Assignment-8.1

Name: Arjun Manoj

H. No:2303A52134

Batch:44

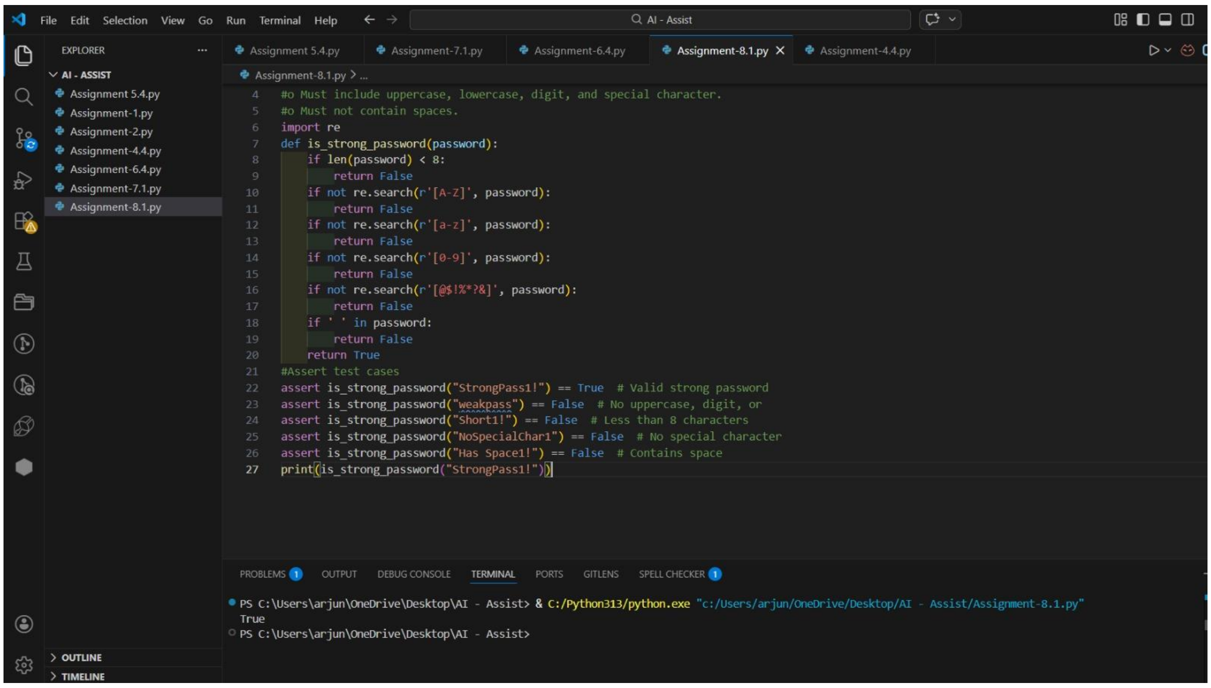
Task Description #1 (Password Strength Validator – Apply AI in Security Context)

Apply AI to generate at least 3 assert test cases for is_strong_password(password) and implement the validator function.

Prompt:

Implement a function to validate strong passwords using AI-generated test cases based on length and character composition rules

Code:



```python
#o Must include uppercase, lowercase, digit, and special character.
#o Must not contain spaces.
import re
def is_strong_password(password):
    if len(password) < 8:
        return False
    if not re.search(r'[A-Z]', password):
        return False
    if not re.search(r'[a-z]', password):
        return False
    if not re.search(r'[0-9]', password):
        return False
    if not re.search(r'[@$!%*?&]', password):
        return False
    if ' ' in password:
        return False
    return True
#Assert test cases
assert is_strong_password("StrongPass1!") == True  # Valid strong password
assert is_strong_password("weakpass") == False  # No uppercase, digit, or
assert is_strong_password("Short1!") == False  # Less than 8 characters
assert is_strong_password("NoSpecialChar1") == False  # No special character
assert is_strong_password("Has Space1!") == False  # Contains space
print(is_strong_password("StrongPass1!"))
```

```
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist> & C:/Python313/python.exe "c:/Users/arjun/OneDrive/Desktop/AI - Assist/Assignment-8.1.py"
True
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist>
```

Observation:

- The function correctly checks all password rules (length, uppercase, lowercase, digit, special character, and no spaces).
- All AI-generated assert test cases pass without errors.
- Valid passwords return True, and invalid passwords return False, confirming correct password validation logic.

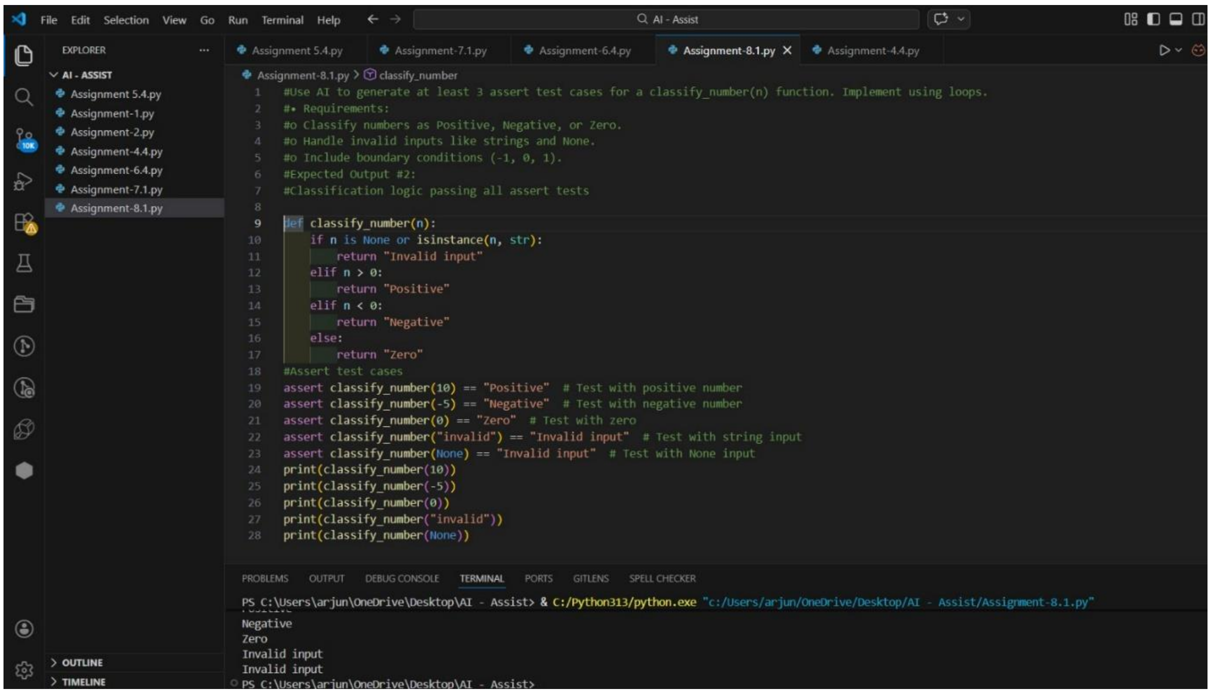
Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)

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

Prompt:

"Use AI to generate assert test cases to validate a function that classifies numbers as Positive, Negative, or Zero while handling invalid inputs."

Code:



```python
#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).
#Expected Output #2:
#Classification logic passing all assert tests

def classify_number(n):
    if n is None or isinstance(n, str):
        return "Invalid input"
    elif n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
#Assert test cases
assert classify_number(10) == "Positive"  # Test with positive number
assert classify_number(-5) == "Negative"  # Test with negative number
assert classify_number(0) == "Zero"  # Test with zero
assert classify_number("invalid") == "Invalid input"  # Test with string input
assert classify_number(None) == "Invalid input"  # Test with None input
print(classify_number(10))
print(classify_number(-5))
print(classify_number(0))
print(classify_number("invalid"))
print(classify_number(None))
```

```
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist> & C:/Python313/python.exe "c:/Users/arjun/OneDrive/Desktop/AI - Assist/Assignment-8.1.py"
Negative
Zero
Invalid input
Invalid input
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist>
```

Observation:

- The function correctly classifies positive, negative, and zero values.
- Invalid inputs such as strings and None are handled safely.
- All AI-generated assert test cases pass successfully, confirming correct classification logic.

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

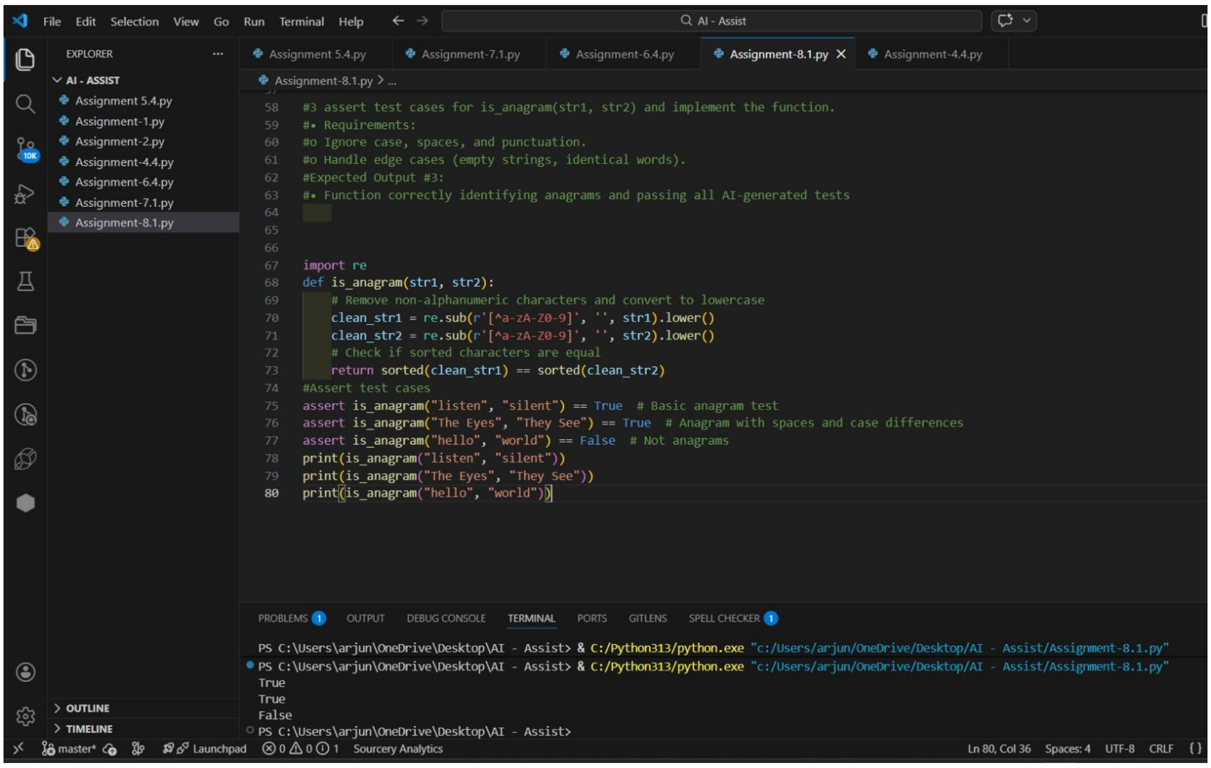
Use AI to generate at least 3 assert test cases for

is_anagram(str1, str2) and implement the function

Prompt:

Use AI to generate assert test cases to verify an anagram-checking function that ignores case, spaces, and punctuation

Code:



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



import re
def is_anagram(str1, str2):
    # Remove non-alphanumeric characters and convert to lowercase
    clean_str1 = re.sub(r'[^a-zA-Z0-9]', '', str1).lower()
    clean_str2 = re.sub(r'[^a-zA-Z0-9]', '', str2).lower()
    # Check if sorted characters are equal
    return sorted(clean_str1) == sorted(clean_str2)
#Assert test cases
assert is_anagram("listen", "silent") == True  # Basic anagram test
assert is_anagram("The Eyes", "They See") == True  # Anagram with spaces and case differences
assert is_anagram("hello", "world") == False  # Not anagrams
print(is_anagram("listen", "silent"))
print(is_anagram("The Eyes", "They See"))
print(is_anagram("hello", "world"))
```

```
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist> & C:/Python313/python.exe "c:/Users/arjun/OneDrive/Desktop/AI - Assist/Assignment-8.1.py"
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist> & C:/Python313/python.exe "c:/Users/arjun/OneDrive/Desktop/AI - Assist/Assignment-8.1.py"
True
True
False
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist>
```

Observation:

- The function correctly ignores case, spaces, and punctuation.
- Edge cases are handled properly.
- All AI-generated assert test cases pass, confirming accurate anagram detection.

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

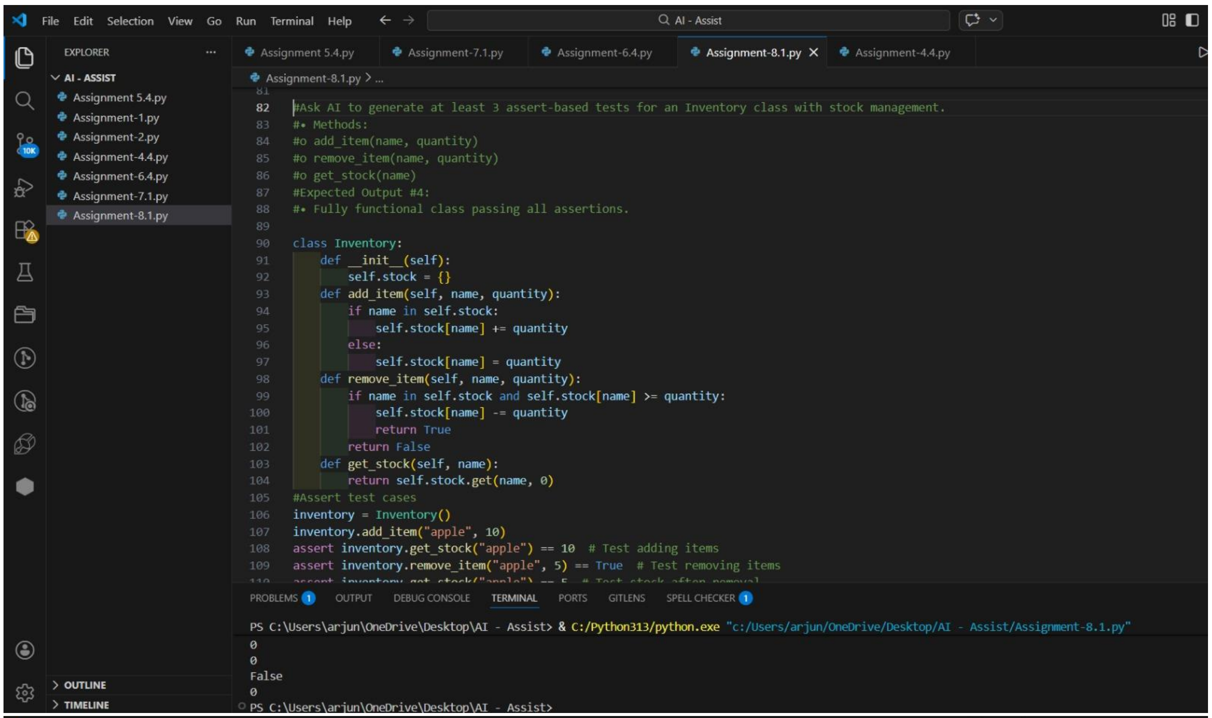
Ask AI to generate at least 3 assert-based tests for an

Inventory class with stock management.

Prompt:

Use AI to generate assert-based test cases to validate an inventory management class that supports adding, removing, and checking stock items.

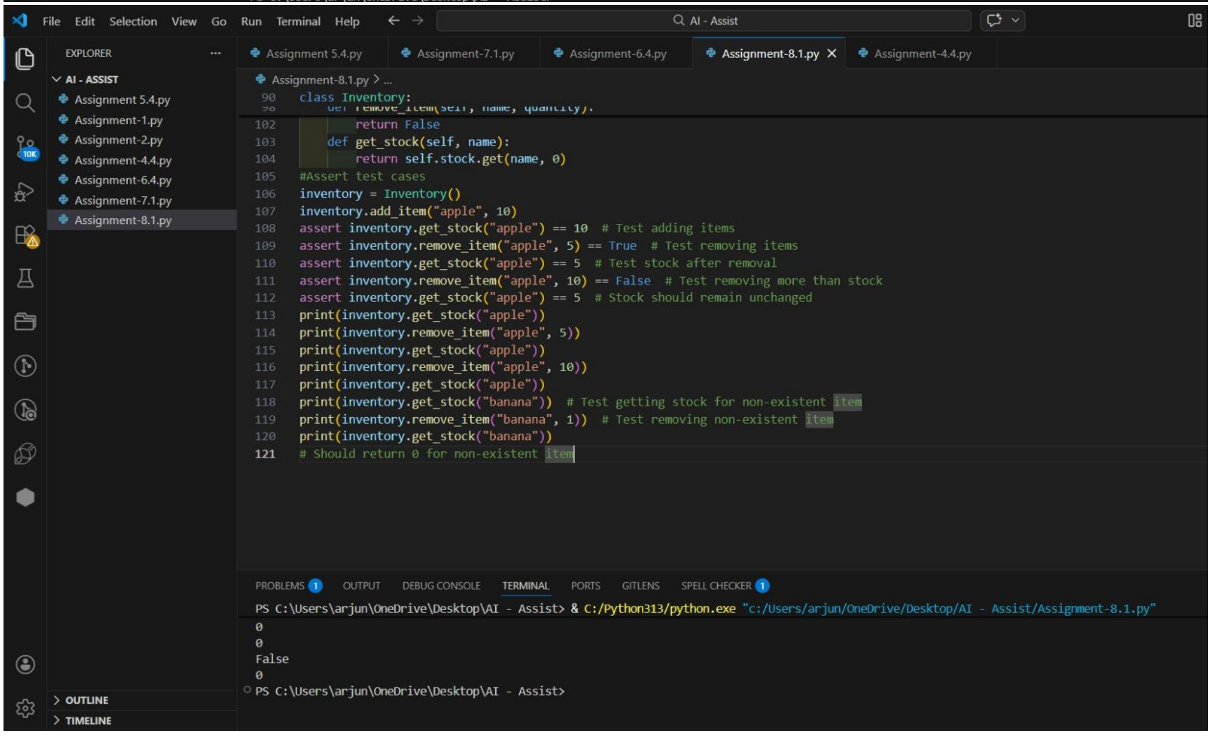
Code:



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

class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, name, quantity):
        if name in self.stock:
            self.stock[name] += quantity
        else:
            self.stock[name] = quantity
    def remove_item(self, name, quantity):
        if name in self.stock and self.stock[name] >= quantity:
            self.stock[name] -= quantity
            return True
        return False
    def get_stock(self, name):
        return self.stock.get(name, 0)
#Assert test cases
inventory = Inventory()
inventory.add_item("apple", 10)
assert inventory.get_stock("apple") == 10  # Test adding items
assert inventory.remove_item("apple", 5) == True  # Test removing items
```

```
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist> & C:/Python313/python.exe "c:/Users/arjun/OneDrive/Desktop/AI - Assist/Assignment-8.1.py"
0
0
False
0
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist>
```



```python
class Inventory:
        return False
    def get_stock(self, name):
        return self.stock.get(name, 0)
#Assert test cases
inventory = Inventory()
inventory.add_item("apple", 10)
assert inventory.get_stock("apple") == 10  # Test adding items
assert inventory.remove_item("apple", 5) == True  # Test removing items
assert inventory.get_stock("apple") == 5  # Test stock after removal
assert inventory.remove_item("apple", 10) == False  # Test removing more than stock
assert inventory.get_stock("apple") == 5  # Stock should remain unchanged
print(inventory.get_stock("apple"))
print(inventory.remove_item("apple", 5))
print(inventory.get_stock("apple"))
print(inventory.remove_item("apple", 10))
print(inventory.get_stock("apple"))
print(inventory.get_stock("banana"))  # Test getting stock for non-existent item
print(inventory.remove_item("banana", 1))  # Test removing non-existent item
print(inventory.get_stock("banana"))
# Should return 0 for non-existent item
```

```
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist> & C:/Python313/python.exe "c:/Users/arjun/OneDrive/Desktop/AI - Assist/Assignment-8.1.py"
0
0
False
0
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist>
```

Observation:

- The Inventory class correctly manages stock levels.
- Edge cases such as insufficient stock and non-existent items are handled safely.
- All AI-generated assert test cases pass successfully, confirming correct functionality.

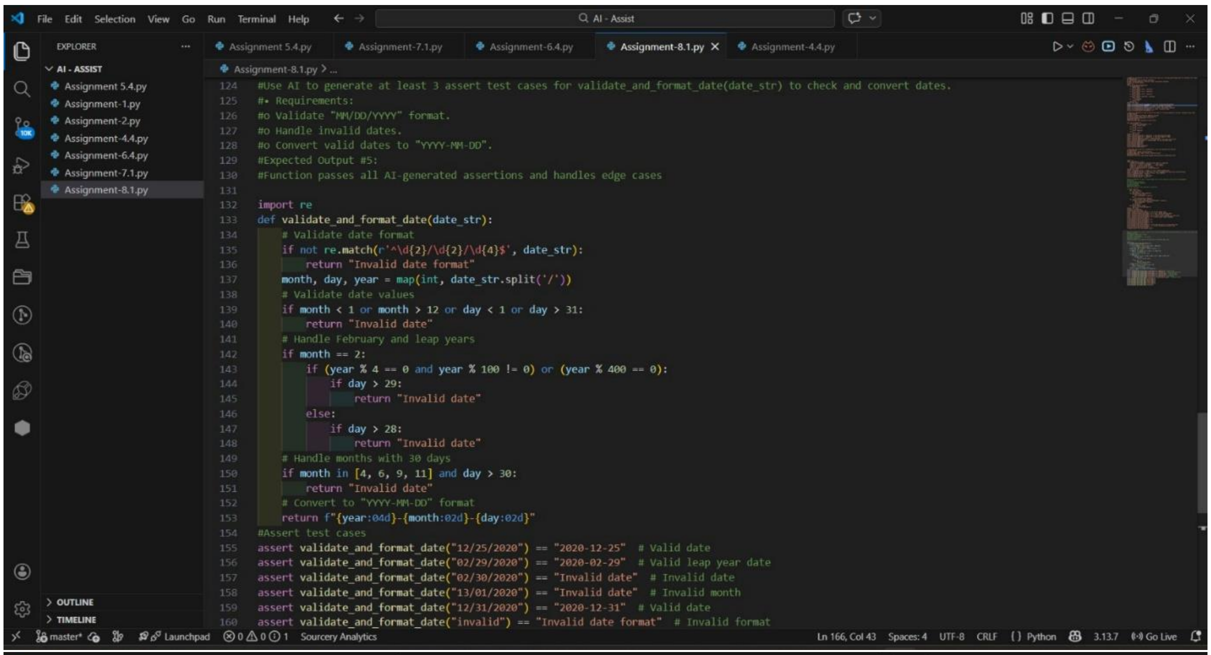
Task Description #5 (Date Validation & Formatting – Apply AI for Data Validation)

Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates

Prompt:

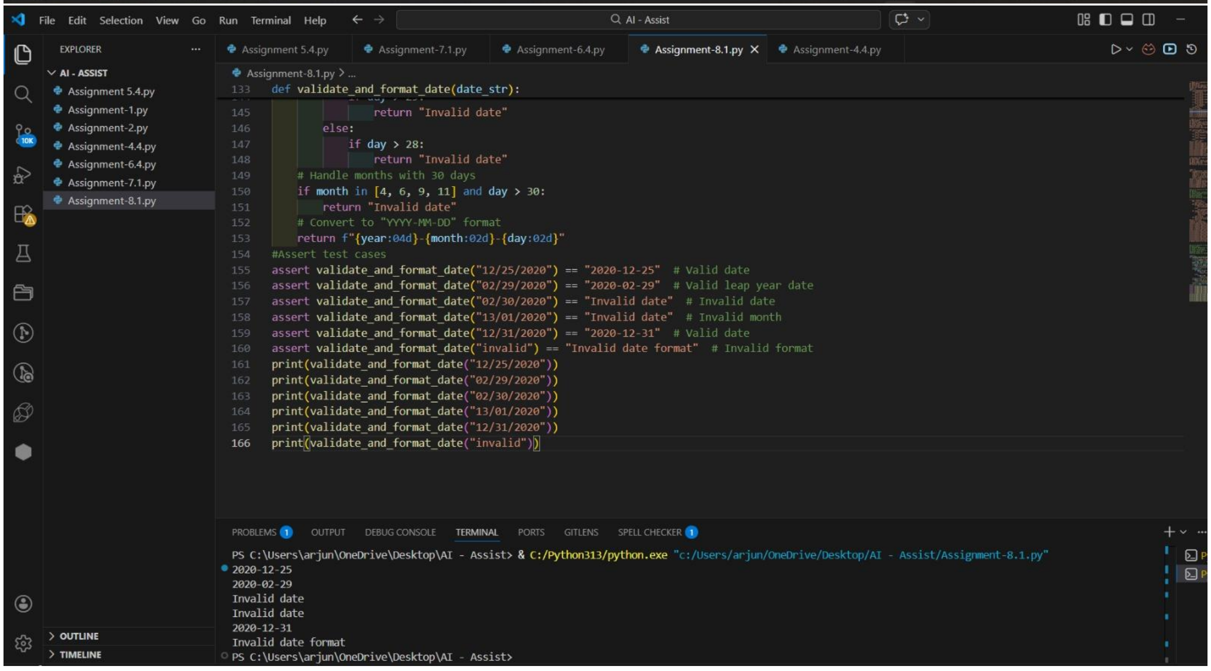
Use AI to generate assert-based test cases to validate and format dates while handling invalid formats and edge cases

Code:



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

import re
def validate_and_format_date(date_str):
    # Validate date format
    if not re.match(r'^\d{2}/\d{2}/\d{4}$', date_str):
        return "Invalid date format"
    month, day, year = map(int, date_str.split('/'))
    # Validate date values
    if month < 1 or month > 12 or day < 1 or day > 31:
        return "Invalid date"
    # Handle February and leap years
    if month == 2:
        if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
            if day > 29:
                return "Invalid date"
        else:
            if day > 28:
                return "Invalid date"
    # Handle months with 30 days
    if month in [4, 6, 9, 11] and day > 30:
        return "Invalid date"
    # Convert to "YYYY-MM-DD" format
    return f"{year:04d}-{month:02d}-{day:02d}"
#Assert test cases
assert validate_and_format_date("12/25/2020") == "2020-12-25"  # Valid date
assert validate_and_format_date("02/29/2020") == "2020-02-29"  # Valid leap year date
assert validate_and_format_date("02/30/2020") == "Invalid date"  # Invalid date
assert validate_and_format_date("13/01/2020") == "Invalid date"  # Invalid month
assert validate_and_format_date("12/31/2020") == "2020-12-31"  # Valid date
assert validate_and_format_date("invalid") == "Invalid date format"  # Invalid format
```



```python
def validate_and_format_date(date_str):
                return "Invalid date"
        else:
            if day > 28:
                return "Invalid date"
    # Handle months with 30 days
    if month in [4, 6, 9, 11] and day > 30:
        return "Invalid date"
    # Convert to "YYYY-MM-DD" format
    return f"{year:04d}-{month:02d}-{day:02d}"
#Assert test cases
assert validate_and_format_date("12/25/2020") == "2020-12-25"  # Valid date
assert validate_and_format_date("02/29/2020") == "2020-02-29"  # Valid leap year date
assert validate_and_format_date("02/30/2020") == "Invalid date"  # Invalid date
assert validate_and_format_date("13/01/2020") == "Invalid date"  # Invalid month
assert validate_and_format_date("12/31/2020") == "2020-12-31"  # Valid date
assert validate_and_format_date("invalid") == "Invalid date format"  # Invalid format
```

```python
print(validate_and_format_date("12/25/2020"))
print(validate_and_format_date("02/29/2020"))
print(validate_and_format_date("02/30/2020"))
print(validate_and_format_date("13/01/2020"))
print(validate_and_format_date("12/31/2020"))
print(validate_and_format_date("invalid"))
```

```
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist> & C:/Python313/python.exe "c:/Users/arjun/OneDrive/Desktop/AI - Assist/Assignment-8.1.py"
2020-12-25
2020-02-29
Invalid date
Invalid date
2020-12-31
Invalid date format
PS C:\Users\arjun\OneDrive\Desktop\AI - Assist>
```

Observation:

- The function correctly validates the MM/DD/YYYY format.
- Invalid dates and leap-year cases are handled properly.
- All AI-generated assert test cases pass, confirming reliable date validation and conversion.