Lab Assignment # 08

Course Title : AI Assistant Coding Name
of Student : B.sushma
Enrollment No. : 2303A54040
Batch No. : 48

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.

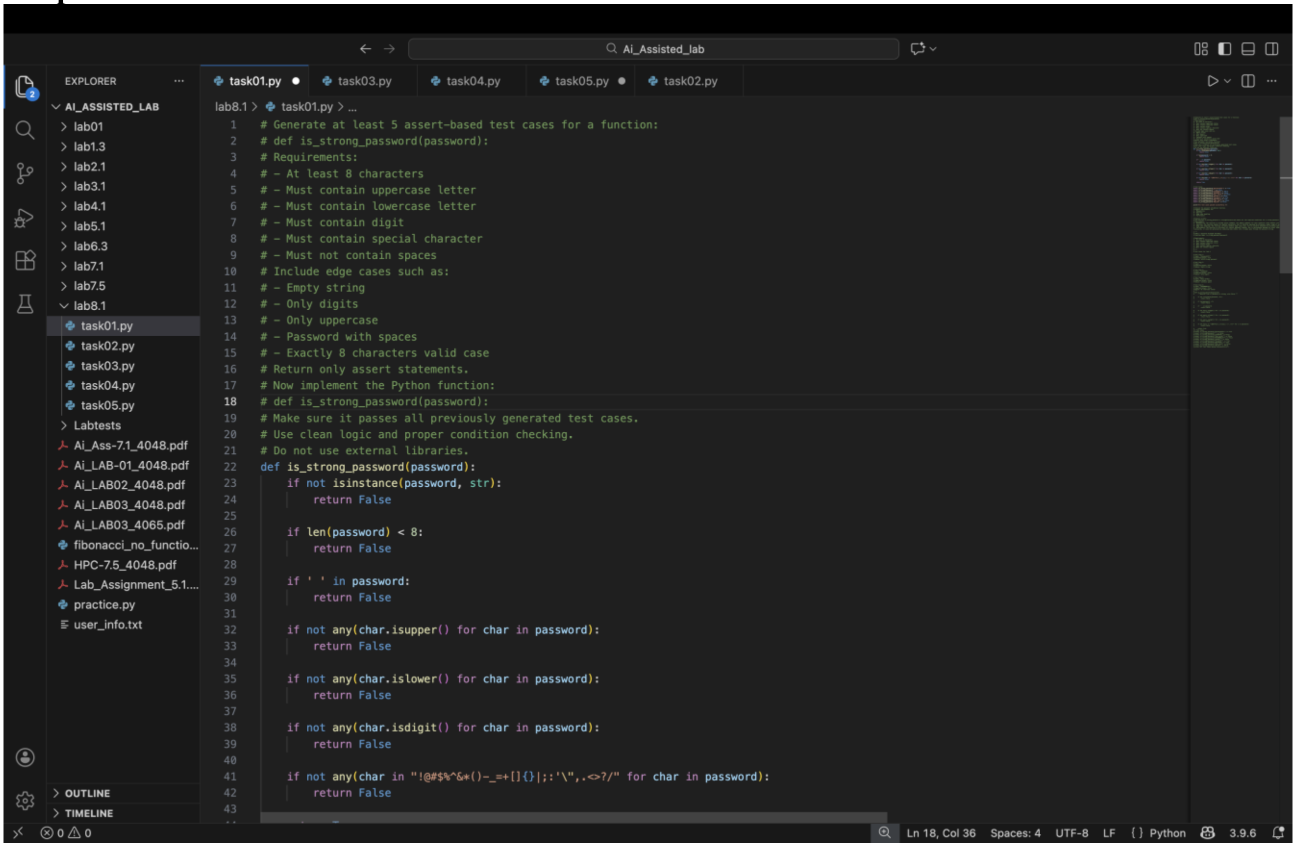
• Requirements:
o Password must have at least 8 characters.
o Must include uppercase, lowercase, digit, and special character.
o Must not contain spaces.

Example Assert Test Cases:

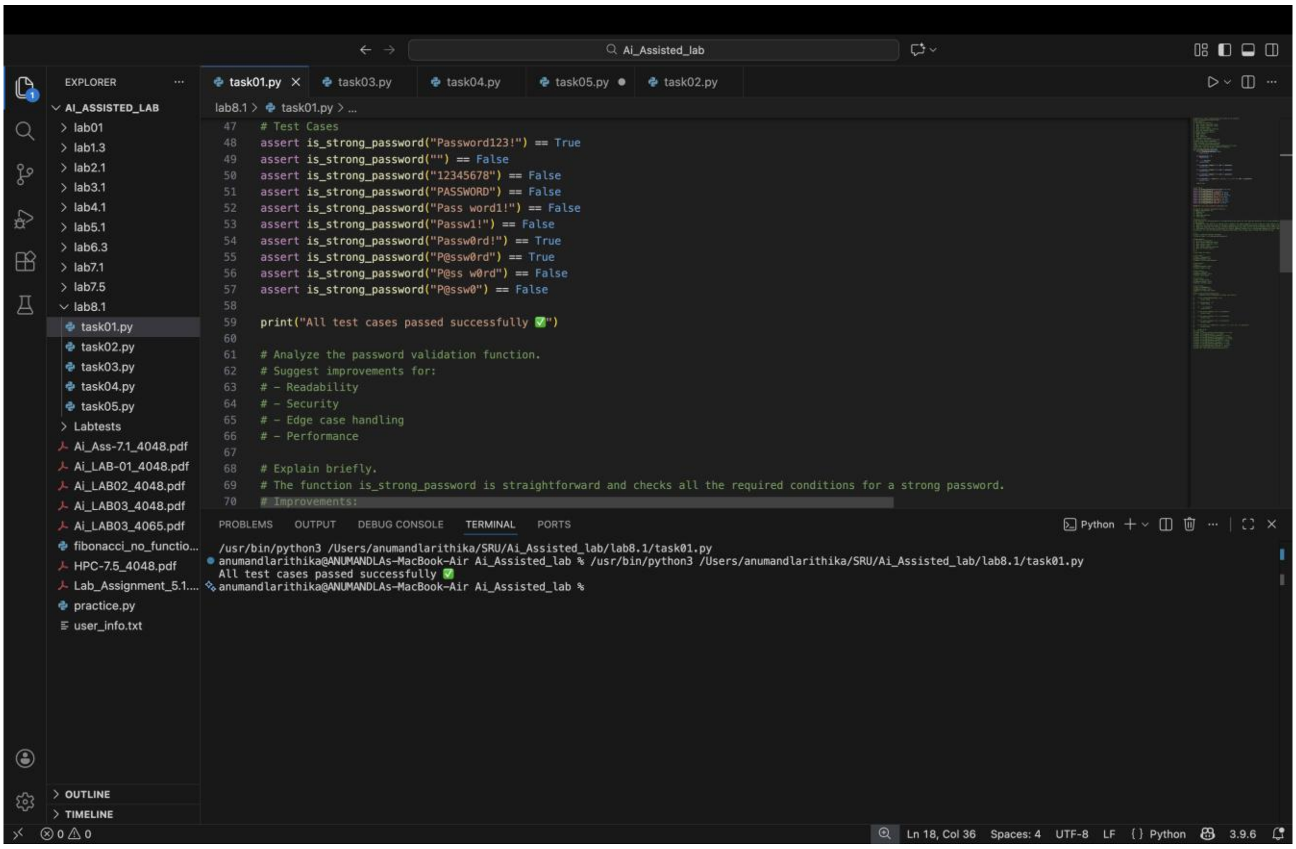
```
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True
```

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

Output Screenshot:



```python
# Generate at least 5 assert-based test cases for a function:
# def is_strong_password(password):
# Requirements:
# - At least 8 characters
# - Must contain uppercase letter
# - Must contain lowercase letter
# - Must contain digit
# - Must contain special character
# - Must not contain spaces
# Include edge cases such as:
# - Empty string
# - Only digits
# - Only uppercase
# - Password with spaces
# - Exactly 8 characters valid case
# Return only assert statements.
# Now implement the Python function:
# def is_strong_password(password):
# Make sure it passes all previously generated test cases.
# Use clean logic and proper condition checking.
# Do not use external libraries.
def is_strong_password(password):
    if not isinstance(password, str):
        return False

    if len(password) < 8:
        return False

    if ' ' in password:
        return False

    if not any(char.isupper() for char in password):
        return False

    if not any(char.islower() for char in password):
        return False

    if not any(char.isdigit() for char in password):
        return False

    if not any(char in "!@#$%^&*()-_=+[]{}|;:'\",.<>?/" for char in password):
        return False

```



```python
# Test Cases
assert is_strong_password("Password123!") == True
assert is_strong_password("") == False
assert is_strong_password("12345678") == False
assert is_strong_password("PASSWORD") == False
assert is_strong_password("Pass word1!") == False
assert is_strong_password("Passw1!") == False
assert is_strong_password("Passw0rd!") == True
assert is_strong_password("P@ssw0rd") == True
assert is_strong_password("P@ss w0rd") == False
assert is_strong_password("P@ssw0") == False

print("All test cases passed successfully ✅")

# Analyze the password validation function.
# Suggest improvements for:
# - Readability
# - Security
# - Edge case handling
# - Performance

# Explain briefly.
# The function is_strong_password is straightforward and checks all the required conditions for a strong password.
# Improvements:
```

```
/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:

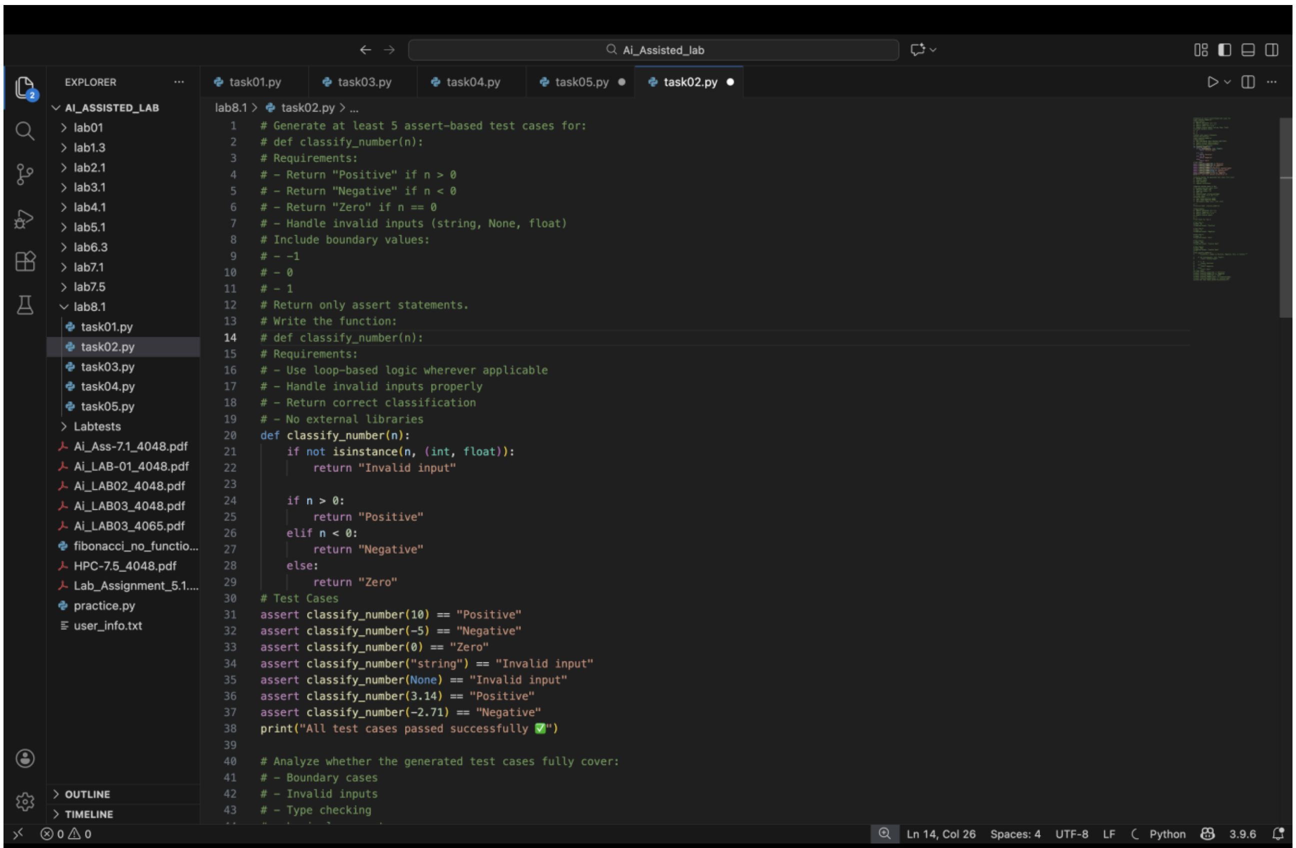
o Classify numbers as Positive, Negative, or Zero.
o Handle invalid inputs like strings and None.
o Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:

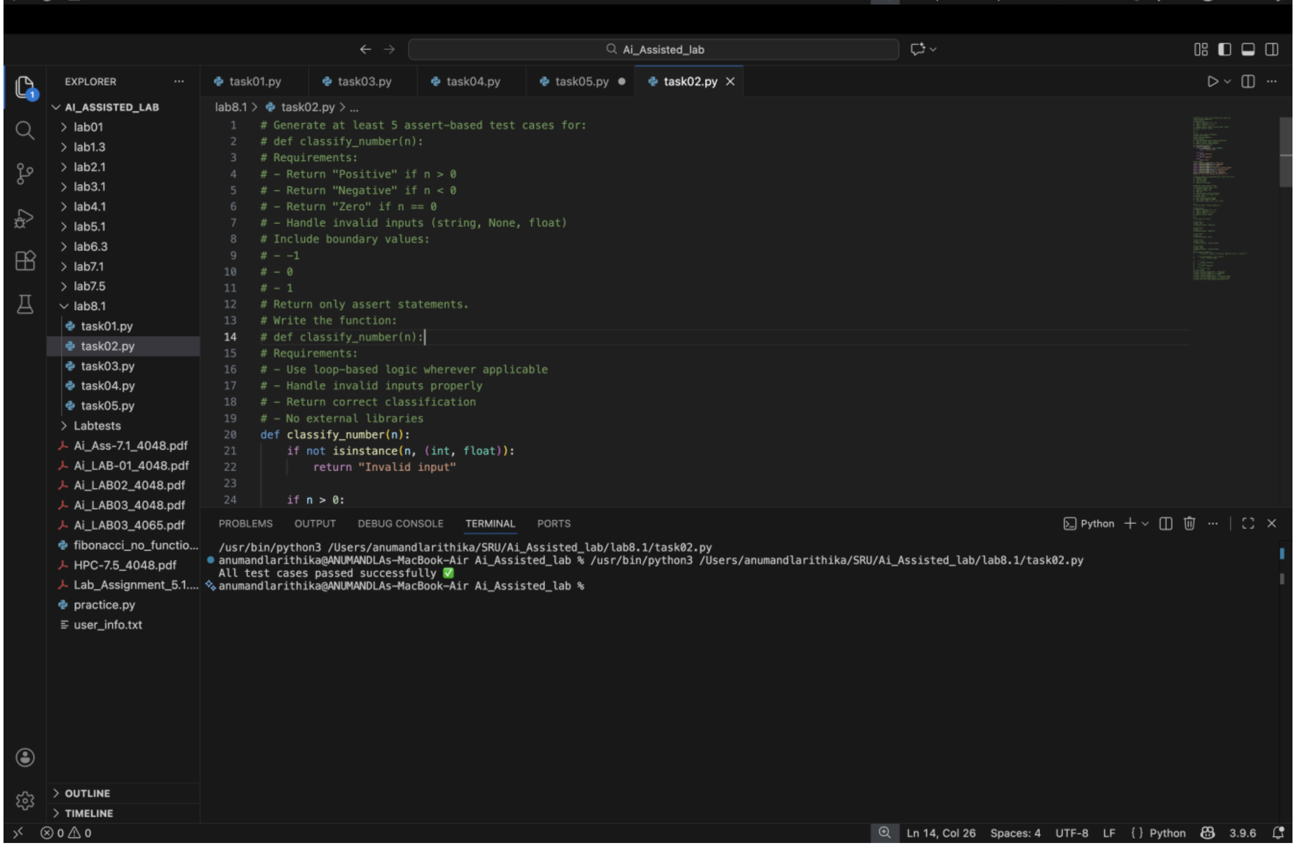
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"

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
```



```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:
```

```
/usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task02.py
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task02.py
All test cases passed successfully ✅
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab %
```

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 formis_anagram(str1, str2) and implement the function.

• Requirements:

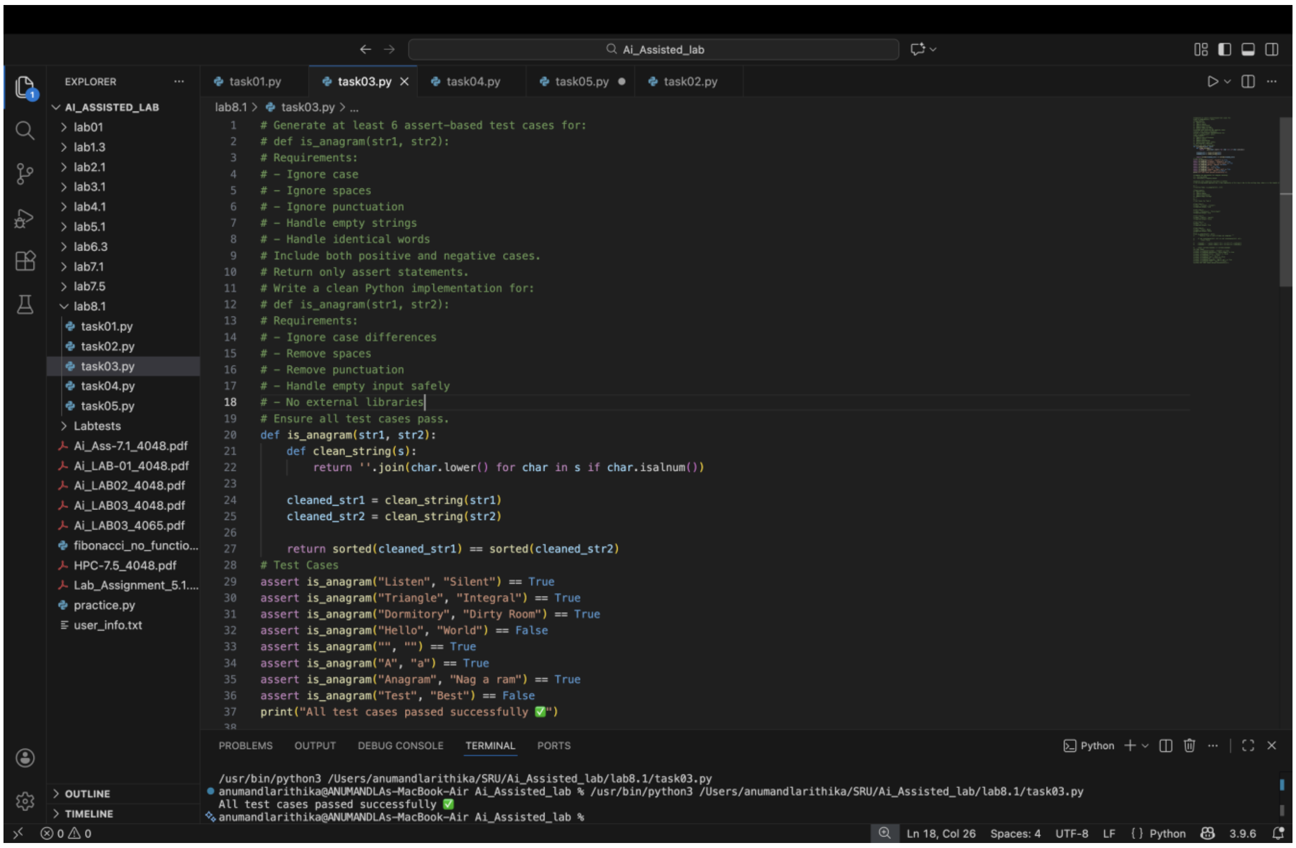
o Ignore case, spaces, and punctuation.
o Handle edge cases (empty strings, identical words).

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:



```python
# Generate at least 6 assert-based test cases for:
# def is_anagram(str1, str2):
# Requirements:
# - Ignore case
# - Ignore spaces
# - Ignore punctuation
# - Handle empty strings
# - Handle identical words
# Include both positive and negative cases.
# Return only assert statements.
# Write a clean Python implementation for:
# def is_anagram(str1, str2):
# Requirements:
# - Ignore case differences
# - Remove spaces
# - Remove punctuation
# - Handle empty input safely
# - No external libraries
# Ensure all test cases pass.
def is_anagram(str1, str2):
    def clean_string(s):
        return ''.join(char.lower() for char in s if char.isalnum())

    cleaned_str1 = clean_string(str1)
    cleaned_str2 = clean_string(str2)

    return sorted(cleaned_str1) == sorted(cleaned_str2)
# Test Cases
assert is_anagram("Listen", "Silent") == True
assert is_anagram("Triangle", "Integral") == True
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("Hello", "World") == False
assert is_anagram("", "") == True
assert is_anagram("A", "a") == True
assert is_anagram("Anagram", "Nag a ram") == True
assert is_anagram("Test", "Best") == False
print("All test cases passed successfully ✅")
```

```
/usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task03.py
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task03.py
All test cases passed successfully ✅
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab %
```

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:

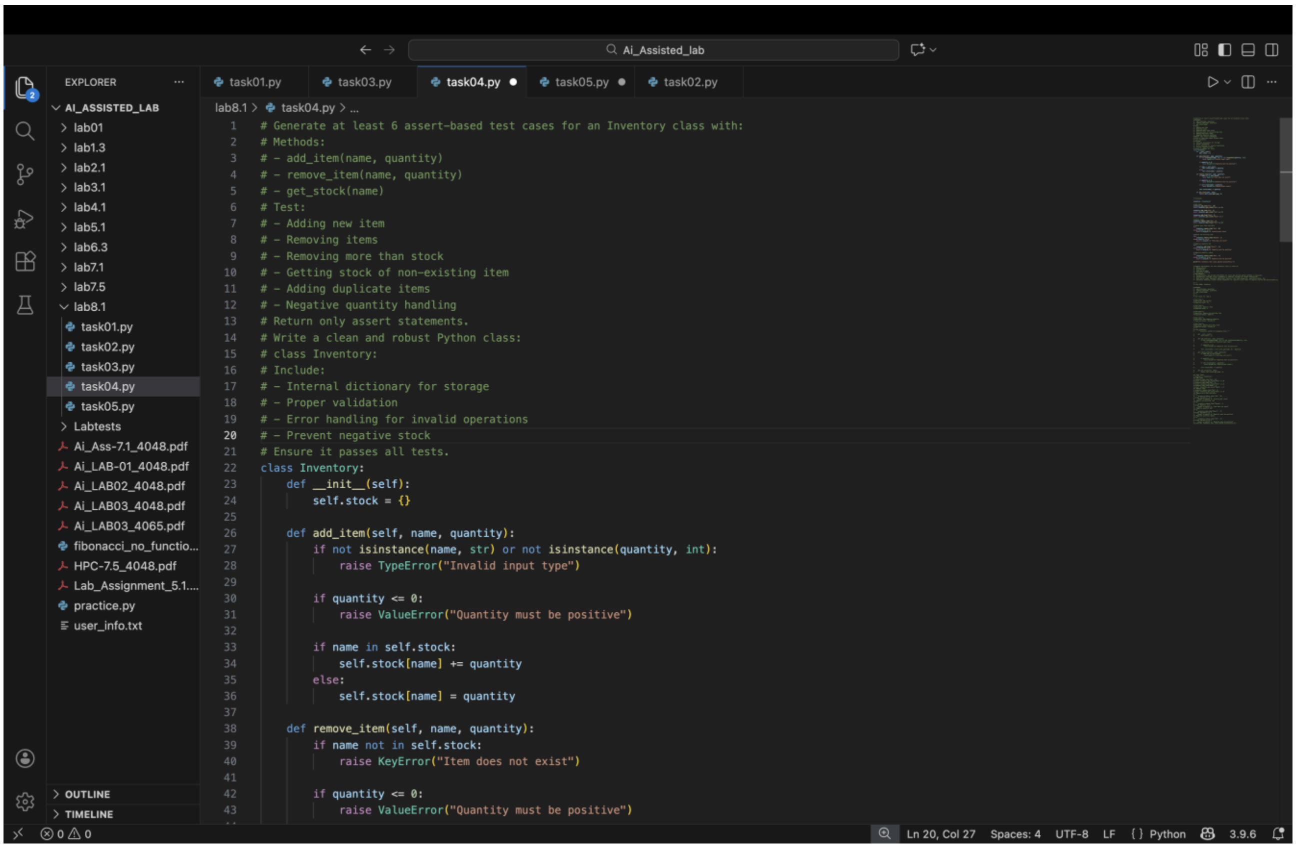
inv = Inventory()

inv.add_item("Pen", 10)
assert inv.get_stock("Pen") == 10

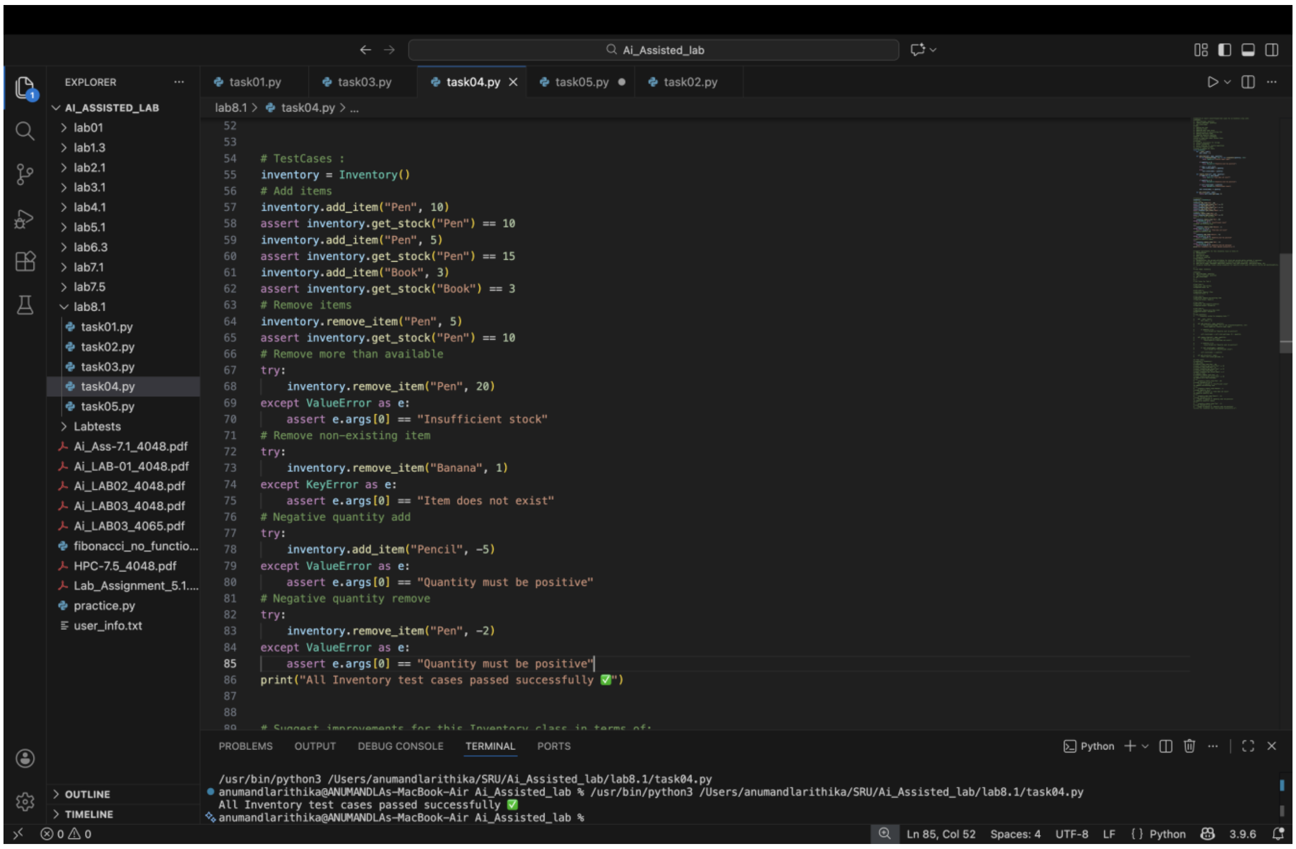
```
inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5
inv.add_item("Book", 3)
assert inv.get_stock("Book") == 3
```

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

Output Screenshot:



```python
# Generate at least 6 assert-based test cases for an Inventory class with:
# Methods:
# - add_item(name, quantity)
# - remove_item(name, quantity)
# - get_stock(name)
# Test:
# - Adding new item
# - Removing items
# - Removing more than stock
# - Getting stock of non-existing item
# - Adding duplicate items
# - Negative quantity handling
# Return only assert statements.
# Write a clean and robust Python class:
# class Inventory:
# Include:
# - Internal dictionary for storage
# - Proper validation
# - Error handling for invalid operations
# - Prevent negative stock
# Ensure it passes all tests.
class Inventory:
    def __init__(self):
        self.stock = {}

    def add_item(self, name, quantity):
        if not isinstance(name, str) or not isinstance(quantity, int):
            raise TypeError("Invalid input type")

        if quantity <= 0:
            raise ValueError("Quantity must be positive")

        if name in self.stock:
            self.stock[name] += quantity
        else:
            self.stock[name] = quantity

    def remove_item(self, name, quantity):
        if name not in self.stock:
            raise KeyError("Item does not exist")

        if quantity <= 0:
            raise ValueError("Quantity must be positive")
```



```python
# TestCases :
inventory = Inventory()
# Add items
inventory.add_item("Pen", 10)
assert inventory.get_stock("Pen") == 10
inventory.add_item("Pen", 5)
assert inventory.get_stock("Pen") == 15
inventory.add_item("Book", 3)
assert inventory.get_stock("Book") == 3
# Remove items
inventory.remove_item("Pen", 5)
assert inventory.get_stock("Pen") == 10
# Remove more than available
try:
    inventory.remove_item("Pen", 20)
except ValueError as e:
    assert e.args[0] == "Insufficient stock"
# Remove non-existing item
try:
    inventory.remove_item("Banana", 1)
except KeyError as e:
    assert e.args[0] == "Item does not exist"
# Negative quantity add
try:
    inventory.add_item("Pencil", -5)
except ValueError as e:
    assert e.args[0] == "Quantity must be positive"
# Negative quantity remove
try:
    inventory.remove_item("Pen", -2)
except ValueError as e:
    assert e.args[0] == "Quantity must be positive"
print("All Inventory test cases passed successfully ✅")


# Suggest improvements for this Inventory class in terms of:
```

```
/usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task04.py
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab8.1/task04.py
All Inventory test cases passed successfully ✅
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab %
```

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:

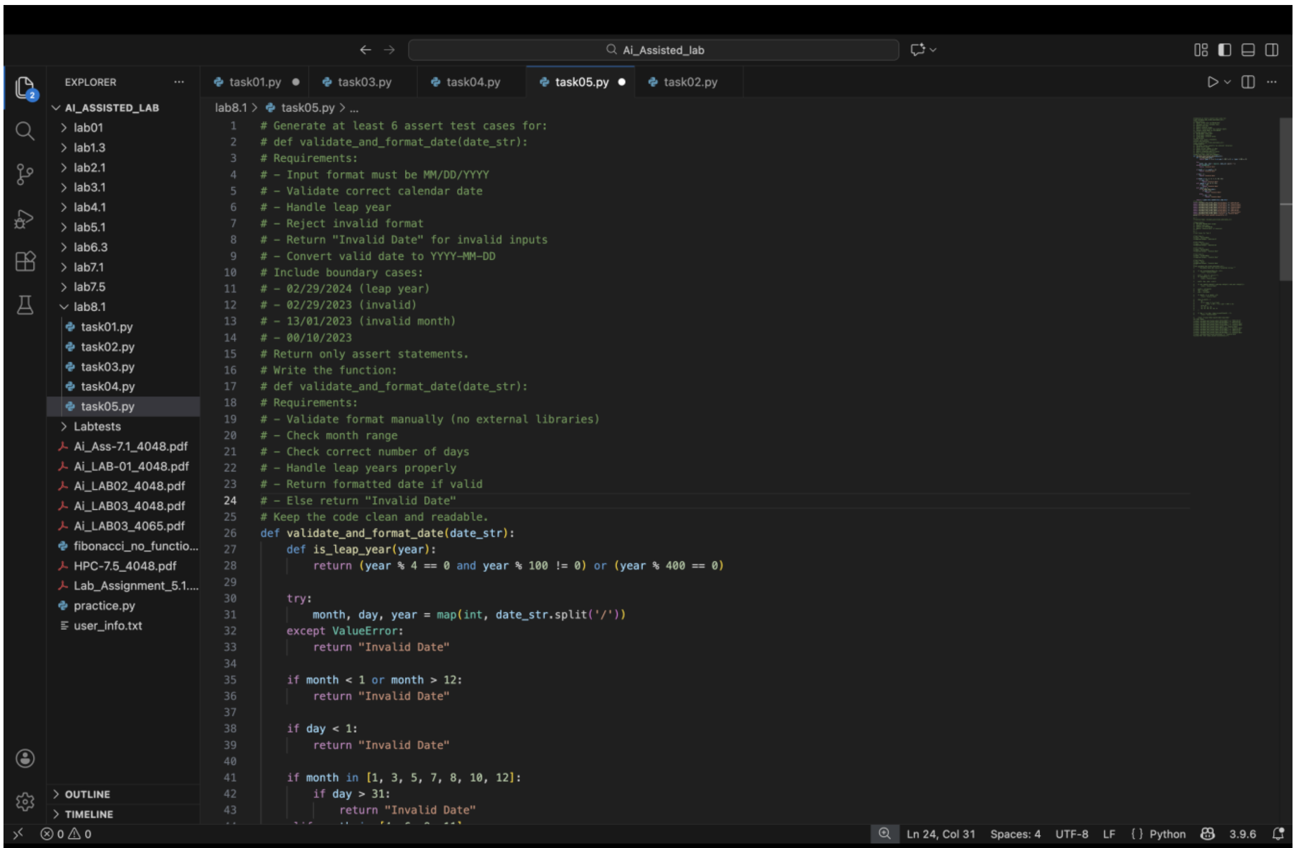
o Validate "MM/DD/YYYY" format.
o Handle invalid dates.
o Convert valid dates to "YYYY-MM-DD".

Example Assert Test Cases:

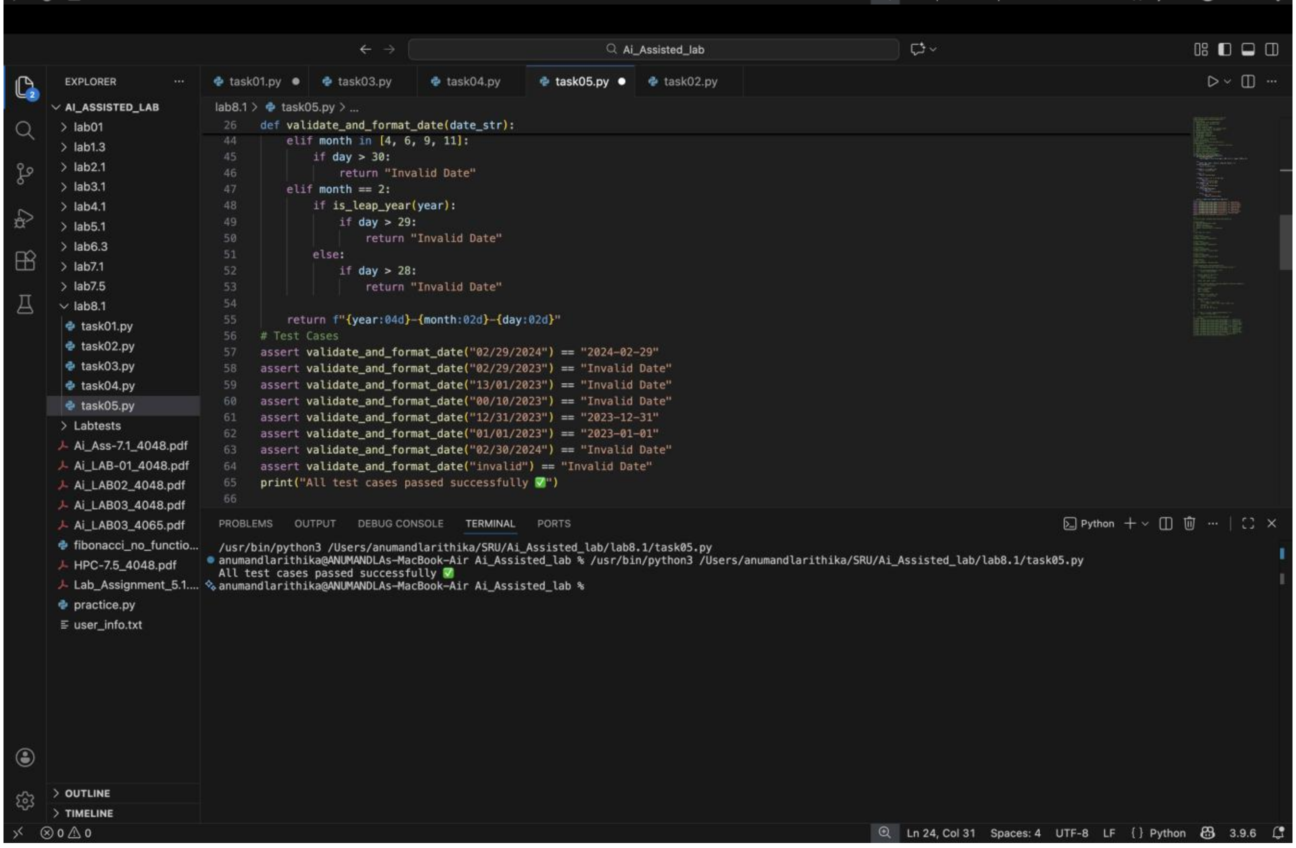
assert validate_and_format_date("10/15/2023") == "2023-10-15"
assert validate_and_format_date("02/30/2023") == "Invalid Date"
assert validate_and_format_date("01/01/2024") == "2024-01-01"

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

Output Screenshot:



```python
# Generate at least 6 assert test cases for:
# def validate_and_format_date(date_str):
# Requirements:
# - Input format must be MM/DD/YYYY
# - Validate correct calendar date
# - Handle leap year
# - Reject invalid format
# - Return "Invalid Date" for invalid inputs
# - Convert valid date to YYYY-MM-DD
# Include boundary cases:
# - 02/29/2024 (leap year)
# - 02/29/2023 (invalid)
# - 13/01/2023 (invalid month)
# - 00/10/2023
# Return only assert statements.
# Write the function:
# def validate_and_format_date(date_str):
# Requirements:
# - Validate format manually (no external libraries)
# - Check month range
# - Check correct number of days
# - Handle leap years properly
# - Return formatted date if valid
# - Else return "Invalid Date"
# Keep the code clean and readable.
def validate_and_format_date(date_str):
    def is_leap_year(year):
        return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

    try:
        month, day, year = map(int, date_str.split('/'))
    except ValueError:
        return "Invalid Date"

    if month < 1 or month > 12:
        return "Invalid Date"

    if day < 1:
        return "Invalid Date"

    if month in [1, 3, 5, 7, 8, 10, 12]:
        if day > 31:
            return "Invalid Date"
```



```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.