Lab Assignment # 08

Course Title   : AI Assistant Coding

Name of Student   : V.SUSHMITHA
Enrollment No.   : 2303A54055
Batch No.   : 47-b

Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases

Task Description #1 (Password Strength Validator – Apply AI in Security Context)
• Task: Apply AI to generate at least 3 assert test cases for is_strong_password(password) and implement the validator function.

• Requirements:
o Password must have at least 8 characters.
o Must include uppercase, lowercase, digit, and special character.
o Must not contain spaces.

Example Assert Test Cases:
     assert is_strong_password("Abcd@123") == True
     assert is_strong_password("abcd123") == False
     assert is_strong_password("ABCD@1234") == True

Expected Output #1: • Password validation logic passing all AI-generated test cases.

Output Screenshot:

```
lab8.1 > task01.py > ...
  1   # Generate at least 5 assert-based test cases for a function:
  2   # def is_strong_password(password):
  3   # Requirements:
  4   # - At least 8 characters
  5   # - Must contain uppercase letter
  6   # - Must contain lowercase letter
  7   # - Must contain digit
  8   # - Must contain special character
  9   # - Must not contain spaces
 10   # Include edge cases such as:
 11   # - Empty string
 12   # - Only digits
 13   # - Only uppercase
 14   # - Password with spaces
 15   # - Exactly 8 characters valid case
 16   # Return only assert statements.
 17   # Now implement the Python function:
 18   # def is_strong_password(password):
 19   # Make sure it passes all previously generated test cases.
 20   # Use clean logic and proper condition checking.
 21   # Do not use external libraries.
 22   def is_strong_password(password):
 23       if not isinstance(password, str):
 24           return False
 25
 26       if len(password) < 8:
 27           return False
 28
 29       if ' ' in password:
 30           return False
 31
 32       if not any(char.isupper() for char in password):
 33           return False
 34
 35       if not any(char.islower() for char in password):
 36           return False
 37
 38       if not any(char.isdigit() for char in password):
 39           return False
 40
 41       if not any(char in "!@#$%^&*()-_=+[]{}|;:'\",.<>?/" for char in password):
 42           return False
 43
```



```
lab8.1 > task01.py > ...
 47   # Test Cases
 48   assert is_strong_password("Password123!") == True
 49   assert is_strong_password("") == False
 50   assert is_strong_password("12345678") == False
 51   assert is_strong_password("PASSWORD") == False
 52   assert is_strong_password("Pass word1!") == False
 53   assert is_strong_password("Passw1!") == False
 54   assert is_strong_password("Passw0rd!") == True
 55   assert is_strong_password("P@ssw0rd") == True
 56   assert is_strong_password("P@ss w0rd") == False
 57   assert is_strong_password("P@ssw0") == False
 58
 59   print("All test cases passed successfully ✅")
 60
 61   # Analyze the password validation function.
 62   # Suggest improvements for:
 63   # - Readability
 64   # - Security
 65   # - Edge case handling
 66   # - Performance
 67
 68   # Explain briefly.
 69   # The function is_strong_password is straightforward and checks all the required conditions for a strong password.
 70   # Improvements:
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
/usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task01.py
● anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task01.py
  All test cases passed successfully ✅
⟲ anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab %
```

Explanation: This checks if a password is truly "strong" by making sure it has the right mix of characters and no spaces.

Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)

• **Task: Use AI to generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.**
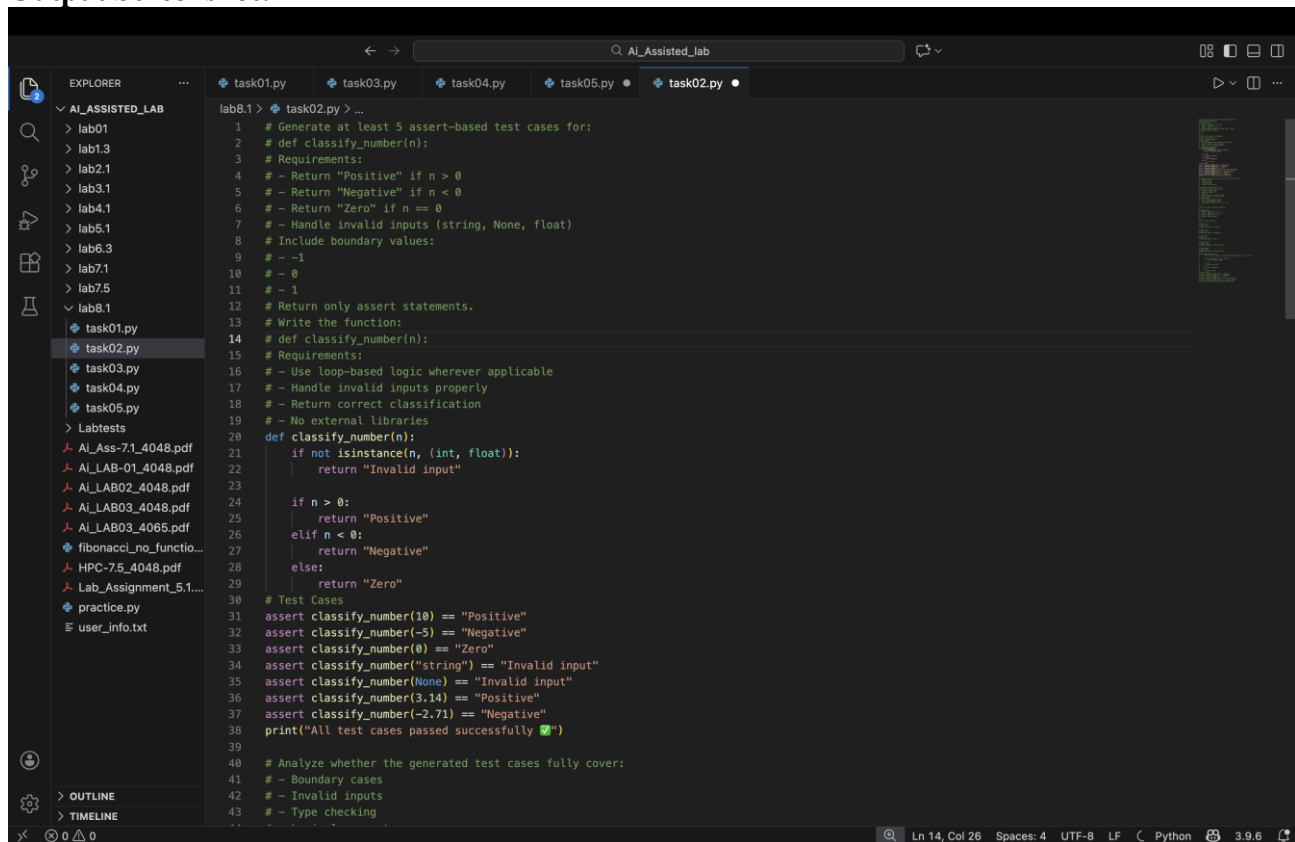
• Requirements:

o Classify numbers as Positive, Negative, or Zero.
o Handle invalid inputs like strings and None.
o Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:
    assert classify_number(10) == "Positive"
    assert classify_number(-5) == "Negative"
    assert classify_number(0) == "Zero"

Expected Output #2: • Classification logic passing all assert tests.

Output Screenshot:


```python
# Generate at least 5 assert-based test cases for:
# def classify_number(n):
# Requirements:
# - Return "Positive" if n > 0
# - Return "Negative" if n < 0
# - Return "Zero" if n == 0
# - Handle invalid inputs (string, None, float)
# Include boundary values:
# - -1
# - 0
# - 1
# Return only assert statements.
# Write the function:
# def classify_number(n):
# Requirements:
# - Use loop-based logic wherever applicable
# - Handle invalid inputs properly
# - Return correct classification
# - No external libraries
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid input"

    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
# Test Cases
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number("string") == "Invalid input"
assert classify_number(None) == "Invalid input"
assert classify_number(3.14) == "Positive"
assert classify_number(-2.71) == "Negative"
print("All test cases passed successfully ✅")

# Analyze whether the generated test cases fully cover:
# - Boundary cases
# - Invalid inputs
# - Type checking
```

Explanation: This sorts numbers into positive, negative, or zero, while politely rejecting anything that isn't a number.

Task Description #3 (Anagram Checker – Apply AI for String Analysis)

• **Task: Use AI to generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.**

• **Requirements:**
    o Ignore case, spaces, and punctuation.
    o Handle edge cases (empty strings, identical words).

Example Assert Test Cases:
    assert is_anagram("listen", "silent") == True
    assert is_anagram("hello", "world") == False
    assert is_anagram("Dormitory", "Dirty Room") == True

Expected Output #3: • Function correctly identifying anagrams and passing all AI-generated tests.

Output Screenshot:

Explanation: This spots whether two words or phrases are made of the same letters, ignoring case, spaces, and punctuation.

Task Description #4 (Inventory Class – Apply AI to Simulate Real- World Inventory System)

• **Task: Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.**

• **Methods:**
    o add_item(name, quantity)
    o remove_item(name, quantity)
    o get_stock(name)

Example Assert Test Cases:
    inv = Inventory()

    inv.add_item("Pen", 10)
    assert inv.get_stock("Pen") == 10
    inv.remove_item("Pen", 5)
    assert inv.get_stock("Pen") == 5
    inv.add_item("Book", 3)
    assert inv.get_stock("Book") == 3

Expected Output #4: • Fully functional class passing all assertions.

Output Screenshot:

Explanation: This simulates a mini store system where you can add, remove, and check stock for items.

Task Description #5 (Date Validation & Formatting – Apply AI for Data Validation)

• **Task: Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.**
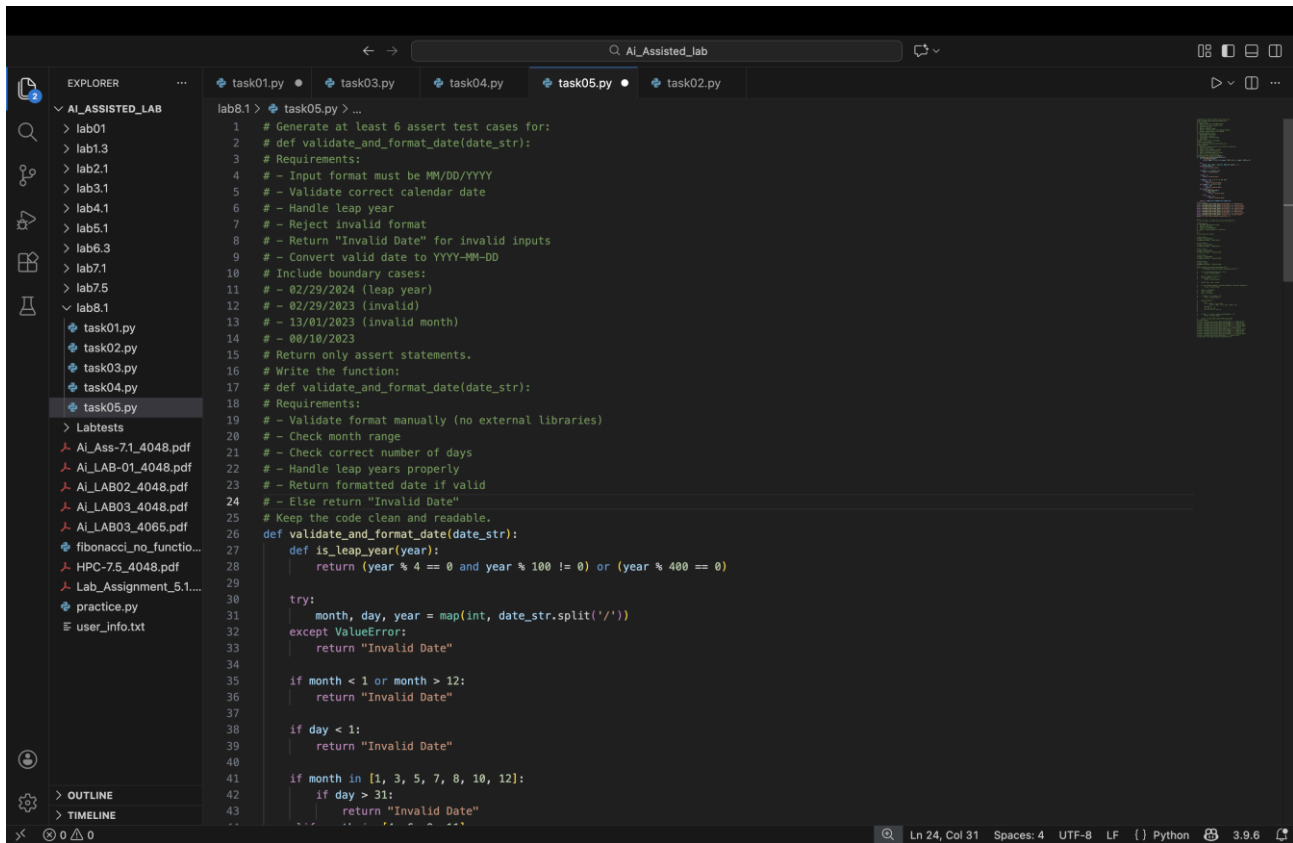
• Requirements:
      o Validate "MM/DD/YYYY" format.
      o Handle invalid dates.
      o Convert valid dates to "YYYY-MM-DD".

Example Assert Test Cases:
      assert validate_and_format_date("10/15/2023") == "2023-10-15"
      assert validate_and_format_date("02/30/2023") == "Invalid Date"
      assert validate_and_format_date("01/01/2024") == "2024-01-01"

Expected Output #5: • Function passes all AI-generated assertions and handles edge cases.

Output Screenshot:

```python
def validate_and_format_date(date_str):
        elif month in [4, 6, 9, 11]:
            if day > 30:
                return "Invalid Date"
        elif month == 2:
            if is_leap_year(year):
                if day > 29:
                    return "Invalid Date"
            else:
                if day > 28:
                    return "Invalid Date"

    return f"{year:04d}-{month:02d}-{day:02d}"
# Test Cases
assert validate_and_format_date("02/29/2024") == "2024-02-29"
assert validate_and_format_date("02/29/2023") == "Invalid Date"
assert validate_and_format_date("13/01/2023") == "Invalid Date"
assert validate_and_format_date("00/10/2023") == "Invalid Date"
assert validate_and_format_date("12/31/2023") == "2023-12-31"
assert validate_and_format_date("01/01/2023") == "2023-01-01"
assert validate_and_format_date("02/30/2024") == "Invalid Date"
assert validate_and_format_date("invalid") == "Invalid Date"
print("All test cases passed successfully ✅")
```

```
/usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task05.py
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task05.py
All test cases passed successfully ✅
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab %
```

Explanation: This ensures a date is valid in "MM/DD/YYYY" format and neatly converts it into "YYYY-MM-DD.