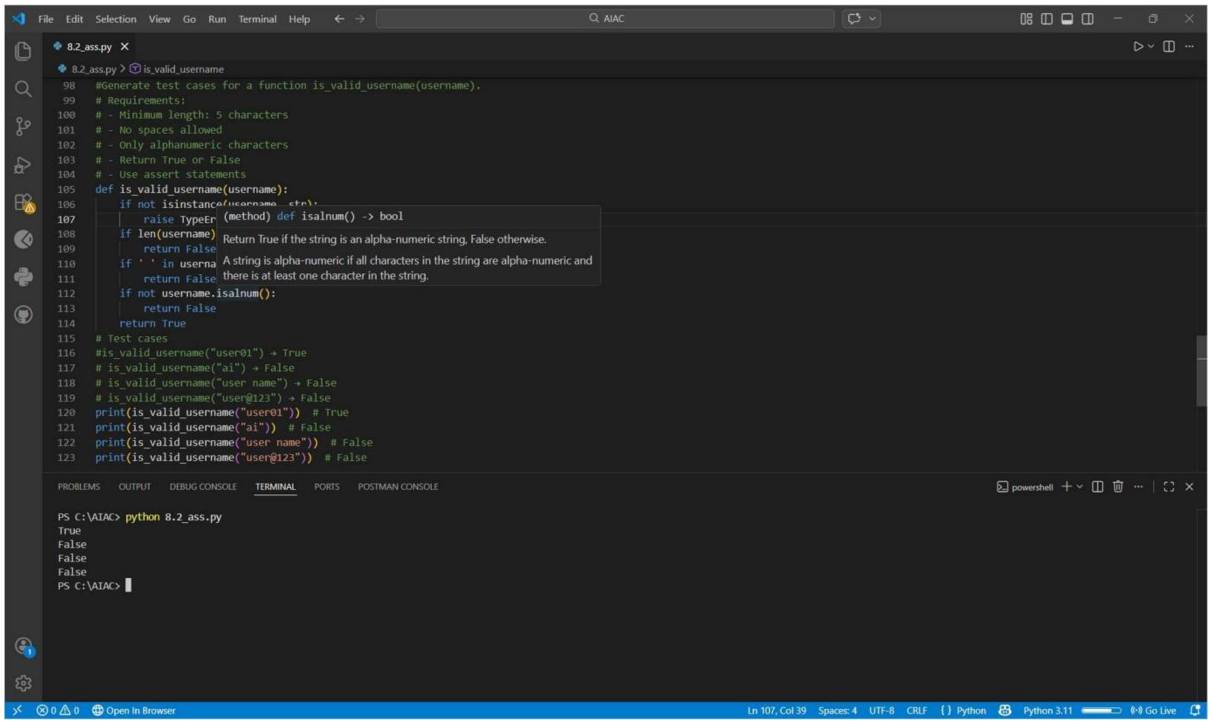
is_valid_username("user01") → True

is_valid_username("ai") → False

is_valid_username("user name") → False

is_valid_username("user@123") → False

Expected Output



```python
#Generate test cases for a function is_valid_username(username).
# Requirements:
# - Minimum length: 5 characters
# - No spaces allowed
# - Only alphanumeric characters
# - Return True or False
# - Use assert statements
def is_valid_username(username):
    if not isinstance(username, str):
        raise TypeEr
    if len(username)
        return False
    if ' ' in userna
        return False
    if not username.isalnum():
        return False
    return True
# Test cases
#is_valid_username("user01") → True
# is_valid_username("ai") → False
# is_valid_username("user name") → False
# is_valid_username("user@123") → False
print(is_valid_username("user01"))  # True
print(is_valid_username("ai"))  # False
print(is_valid_username("user name"))  # False
print(is_valid_username("user@123"))  # False
```

(method) def isalnum() -> bool

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

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