# Assignment 8.1

**Task 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

**Prompt:**

Generate at least 3 assert test cases for a function is\_strong\_password(password) that validates passwords. The rules are:

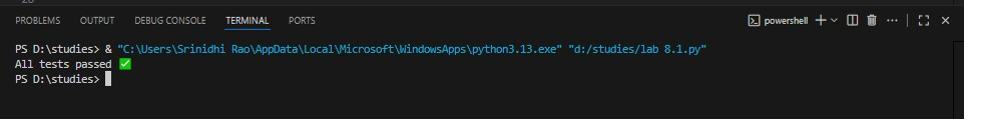
* Minimum 8 characters,
* Must include uppercase, lowercase, digit, and special character,
* Must not contain spaces.  
  Provide asserts where the expected output is True and others where it is False.

**Code generated:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Output**

****

**Observation**

The code defines a function is\_strong\_password that checks if a password meets security rules. It verifies the length (minimum eight characters), absence of spaces, and presence of at least one uppercase letter, lowercase letter, digit, and special character. Using regular expressions, each condition is tested. Assert statements confirm correctness by checking valid and invalid passwords.

**Task 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.

**Prompt:** *"Generate at least 3 assert test cases for a function classify\_number(n) that classifies input as Positive, Negative, or Zero. The function should handle invalid inputs (like strings, None) and include boundary cases (-1, 0, 1). Provide asserts where the expected outputs match the classification."*

**Code generated:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Output:A black screen with blue text

AI-generated content may be incorrect.Observation**

The classify\_number function checks if input is a number and classifies it as Positive, Negative, or Zero using a simple loop. Invalid inputs like strings or None return "Invalid Input". Assert statements test normal, invalid, and boundary cases, ensuring the logic is correct.

**Task 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:  
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

**Prompt:**

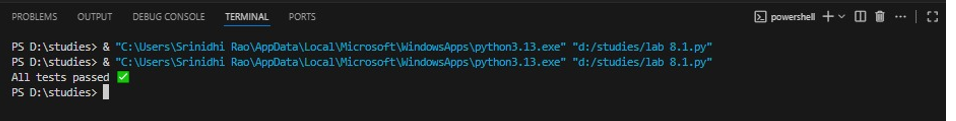
Generate at least 3 assert test cases for a function is\_anagram(str1, str2) that checks if two strings are anagrams, ignoring case, spaces, and punctuation. Implement the function so it passes all tests, including edge cases like empty strings and identical words.

**Code generated:**

A screenshot of a computer

AI-generated content may be incorrect.

**Output:**

****

**Observation**

1.The is\_anagram function cleans both input strings by removing spaces and punctuation, converting to lowercase, and sorting the characters.

2.It then compares the cleaned versions to determine if they are anagrams.

3.The assert test cases check for typical anagrams, non-anagrams, case insensitivity, spaces, punctuation, empty strings, and identical words.

4.The function passes all tests, correctly identifying anagrams under the specified requirements

**Task 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.

**Prompt:**Generate at least 3 assert-based test cases for an Inventory class with methods: add\_item(name, quantity), remove\_item(name, quantity), and get\_stock(name). Implement the class so it passes all tests.

**Code generated:**

A screen shot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Output:**

**A computer screen with blue text

AI-generated content may be incorrect.**

**Observation**

* The Inventory class manages stock using a dictionary.
* add\_item increases the quantity for an item, creating it if needed.
* remove\_item decreases the quantity but never below zero.
* get\_stock returns the current quantity or zero if the item doesn't exist.
* The assert tests check adding, removing, and retrieving stock for different items, ensuring the class works as expected.

**Task 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"

**Prompt:**  
Generate at least 3 assert-based test cases for an [Inventory](vscode-file://vscode-app/c:/Users/Srinidhi%20Rao/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) class with methods: [add\_item(name, quantity)](vscode-file://vscode-app/c:/Users/Srinidhi%20Rao/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o "), [remove\_item(name, quantity)](vscode-file://vscode-app/c:/Users/Srinidhi%20Rao/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o "), and [get\_stock(name)](vscode-file://vscode-app/c:/Users/Srinidhi%20Rao/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o "). The class should handle negative quantities and removing more items than are in stock by raising a [ValueError](vscode-file://vscode-app/c:/Users/Srinidhi%20Rao/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o "). Implement the class so it passes all tests, including edge cases like removing all items or checking stock for a non-existent item.

**Code generated:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Output:**

**A computer screen with blue text

AI-generated content may be incorrect.**

**Observation:**

1. The program defines a function validate\_and\_format\_date(date\_str) to check whether a given string is a **valid date** in MM/DD/YYYY format.
2. If the string is a valid date, it converts and returns the date in the **YYYY-MM-DD format**.
3. If the string is invalid (wrong format, invalid month/day, or non-date input), the function returns **"Invalid Date"**.
4. Several **assert test cases** are written to automatically verify the correctness of the function for:
   * Valid date inputs (e.g., "10/15/2023" → "2023-10-15")
   * Invalid dates (e.g., "02/30/2023", "13/01/2024")
   * Completely wrong strings (e.g., "abc")
5. If all test cases succeed, the program prints **“All date validation tests passed”**.