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

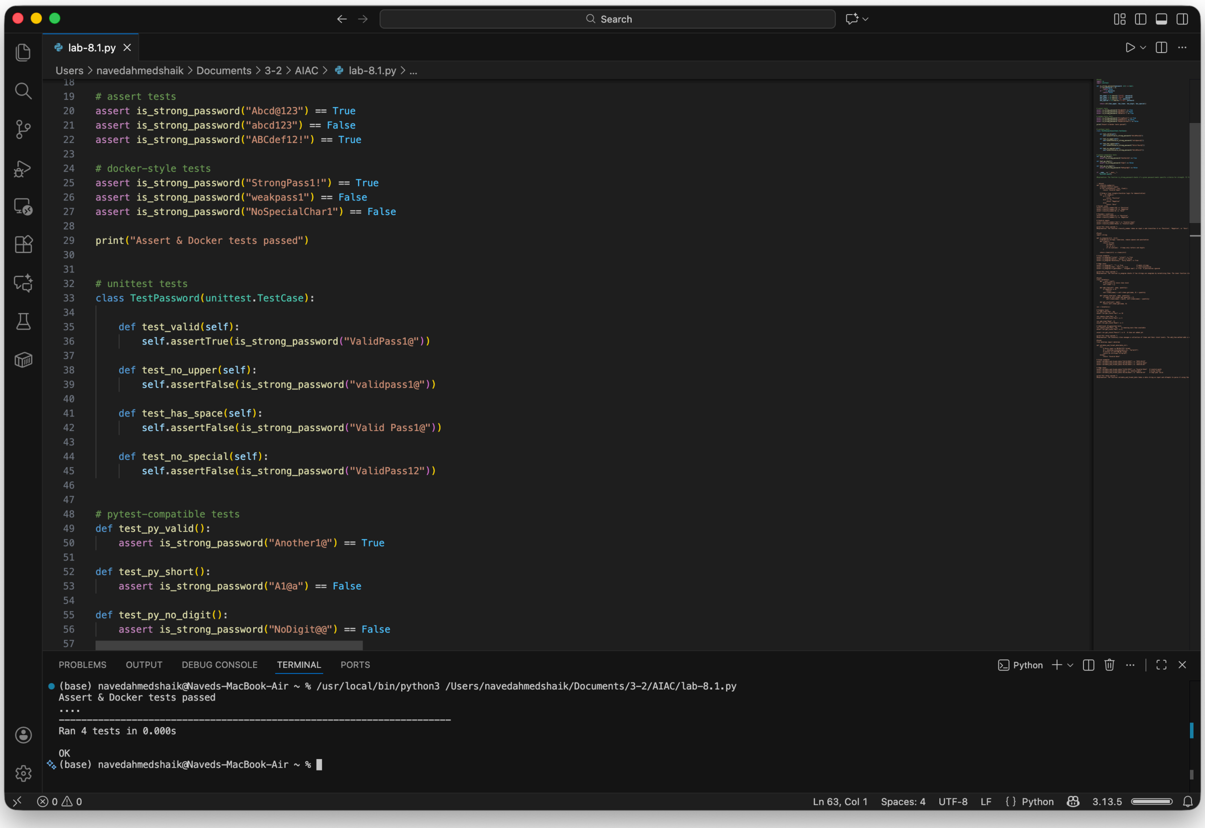
Expected Output #1:

Password validation logic passing all AI-generated test cases.

EXPLANATION:

The function is_strong_password checks if a given password meets specific strength criteria. It first checks if the password is at least 8 characters long and 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 assert statements test the function with various passwords to ensure it correctly identifies strong and weak passwords based on the defined criteria.

SCREENSHOT OF GENERATED CODE:



```python
# assert tests
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCdef12!") == True

# docker-style tests
assert is_strong_password("StrongPass1!") == True
assert is_strong_password("weakpass1") == False
assert is_strong_password("NoSpecialChar1") == False

print("Assert & Docker tests passed")


# unittest tests
class TestPassword(unittest.TestCase):

    def test_valid(self):
        self.assertTrue(is_strong_password("ValidPass1@"))

    def test_no_upper(self):
        self.assertFalse(is_strong_password("validpass1@"))

    def test_has_space(self):
        self.assertFalse(is_strong_password("Valid Pass1@"))

    def test_no_special(self):
        self.assertFalse(is_strong_password("ValidPass12"))


# pytest-compatible tests
def test_py_valid():
    assert is_strong_password("Another1@") == True

def test_py_short():
    assert is_strong_password("A1@a") == False

def test_py_no_digit():
    assert is_strong_password("NoDigit@@") == False
```

```
(base) navedahmedshaik@Naveds-MacBook-Air ~ % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
Assert & Docker tests passed
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
(base) navedahmedshaik@Naveds-MacBook-Air ~ %
```

lab-8.1.py
Users > navedahmedshaik > Documents > 3-2 > AIAC > lab-8.1.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Ln 63, Col 1 Spaces: 4 UTF-8 LF { } Python 3.13.5

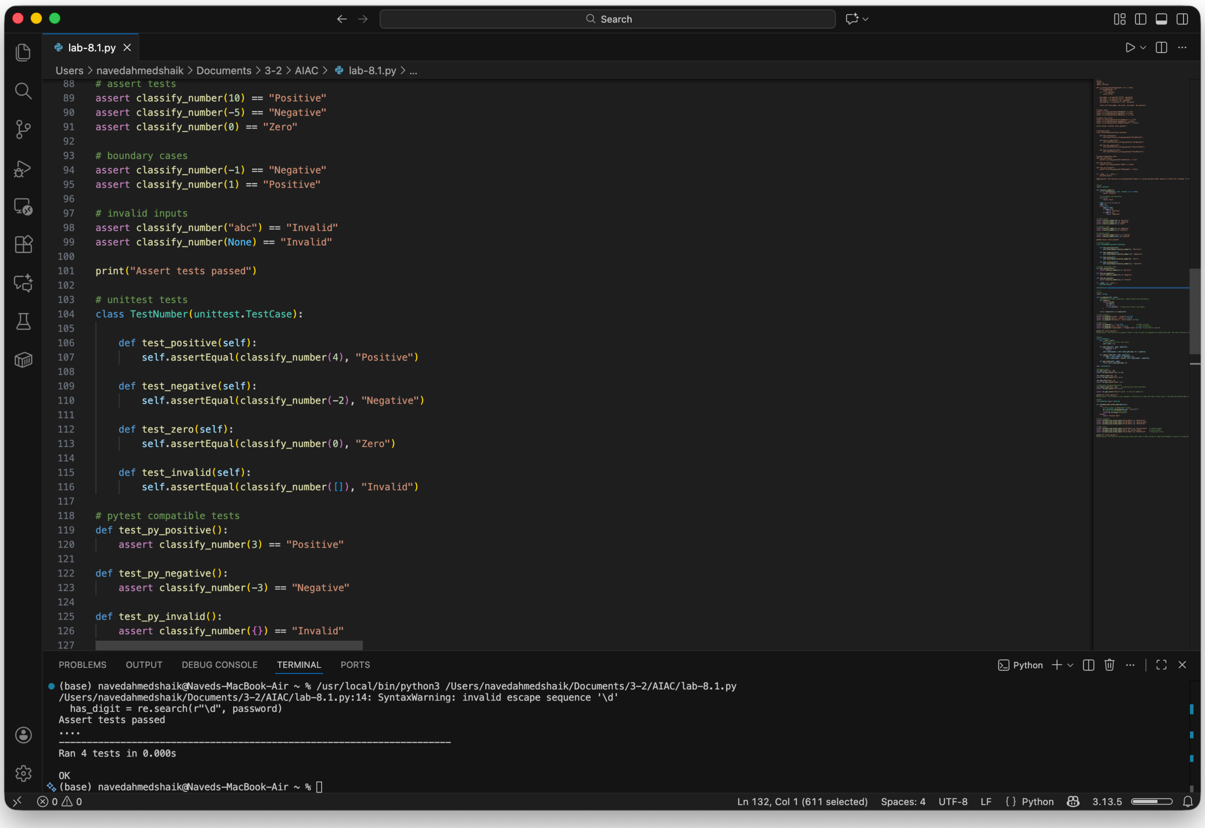
Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)

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 determines if a given input is a positive number, negative number, zero, or invalid. It first checks if the input is an instance of int or float and not None. If the input is valid, it uses a loop to determine the sign of the number by incrementing or decrementing a temporary variable until it reaches 1 or -1. The assert statements test the function with various inputs, including valid numbers and invalid types. The unittest class provides structured tests for different scenarios, while the pytest-compatible functions allow for additional testing in a more flexible manner.

SCREENSHOT OF GENERATED CODE:



```python
# assert tests
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"

# boundary cases
assert classify_number(-1) == "Negative"
assert classify_number(1) == "Positive"

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

print("Assert tests passed")

# unittest tests
class TestNumber(unittest.TestCase):

    def test_positive(self):
        self.assertEqual(classify_number(4), "Positive")

    def test_negative(self):
        self.assertEqual(classify_number(-2), "Negative")

    def test_zero(self):
        self.assertEqual(classify_number(0), "Zero")

    def test_invalid(self):
        self.assertEqual(classify_number([]), "Invalid")

# pytest compatible tests
def test_py_positive():
    assert classify_number(3) == "Positive"

def test_py_negative():
    assert classify_number(-3) == "Negative"

def test_py_invalid():
    assert classify_number({}) == "Invalid"
```

```
(base) navedahmedshaik@Naveds-MacBook-Air ~ % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
/Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py:14: SyntaxWarning: invalid escape sequence '\d'
  has_digit = re.search(r"\d", password)
Assert tests passed
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
(base) navedahmedshaik@Naveds-MacBook-Air ~ % 
```

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

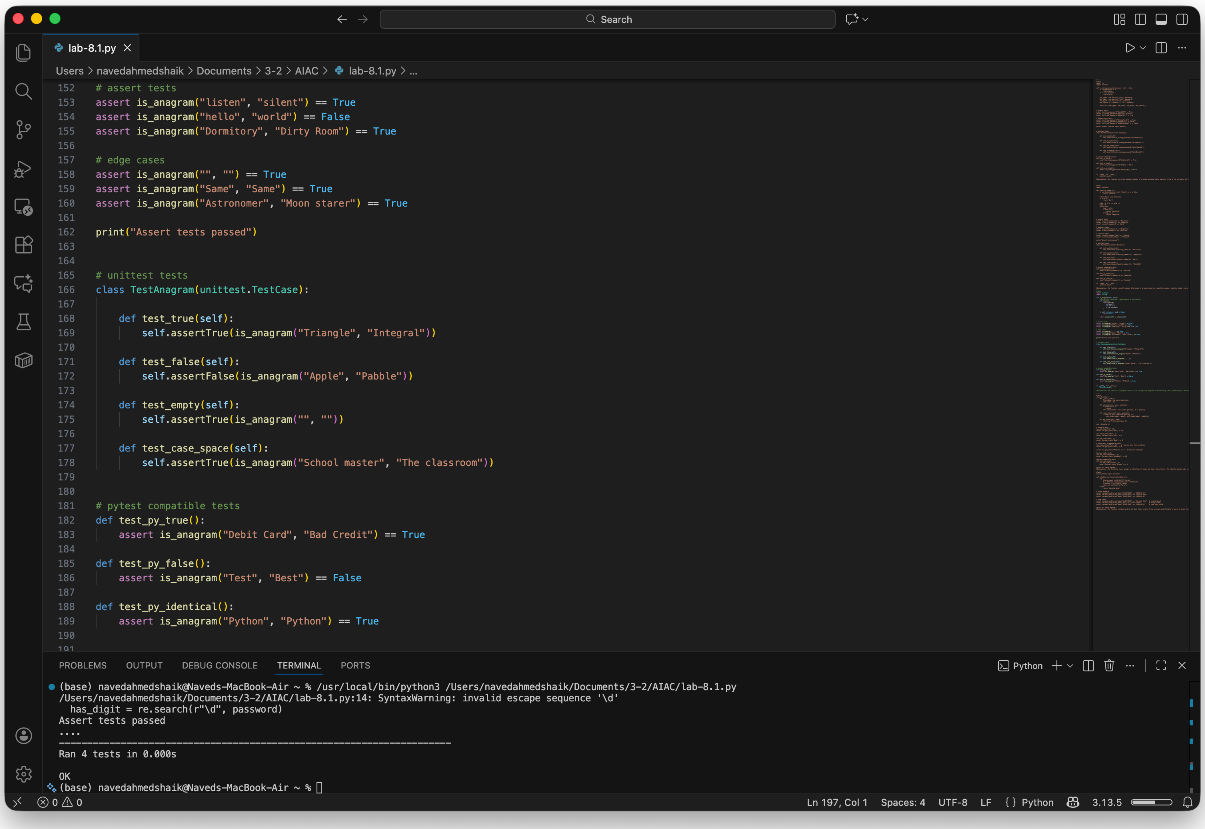
Expected Output #3:

Function correctly identifying anagrams and passing all AI-generated tests

EXPLANATION:

The function is_anagram checks if two strings are anagrams by normalizing them (converting to lowercase and removing spaces and punctuation) and then comparing the sorted characters. If the sorted characters of both strings are the same, they are anagrams. The assert statements test the function with various pairs of strings, including edge cases like empty strings and identical strings. The unittest class provides structured tests for different scenarios, while the pytest-compatible functions allow for additional testing in a more flexible manner.

SCREENSHOT OF GENERATED CODE:



```python
# assert tests
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
assert is_anagram("Same", "Same") == True
assert is_anagram("Astronomer", "Moon starer") == True

print("Assert tests passed")


# unittest tests
class TestAnagram(unittest.TestCase):

    def test_true(self):
        self.assertTrue(is_anagram("Triangle", "Integral"))

    def test_false(self):
        self.assertFalse(is_anagram("Apple", "Pabble"))

    def test_empty(self):
        self.assertTrue(is_anagram("", ""))

    def test_case_space(self):
        self.assertTrue(is_anagram("School master", "The classroom"))


# pytest compatible tests
def test_py_true():
    assert is_anagram("Debit Card", "Bad Credit") == True

def test_py_false():
    assert is_anagram("Test", "Best") == False

def test_py_identical():
    assert is_anagram("Python", "Python") == True

```

```
(base) navedahmedshaik@Naveds-MacBook-Air ~ % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
/Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py:14: SyntaxWarning: invalid escape sequence '\d'
  has_digit = re.search(r"\d", password)
Assert tests passed
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
(base) navedahmedshaik@Naveds-MacBook-Air ~ %
```

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

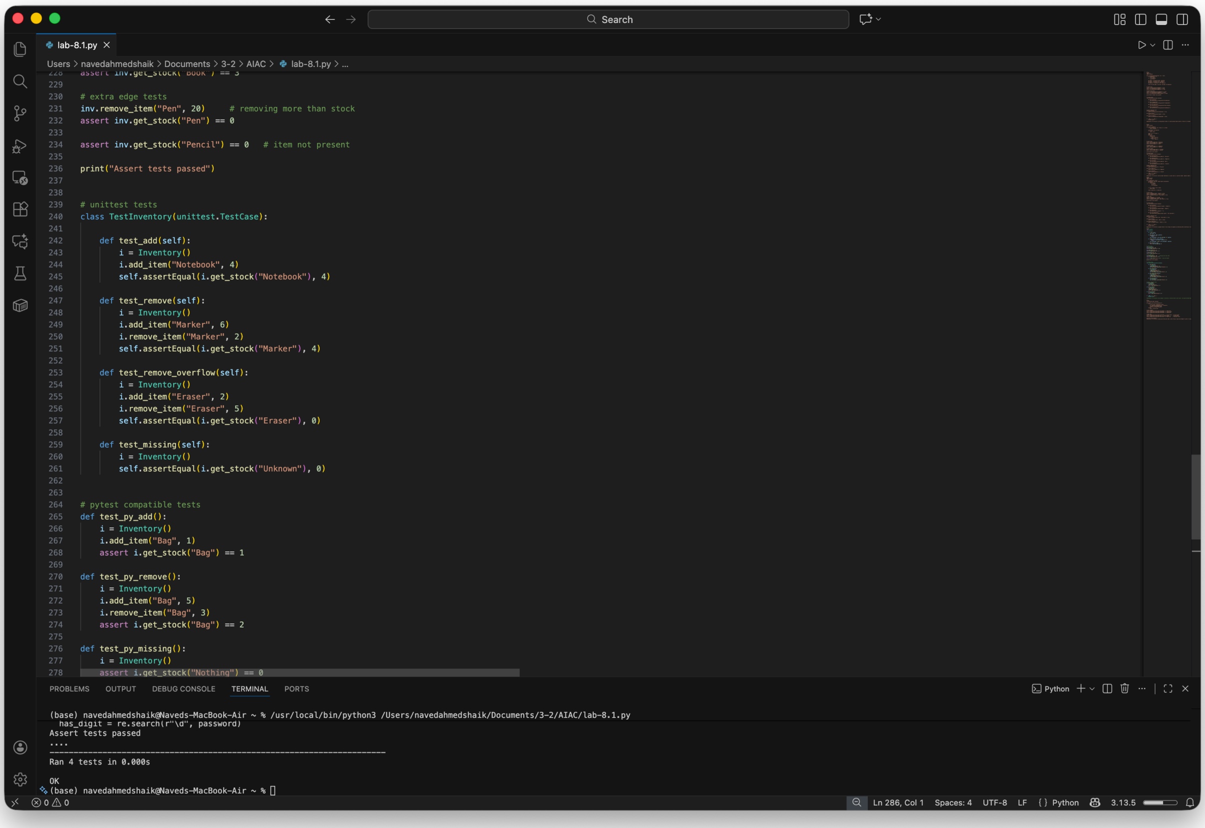
Expected Output #4:

Fully functional class passing all assertions.

EXPLANATION:

The Inventory class manages a simple stock system where you can add items, remove items, and check the stock of an item. The add_item method increases the stock of a given item by a specified quantity, while the remove_item method decreases the stock but ensures it does not go below zero. The get_stock method returns the current stock of an item or zero if the item is not present. The assert statements test the functionality of adding and removing items, as well as edge cases like removing more than the available stock and checking for items that do not exist. The unittest class provides structured tests for different scenarios, while the pytest-compatible functions allow for additional testing in a more flexible manner.

SCREENSHOT OF GENERATED CODE:



lab-8.1.py
Users > navedahmedshaik > Documents > 3-2 > AIAC > lab-8.1.py > ...

```python
assert inv.get_stock("Book") == 3

# extra edge tests
inv.remove_item("Pen", 20)     # removing more than stock
assert inv.get_stock("Pen") == 0

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

print("Assert tests passed")


# unittest tests
class TestInventory(unittest.TestCase):

    def test_add(self):
        i = Inventory()
        i.add_item("Notebook", 4)
        self.assertEqual(i.get_stock("Notebook"), 4)

    def test_remove(self):
        i = Inventory()
        i.add_item("Marker", 6)
        i.remove_item("Marker", 2)
        self.assertEqual(i.get_stock("Marker"), 4)

    def test_remove_overflow(self):
        i = Inventory()
        i.add_item("Eraser", 2)
        i.remove_item("Eraser", 5)
        self.assertEqual(i.get_stock("Eraser"), 0)

    def test_missing(self):
        i = Inventory()
        self.assertEqual(i.get_stock("Unknown"), 0)


# pytest compatible tests
def test_py_add():
    i = Inventory()
    i.add_item("Bag", 1)
    assert i.get_stock("Bag") == 1

def test_py_remove():
    i = Inventory()
    i.add_item("Bag", 5)
    i.remove_item("Bag", 3)
    assert i.get_stock("Bag") == 2

def test_py_missing():
    i = Inventory()
    assert i.get_stock("Nothing") == 0
```

```
(base) navedahmedshaik@Naveds-MacBook-Air ~ % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
  has_digit = re.search(r"\d", password)
Assert tests passed
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
(base) navedahmedshaik@Naveds-MacBook-Air ~ % 
```

Ln 286, Col 1   Spaces: 4   UTF-8   LF   { } Python   3.13.5

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"

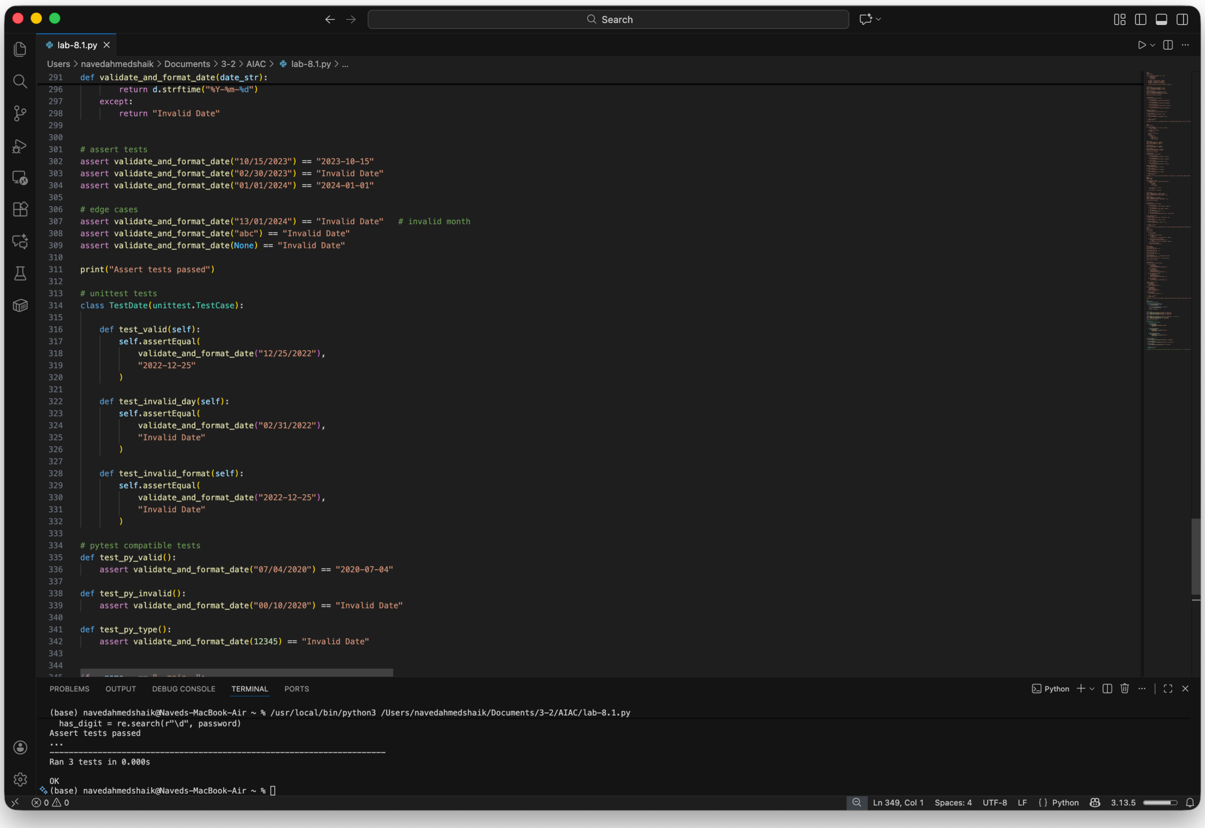
Expected Output #5:

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

EXPLANATION:

The function validate_and_format_date takes a string input and checks if it is a valid date in the format "MM/DD/YYYY". It uses the datetime module to attempt to parse the date string. If parsing is successful, it returns the date in the format "YYYY-MM-DD". If parsing fails (due to an invalid date or incorrect format), it returns "Invalid Date". The assert statements test the function with various valid and invalid date strings, including edge cases. The unittest class provides structured tests for different scenarios, while the pytest-compatible functions allow for additional testing in a more flexible manner.

SCREENSHOT OF GENERATED CODE:


lab-8.1.py
Users > navedahmedshaik > Documents > 3-2 > AIAC > lab-8.1.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Ln 349, Col 1 Spaces: 4 UTF-8 LF {} Python 3.13.5

```python
def validate_and_format_date(date_str):
        return d.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"


# assert tests
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("13/01/2024") == "Invalid Date"   # invalid month
assert validate_and_format_date("abc") == "Invalid Date"
assert validate_and_format_date(None) == "Invalid Date"

print("Assert tests passed")

# unittest tests
class TestDate(unittest.TestCase):

    def test_valid(self):
        self.assertEqual(
            validate_and_format_date("12/25/2022"),
            "2022-12-25"
        )

    def test_invalid_day(self):
        self.assertEqual(
            validate_and_format_date("02/31/2022"),
            "Invalid Date"
        )

    def test_invalid_format(self):
        self.assertEqual(
            validate_and_format_date("2022-12-25"),
            "Invalid Date"
        )

# pytest compatible tests
def test_py_valid():
    assert validate_and_format_date("07/04/2020") == "2020-07-04"

def test_py_invalid():
    assert validate_and_format_date("00/10/2020") == "Invalid Date"

def test_py_type():
    assert validate_and_format_date(12345) == "Invalid Date"
```

```
(base) navedahmedshaik@Naveds-MacBook-Air ~ % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-8.1.py
  has_digit = re.search(r"\d", password)
Assert tests passed
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
(base) navedahmedshaik@Naveds-MacBook-Air ~ % 
```