School of Computer Science and Artificial Intelligence

Lab Assignment-8.1

Course Name : AI Assistant Coding
Name of the Student : Shaik Naved Ahmed
Registration No : 2303A54053
Batch No : 47-B

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:

Password must have at least 8 characters.

Must include uppercase, lowercase, digit, and special character.

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

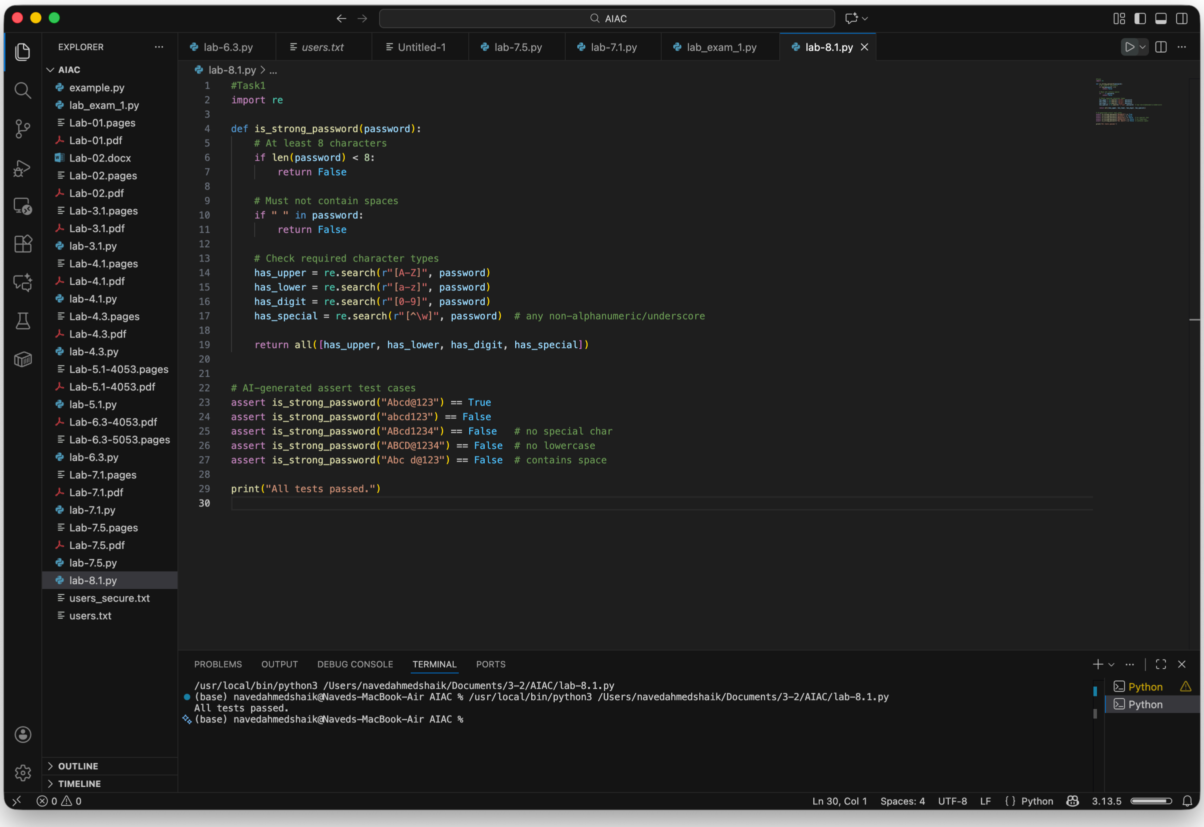
Expected Output #1:

Password validation logic passing all AI-generated test cases.

EXPLANATION:

The function is_strong_password checks if the given password meets the specified criteria for a strong password. It first checks if the length of the password is at least 8 characters and if it does not contain any spaces. Then, it uses regular expressions to check for the presence of at least one uppercase letter, one lowercase letter, one digit, and one special character. The function returns True if all conditions are met, otherwise it returns False. The assert statements are used to test the function with various passwords to ensure it behaves as expected.

SCREENSHOT OF GENERATED CODE:



```python
#Task1
import re

def is_strong_password(password):
    # At least 8 characters
    if len(password) < 8:
        return False

    # Must not contain spaces
    if " " in password:
        return False

    # Check required character types
    has_upper = re.search(r"[A-Z]", password)
    has_lower = re.search(r"[a-z]", password)
    has_digit = re.search(r"[0-9]", password)
    has_special = re.search(r"[^\w]", password)  # any non-alphanumeric/underscore

    return all([has_upper, has_lower, has_digit, has_special])


# AI-generated assert test cases
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABcd1234") == False   # no special char
assert is_strong_password("ABCD@1234") == False  # no lowercase
assert is_strong_password("Abc d@123") == False  # contains space

print("All tests passed.")
```

```
/usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC %
```

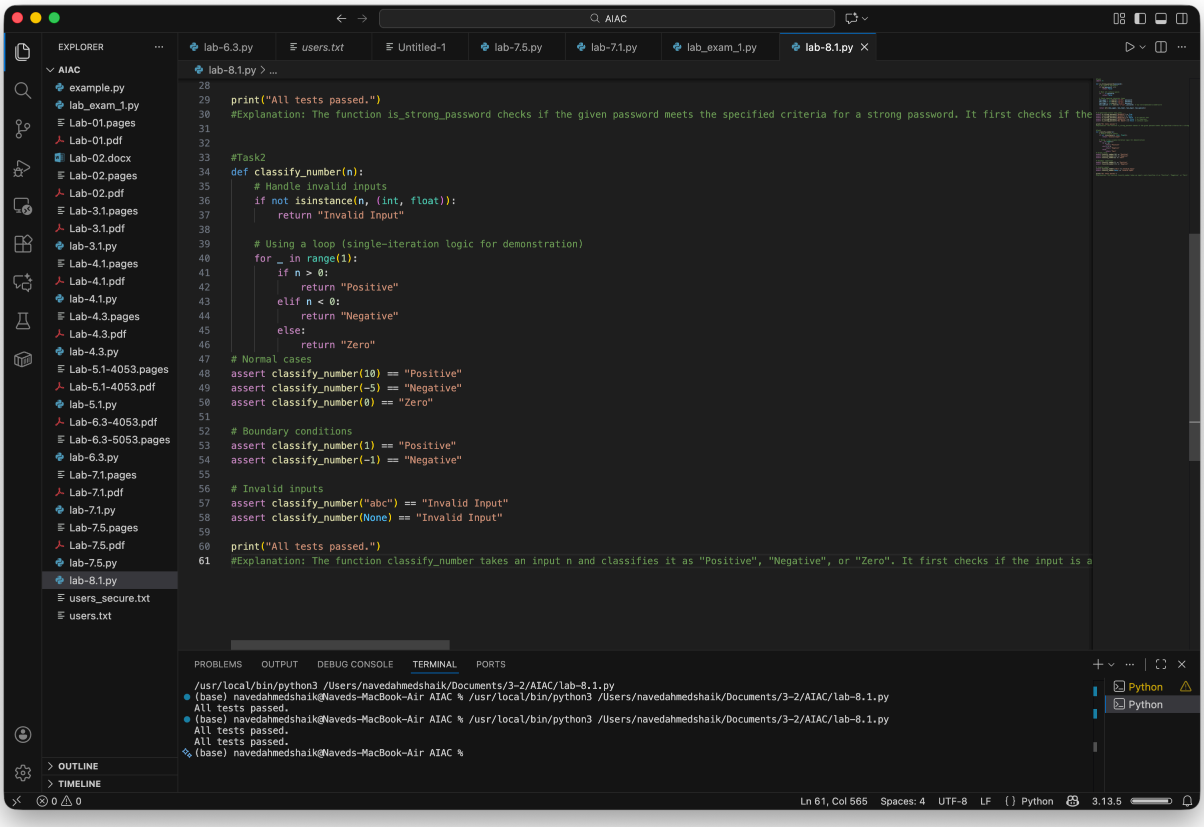
Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)

Task: Use AI to generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.
Requirements:
Classify numbers as Positive, Negative, or Zero.
Handle invalid inputs like strings and None.
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.

EXPLANATION:

The function classify_number takes an input n and classifies it as "Positive", "Negative", or "Zero". It first checks if the input is a valid number (integer or float). If the input is invalid, it returns "Invalid Input". Then, it uses a loop (with a single iteration) to check if n is greater than 0 (positive), less than 0 (negative), or equal to 0 (zero) and returns the appropriate classification. The assert statements test the function with normal cases, boundary conditions, and invalid inputs to ensure it behaves correctly in all scenarios.

SCREENSHOT OF GENERATED CODE:



```python

print("All tests passed.")
#Explanation: The function is_strong_password checks if the given password meets the specified criteria for a strong password. It first checks if the


#Task2
def classify_number(n):
    # Handle invalid inputs
    if not isinstance(n, (int, float)):
        return "Invalid Input"

    # Using a loop (single-iteration logic for demonstration)
    for _ in range(1):
        if n > 0:
            return "Positive"
        elif n < 0:
            return "Negative"
        else:
            return "Zero"
# Normal cases
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"

# Boundary conditions
assert classify_number(1) == "Positive"
assert classify_number(-1) == "Negative"

# Invalid inputs
assert classify_number("abc") == "Invalid Input"
assert classify_number(None) == "Invalid Input"

print("All tests passed.")
#Explanation: The function classify_number takes an input n and classifies it as "Positive", "Negative", or "Zero". It first checks if the input is a
```

```
/usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC %
```

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:

Ignore case, spaces, and punctuation.

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

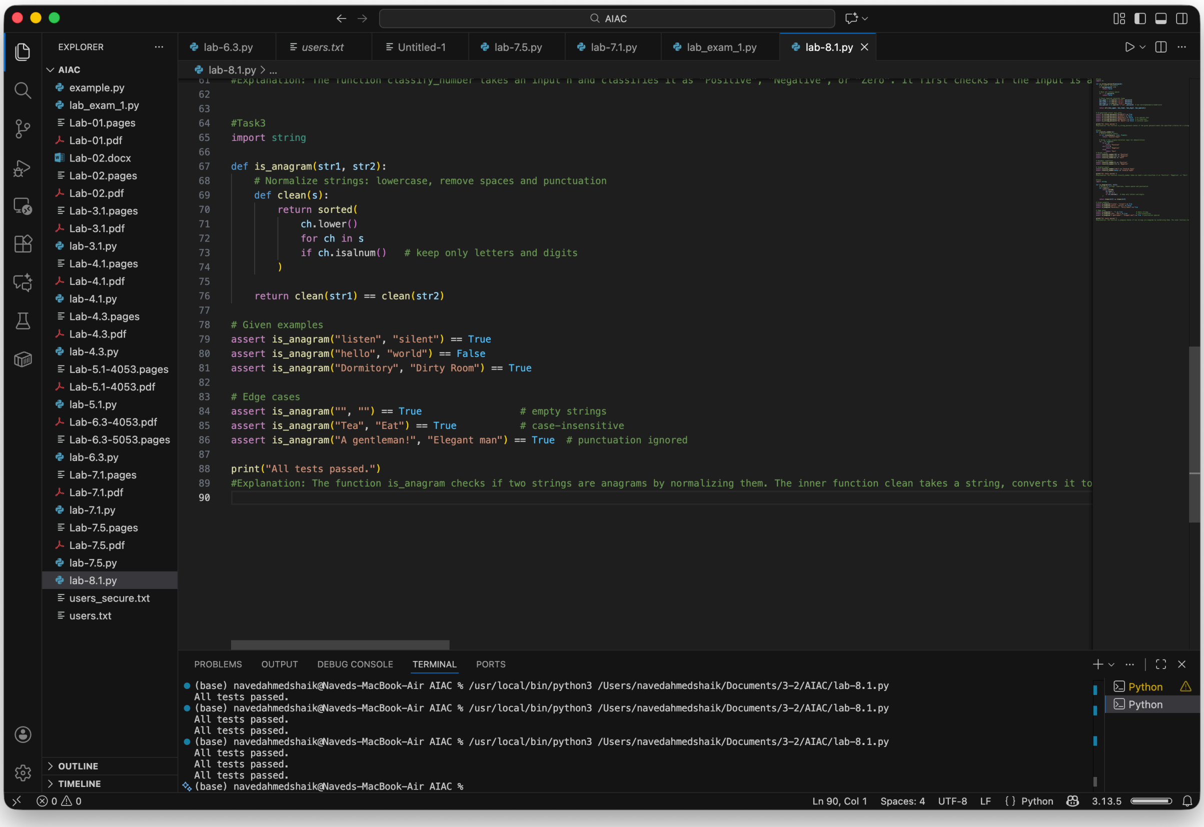
Expected Output #3:

Function correctly identifying anagrams and passing all AI-generated tests

EXPLANATION:

The function is_anagram checks if two strings are anagrams by normalizing them. The inner function clean takes a string, converts it to lowercase, and removes any spaces and punctuation, keeping only alphanumeric characters. It then sorts the characters of the cleaned string. The main function compares the sorted character lists of both strings to determine if they are anagrams. The assert statements test the function with the provided examples and additional edge cases to ensure it works correctly in various scenarios.

SCREENSHOT OF GENERATED CODE:



```
#Explanation: The function classify_number takes an input n and classifies it as "Positive", "Negative", or "Zero". It first checks if the input is a


#Task3
import string

def is_anagram(str1, str2):
    # Normalize strings: lowercase, remove spaces and punctuation
    def clean(s):
        return sorted(
            ch.lower()
            for ch in s
            if ch.isalnum()   # keep only letters and digits
        )

    return clean(str1) == clean(str2)

# Given examples
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True

# Edge cases
assert is_anagram("", "") == True                 # empty strings
assert is_anagram("Tea", "Eat") == True           # case-insensitive
assert is_anagram("A gentleman!", "Elegant man") == True  # punctuation ignored

print("All tests passed.")
#Explanation: The function is_anagram checks if two strings are anagrams by normalizing them. The inner function clean takes a string, converts it to
```

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
All tests passed.
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC %
```

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:

add_item(name, quantity)

remove_item(name, quantity)

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

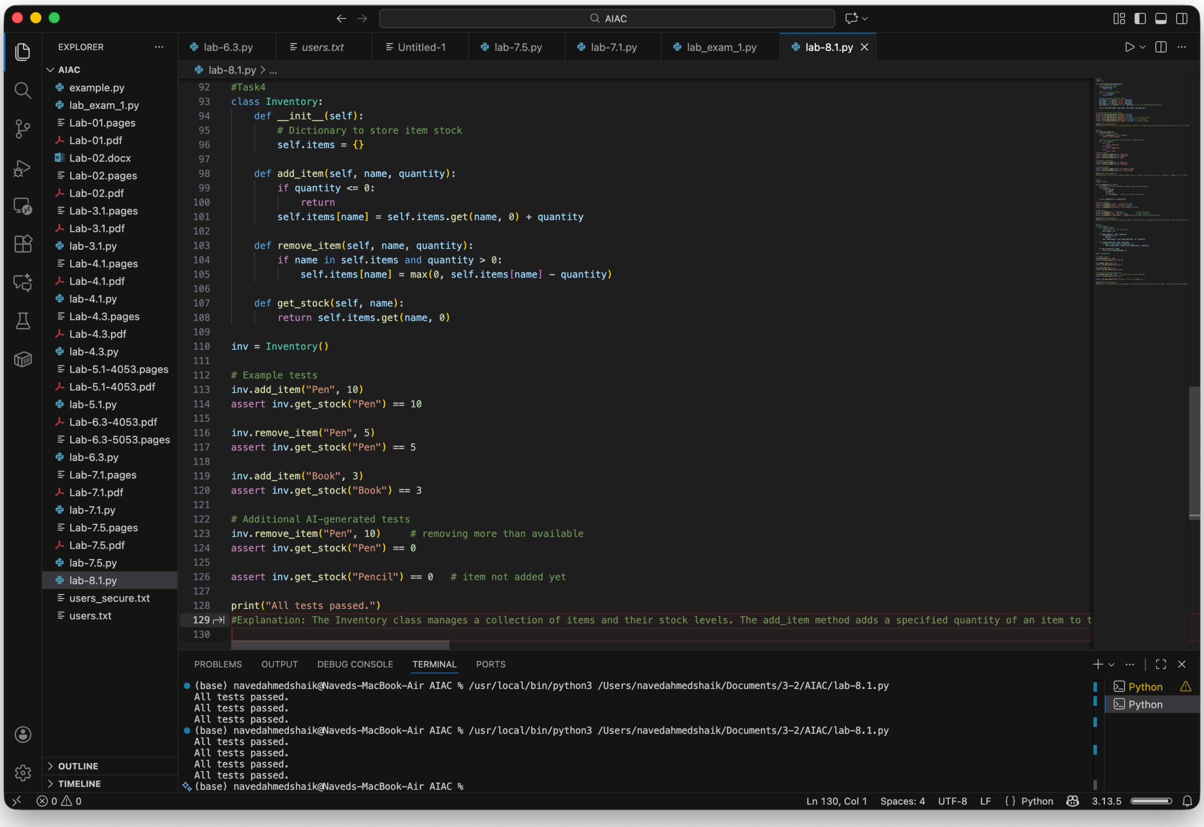
Expected Output #4:

Fully functional class passing all assertions.

EXPLANATION:

The Inventory class manages a collection of items and their stock levels. The add_item method adds a specified quantity of an item to the inventory, while the remove_item method reduces the stock of an item, ensuring it does not go below zero. The get_stock method returns the current stock level of a specified item. The assert statements test the functionality of the Inventory class with various scenarios, including adding and removing items, as well as checking stock levels for existing and non-existing items.

SCREENSHOT OF GENERATED CODE:



```python
#Task4
class Inventory:
    def __init__(self):
        # Dictionary to store item stock
        self.items = {}

    def add_item(self, name, quantity):
        if quantity <= 0:
            return
        self.items[name] = self.items.get(name, 0) + quantity

    def remove_item(self, name, quantity):
        if name in self.items and quantity > 0:
            self.items[name] = max(0, self.items[name] - quantity)

    def get_stock(self, name):
        return self.items.get(name, 0)

inv = Inventory()

# Example tests
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

# Additional AI-generated tests
inv.remove_item("Pen", 10)     # removing more than available
assert inv.get_stock("Pen") == 0

assert inv.get_stock("Pencil") == 0   # item not added yet

print("All tests passed.")
#Explanation: The Inventory class manages a collection of items and their stock levels. The add_item method adds a specified quantity of an item to t
```

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
All tests passed.
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
All tests passed.
All tests passed.
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC %
```

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:

Validate "MM/DD/YYYY" format.

Handle invalid dates.

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"

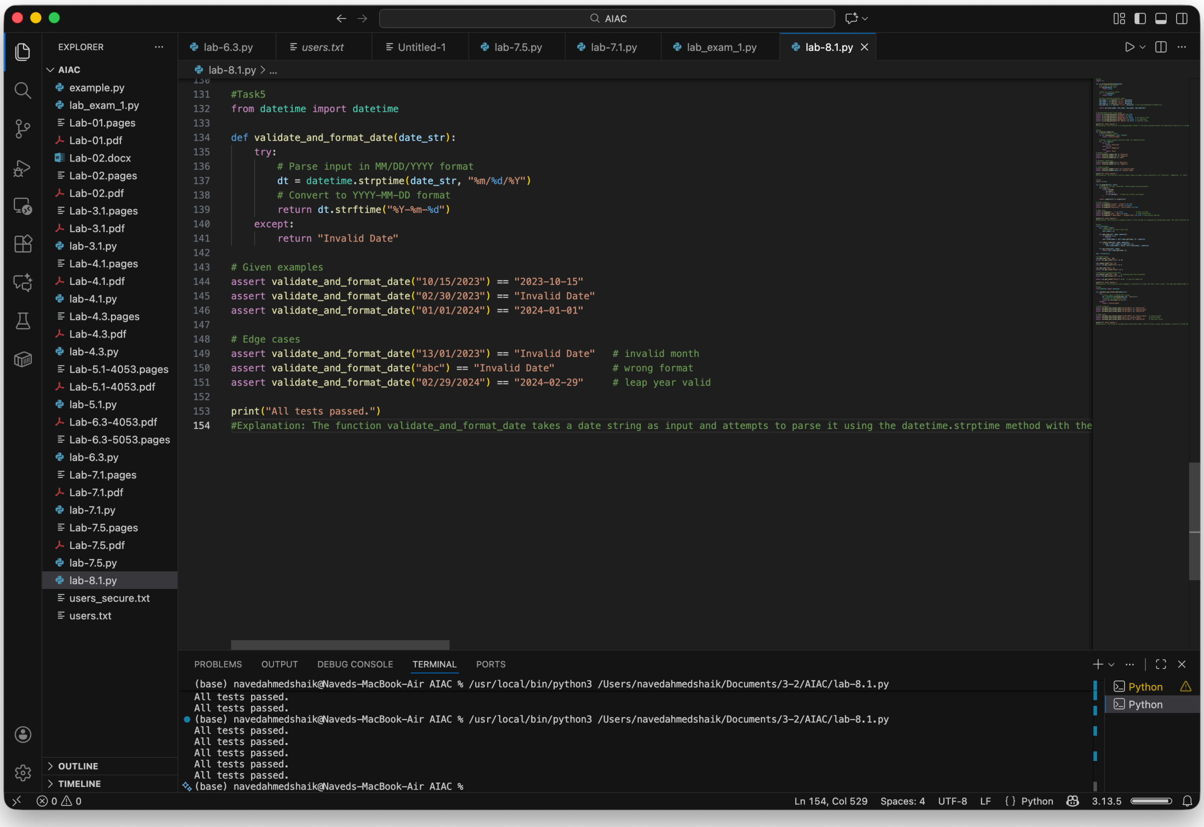
Expected Output #5:

Function passes all AI-generated assertions and handles edge cases.

EXPLANATION:

The function validate_and_format_date takes a date string as input and attempts to parse it using the datetime.strptime method with the expected format of MM/DD/YYYY. If the parsing is successful, it converts the date to the desired format of YYYY-MM-DD using strftime. If the input date is invalid or does not match the expected format, the function returns "Invalid Date". The assert statements test the function with valid dates, invalid dates, and edge cases to ensure it behaves correctly in all scenarios.

SCREENSHOT OF GENERATED CODE:



```python
#Task5
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        # Parse input in MM/DD/YYYY format
        dt = datetime.strptime(date_str, "%m/%d/%Y")
        # Convert to YYYY-MM-DD format
        return dt.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"

# Given examples
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"

# Edge cases
assert validate_and_format_date("13/01/2023") == "Invalid Date"   # invalid month
assert validate_and_format_date("abc") == "Invalid Date"          # wrong format
assert validate_and_format_date("02/29/2024") == "2024-02-29"     # leap year valid

print("All tests passed.")
#Explanation: The function validate_and_format_date takes a date string as input and attempts to parse it using the datetime.strptime method with the
```

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
All tests passed.
All tests passed.
All tests passed.
All tests passed.
All tests passed.
(base) navedahmedshaik@Naveds-MacBook-Air AIAC %
```