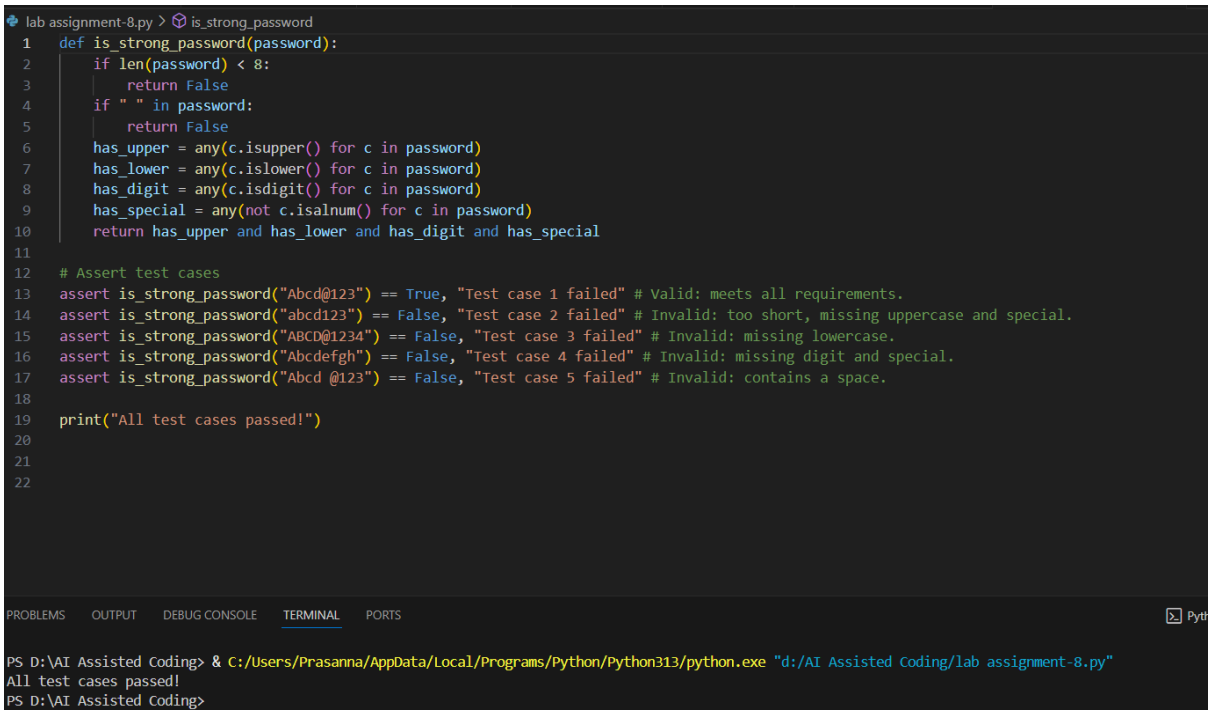
Assignment-8

Task-1:

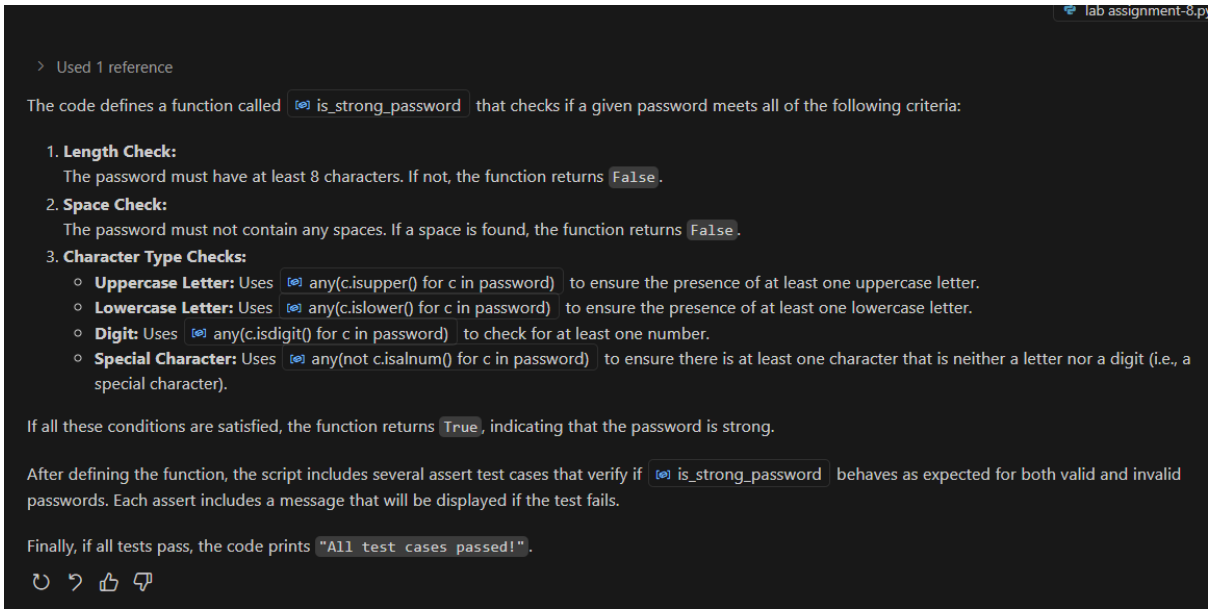
Prompt:

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

Code and Output:



Explanation:



Task 2:

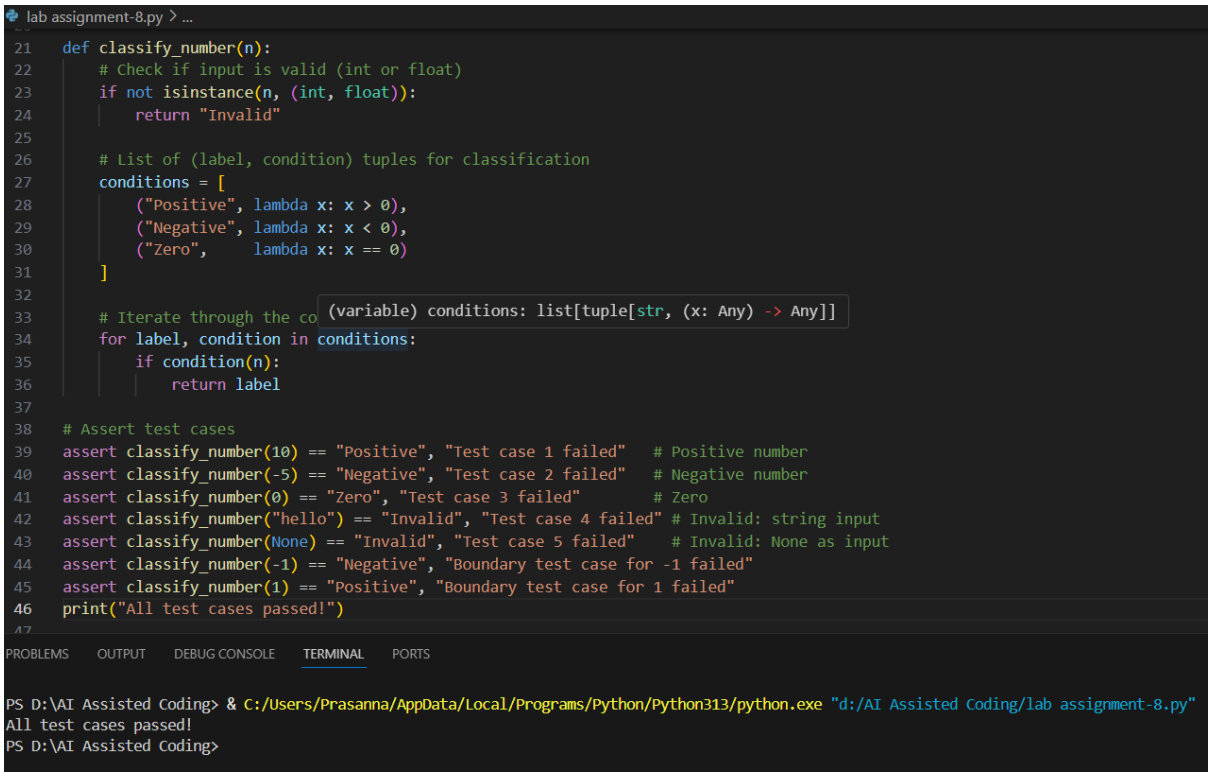
Prompt:

- 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).

Test Cases:

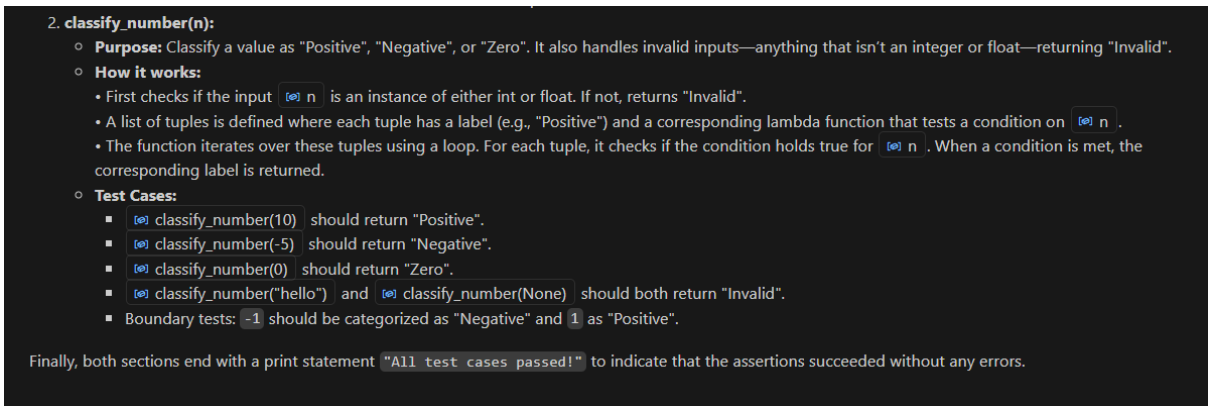
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"

Code and Output:



```
lab assignment-8.py > ...
21  def classify_number(n):
22      # Check if input is valid (int or float)
23      if not isinstance(n, (int, float)):
24          return "Invalid"
25
26      # List of (label, condition) tuples for classification
27      conditions = [
28          ("Positive", lambda x: x > 0),
29          ("Negative", lambda x: x < 0),
30          ("Zero",     lambda x: x == 0)
31      ]
32
33      # Iterate through the co  (variable) conditions: list[tuple[str, (x: Any) -> Any]]
34      for label, condition in conditions:
35          if condition(n):
36              return label
37
38  # Assert test cases
39  assert classify_number(10) == "Positive", "Test case 1 failed"   # Positive number
40  assert classify_number(-5) == "Negative", "Test case 2 failed"   # Negative number
41  assert classify_number(0) == "Zero", "Test case 3 failed"        # Zero
42  assert classify_number("hello") == "Invalid", "Test case 4 failed" # Invalid: string input
43  assert classify_number(None) == "Invalid", "Test case 5 failed"    # Invalid: None as input
44  assert classify_number(-1) == "Negative", "Boundary test case for -1 failed"
45  assert classify_number(1) == "Positive", "Boundary test case for 1 failed"
46  print("All test cases passed!")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\AI Assisted Coding> & C:/Users/Prasanna/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Assisted Coding/lab assignment-8.py"
All test cases passed!
PS D:\AI Assisted Coding>
```

Explanation:



2. **classify_number(n):**
   - **Purpose:** Classify a value as "Positive", "Negative", or "Zero". It also handles invalid inputs—anything that isn't an integer or float—returning "Invalid".
   - **How it works:**
     - First checks if the input `n` is an instance of either int or float. If not, returns "Invalid".
     - A list of tuples is defined where each tuple has a label (e.g., "Positive") and a corresponding lambda function that tests a condition on `n`.
     - The function iterates over these tuples using a loop. For each tuple, it checks if the condition holds true for `n`. When a condition is met, the corresponding label is returned.
   - **Test Cases:**
     - `classify_number(10)` should return "Positive".
     - `classify_number(-5)` should return "Negative".
     - `classify_number(0)` should return "Zero".
     - `classify_number("hello")` and `classify_number(None)` should both return "Invalid".
     - Boundary tests: `-1` should be categorized as "Negative" and `1` as "Positive".

Finally, both sections end with a print statement `"All test cases passed!"` to indicate that the assertions succeeded without any errors.

Task 3:

Prompt:

Anagram Checker

- Test cases for is_anagram(str1, str2) and implement the function.

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

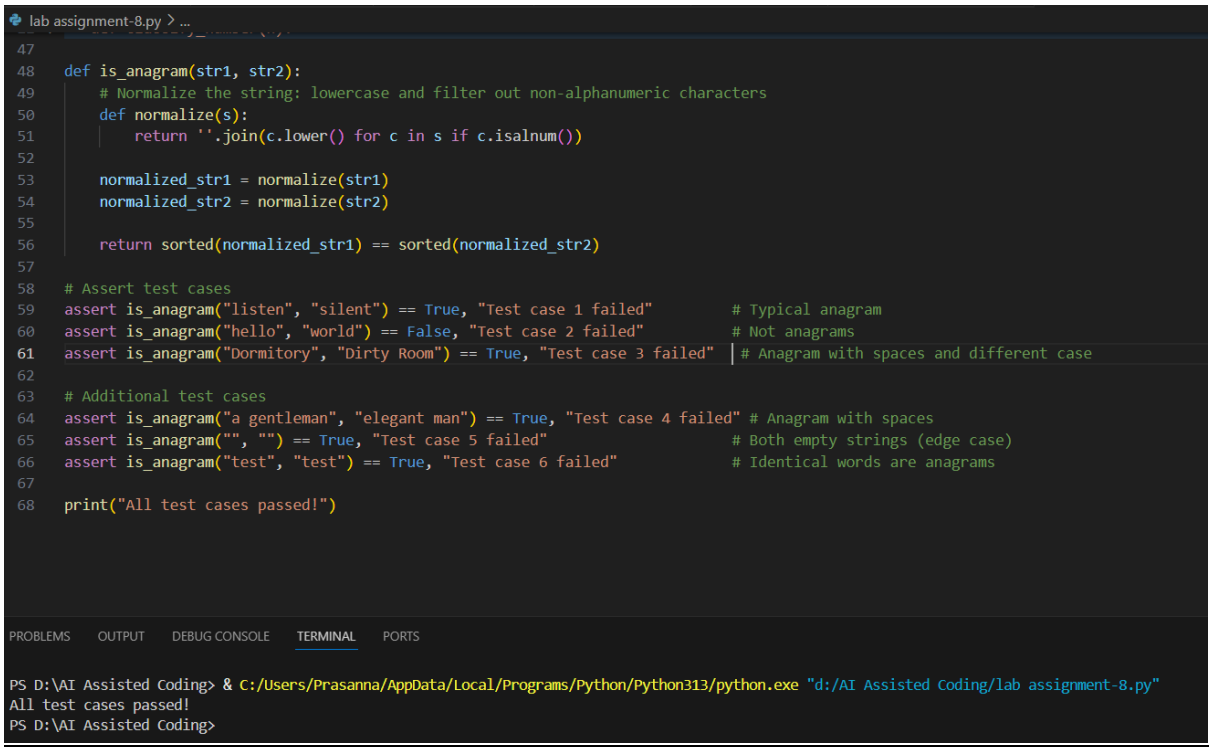
Test Cases:

assert is_anagram("listen", "silent") == True

assert is_anagram("hello", "world") == False

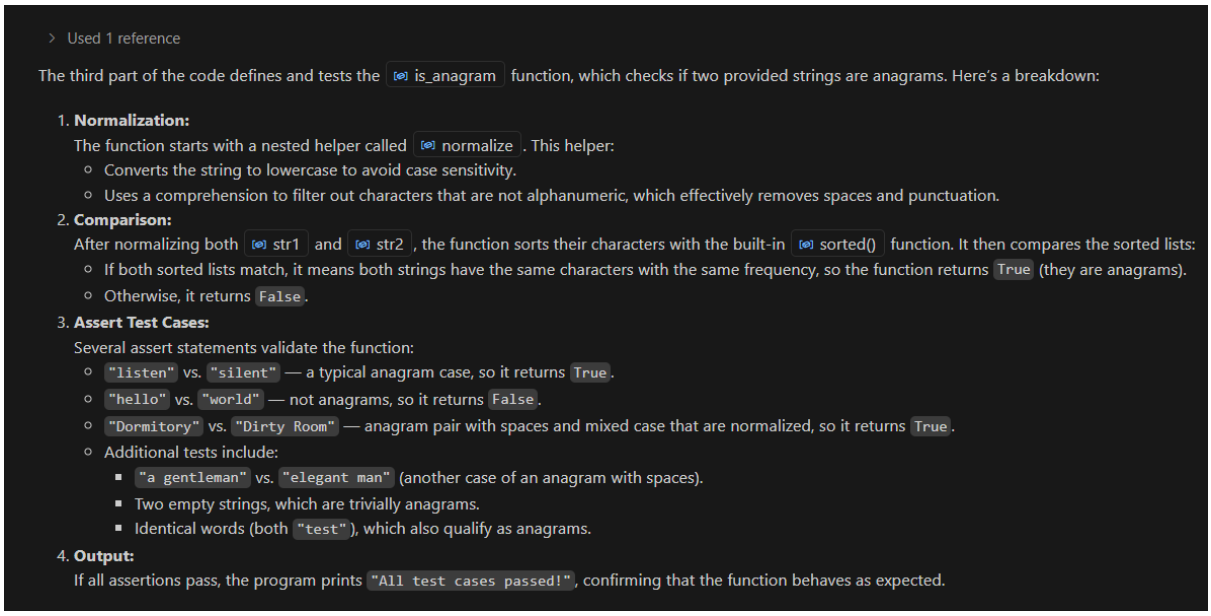
assert is_anagram("Dormitory", "Dirty Room") == True

Code and Output:



```
lab assignment-8.py > ...
47
48  def is_anagram(str1, str2):
49      # Normalize the string: lowercase and filter out non-alphanumeric characters
50      def normalize(s):
51          return ''.join(c.lower() for c in s if c.isalnum())
52
53      normalized_str1 = normalize(str1)
54      normalized_str2 = normalize(str2)
55
56      return sorted(normalized_str1) == sorted(normalized_str2)
57
58  # Assert test cases
59  assert is_anagram("listen", "silent") == True, "Test case 1 failed"         # Typical anagram
60  assert is_anagram("hello", "world") == False, "Test case 2 failed"          # Not anagrams
61  assert is_anagram("Dormitory", "Dirty Room") == True, "Test case 3 failed"  # Anagram with spaces and different case
62
63  # Additional test cases
64  assert is_anagram("a gentleman", "elegant man") == True, "Test case 4 failed" # Anagram with spaces
65  assert is_anagram("", "") == True, "Test case 5 failed"                     # Both empty strings (edge case)
66  assert is_anagram("test", "test") == True, "Test case 6 failed"             # Identical words are anagrams
67
68  print("All test cases passed!")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\AI Assisted Coding> & C:/Users/Prasanna/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Assisted Coding/lab assignment-8.py"
All test cases passed!
PS D:\AI Assisted Coding>
```

Explanation:



> Used 1 reference

The third part of the code defines and tests the `is_anagram` function, which checks if two provided strings are anagrams. Here's a breakdown:

1. **Normalization:**
   The function starts with a nested helper called `normalize`. This helper:
   - Converts the string to lowercase to avoid case sensitivity.
   - Uses a comprehension to filter out characters that are not alphanumeric, which effectively removes spaces and punctuation.
2. **Comparison:**
   After normalizing both `str1` and `str2`, the function sorts their characters with the built-in `sorted()` function. It then compares the sorted lists:
   - If both sorted lists match, it means both strings have the same characters with the same frequency, so the function returns `True` (they are anagrams).
   - Otherwise, it returns `False`.
3. **Assert Test Cases:**
   Several assert statements validate the function:
   - `"listen"` vs. `"silent"` — a typical anagram case, so it returns `True`.
   - `"hello"` vs. `"world"` — not anagrams, so it returns `False`.
   - `"Dormitory"` vs. `"Dirty Room"` — anagram pair with spaces and mixed case that are normalized, so it returns `True`.
   - Additional tests include:
     - `"a gentleman"` vs. `"elegant man"` (another case of an anagram with spaces).
     - Two empty strings, which are trivially anagrams.
     - Identical words (both `"test"`), which also qualify as anagrams.
4. **Output:**
   If all assertions pass, the program prints `"All test cases passed!"`, confirming that the function behaves as expected.

Task 4:

Prompt:

(Inventory Class –Simulate Real-World Inventory System)

- Tests for an Inventory class with stock management.
- Methods:
  - add_item(name, quantity)
  - remove_item(name, quantity)
  - get_stock(name)

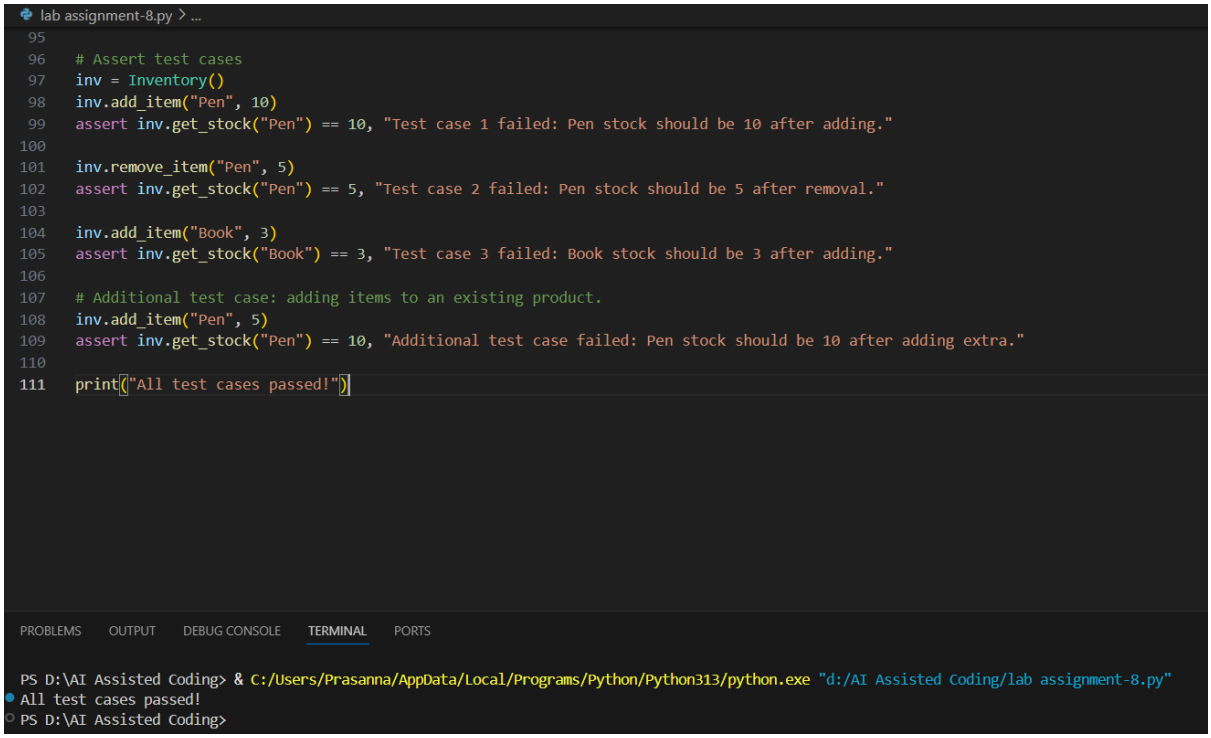
Test Cases:

```
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

Code & Output:

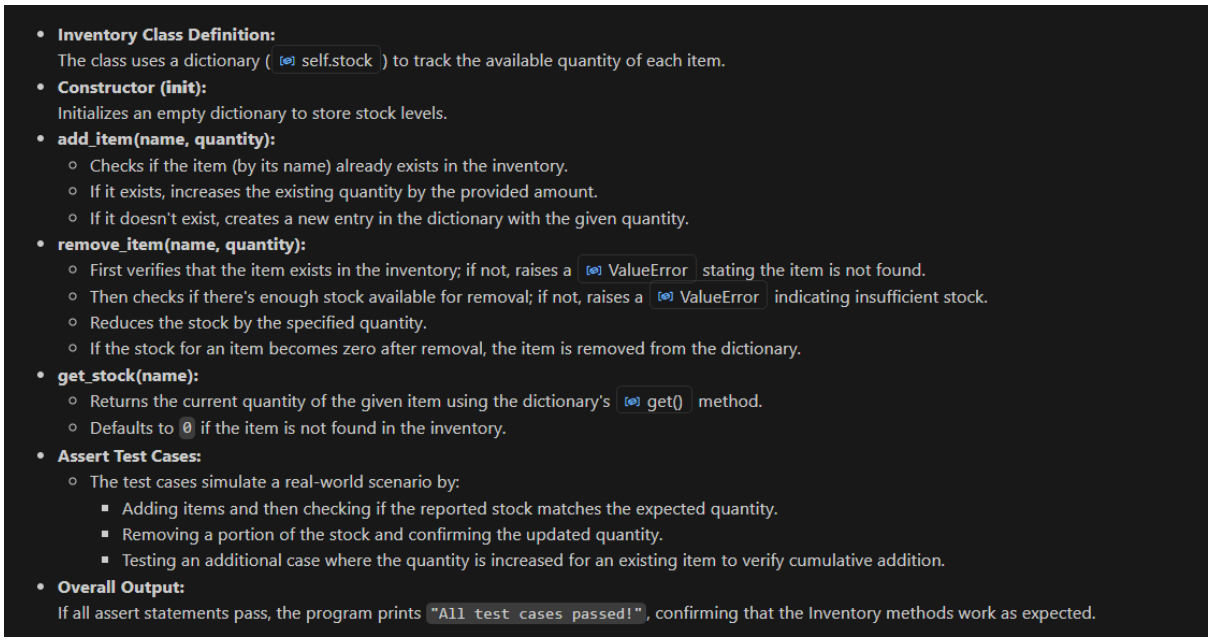
```python
class Inventory:
    def __init__(self):
        self.stock = {}

    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 not in self.stock:
            raise ValueError("Item not found in inventory")
        if self.stock[name] < quantity:
            raise ValueError("Not enough stock to remove")
        self.stock[name] -= quantity
        if self.stock[name] == 0:
            del self.stock[name]

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



```python
# Assert test cases
inv = Inventory()
inv.add_item("Pen", 10)
assert inv.get_stock("Pen") == 10, "Test case 1 failed: Pen stock should be 10 after adding."

inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5, "Test case 2 failed: Pen stock should be 5 after removal."

inv.add_item("Book", 3)
assert inv.get_stock("Book") == 3, "Test case 3 failed: Book stock should be 3 after adding."

# Additional test case: adding items to an existing product.
inv.add_item("Pen", 5)
assert inv.get_stock("Pen") == 10, "Additional test case failed: Pen stock should be 10 after adding extra."

print("All test cases passed!")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\AI Assisted Coding> & C:/Users/Prasanna/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Assisted Coding/lab assignment-8.py"
All test cases passed!
PS D:\AI Assisted Coding>
```

Explanation:



- **Inventory Class Definition:**
  The class uses a dictionary (`self.stock`) to track the available quantity of each item.
- **Constructor (init):**
  Initializes an empty dictionary to store stock levels.
- **add_item(name, quantity):**
  - Checks if the item (by its name) already exists in the inventory.
  - If it exists, increases the existing quantity by the provided amount.
  - If it doesn't exist, creates a new entry in the dictionary with the given quantity.
- **remove_item(name, quantity):**
  - First verifies that the item exists in the inventory; if not, raises a `ValueError` stating the item is not found.
  - Then checks if there's enough stock available for removal; if not, raises a `ValueError` indicating insufficient stock.
  - Reduces the stock by the specified quantity.
  - If the stock for an item becomes zero after removal, the item is removed from the dictionary.
- **get_stock(name):**
  - Returns the current quantity of the given item using the dictionary's `get()` method.
  - Defaults to `0` if the item is not found in the inventory.
- **Assert Test Cases:**
  - The test cases simulate a real-world scenario by:
    - Adding items and then checking if the reported stock matches the expected quantity.
    - Removing a portion of the stock and confirming the updated quantity.
    - Testing an additional case where the quantity is increased for an existing item to verify cumulative addition.
- **Overall Output:**
  If all assert statements pass, the program prints `"All test cases passed!"`, confirming that the Inventory methods work as expected.

Task 5:

Prompt:

Date Validation & Formatting

- 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".

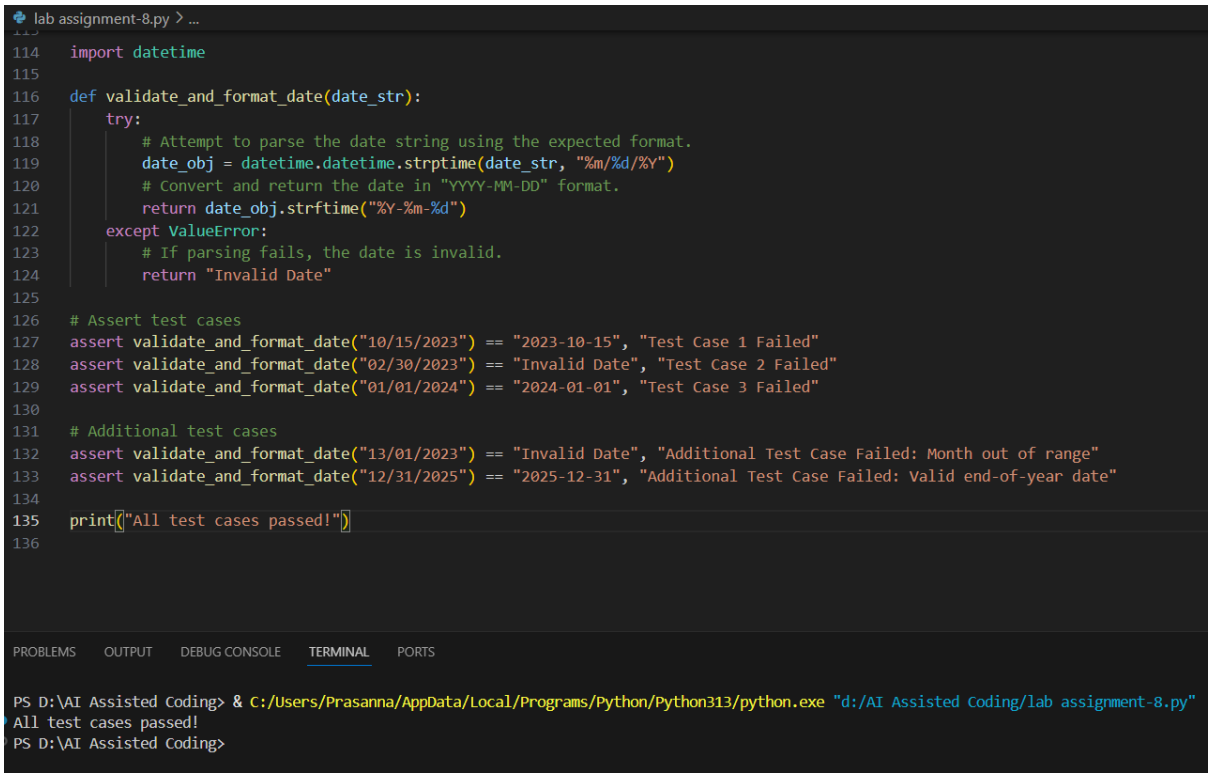
Test Cases:

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"

Code & Output:



```
lab assignment-8.py > ...
114  import datetime
115
116  def validate_and_format_date(date_str):
117      try:
118          # Attempt to parse the date string using the expected format.
119          date_obj = datetime.datetime.strptime(date_str, "%m/%d/%Y")
120          # Convert and return the date in "YYYY-MM-DD" format.
121          return date_obj.strftime("%Y-%m-%d")
122      except ValueError:
123          # If parsing fails, the date is invalid.
124          return "Invalid Date"
125
126  # Assert test cases
127  assert validate_and_format_date("10/15/2023") == "2023-10-15", "Test Case 1 Failed"
128  assert validate_and_format_date("02/30/2023") == "Invalid Date", "Test Case 2 Failed"
129  assert validate_and_format_date("01/01/2024") == "2024-01-01", "Test Case 3 Failed"
130
131  # Additional test cases
132  assert validate_and_format_date("13/01/2023") == "Invalid Date", "Additional Test Case Failed: Month out of range"
133  assert validate_and_format_date("12/31/2025") == "2025-12-31", "Additional Test Case Failed: Valid end-of-year date"
134
135  print("All test cases passed!")
136

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\AI Assisted Coding> & C:/Users/Prasanna/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Assisted Coding/lab assignment-8.py"
All test cases passed!
PS D:\AI Assisted Coding>
```

Explanation:

The fifth code defines and tests a function for validating and formatting dates. Here's a breakdown:

1. **Importing the Library:**
   - The code imports Python's `datetime` module, which is used to work with dates and times.
2. **Function: validate_and_format_date(date_str):**
   - **Purpose:** Validates that the provided string follows the "MM/DD/YYYY" format and represents a valid date.
   - **Implementation:**
     - The function attempts to parse the input string using `datetime.datetime.strptime(date_str, "%m/%d/%Y")`.
     - If successful, it means the date is valid, and the function converts it to the format "YYYY-MM-DD" using `strftime` and returns it.
     - If parsing fails (due to an incorrect format or an invalid date), a `ValueError` is raised, and the function catches it, returning the string `"Invalid Date"`.
3. **Assert Test Cases:**
   - The code includes several assert statements to test the function with different scenarios:
     - `assert validate_and_format_date("10/15/2023") == "2023-10-15"` verifies a valid date is correctly formatted.
     - `assert validate_and_format_date("02/30/2023") == "Invalid Date"` checks that an invalid date (February 30) returns `"Invalid Date"`.
     - `assert validate_and_format_date("01/01/2024") == "2024-01-01"` confirms another valid date conversion.
   - **Additional Test Cases:**
     - `assert validate_and_format_date("13/01/2023") == "Invalid Date"` verifies that an out-of-range month returns an invalid result.
     - `assert validate_and_format_date("12/31/2025") == "2025-12-31"` checks a valid end-of-year date.
4. **Output:**
   - If all the assertions pass without any errors, the message `"All test cases passed!"` is printed at the end.

Overall, this code effectively validates the date format, handles edge cases like invalid dates or formatting issues, and converts valid dates to the desired "YYYY-MM-DD" format.