

# **Assignment**

**PIN : 2503A52L17**

Test-Driven Development with AI – Generating and Working with Test Cases :

## **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.

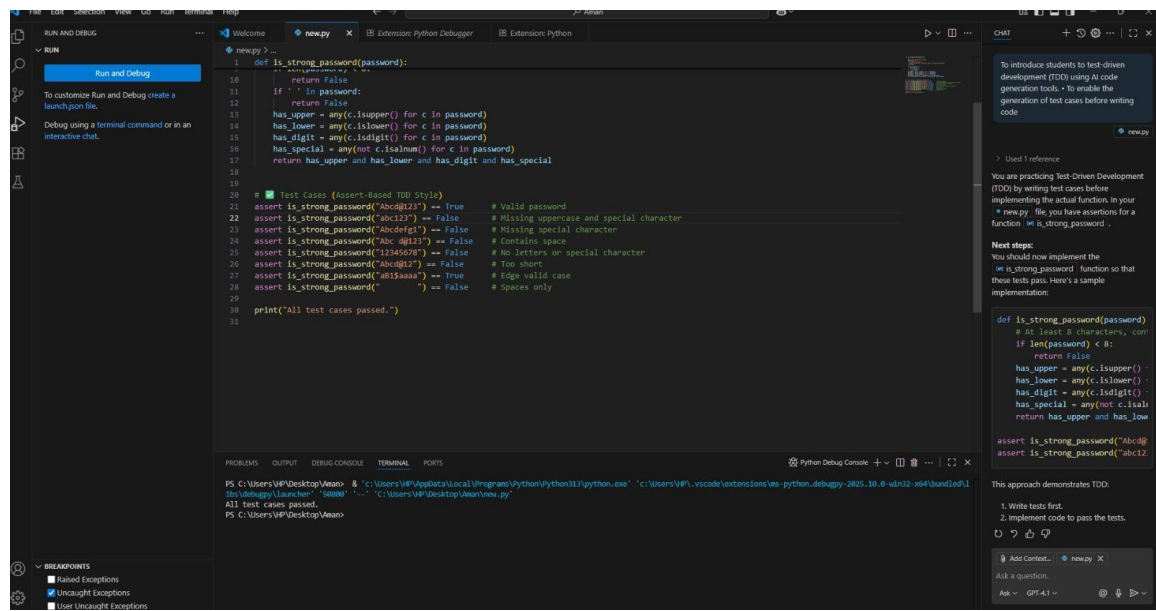
**Example Assert Test Cases:**

```
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True
```

**Expected Output #1:**

- Password validation logic passing all AI-generated test cases

**Code , Output :**



## 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:**
  - Classify numbers as Positive, Negative, or Zero.
  - Handle invalid inputs like strings and None.
  - Include boundary conditions (-1, 0, 1).

### Example Assert Test Cases:

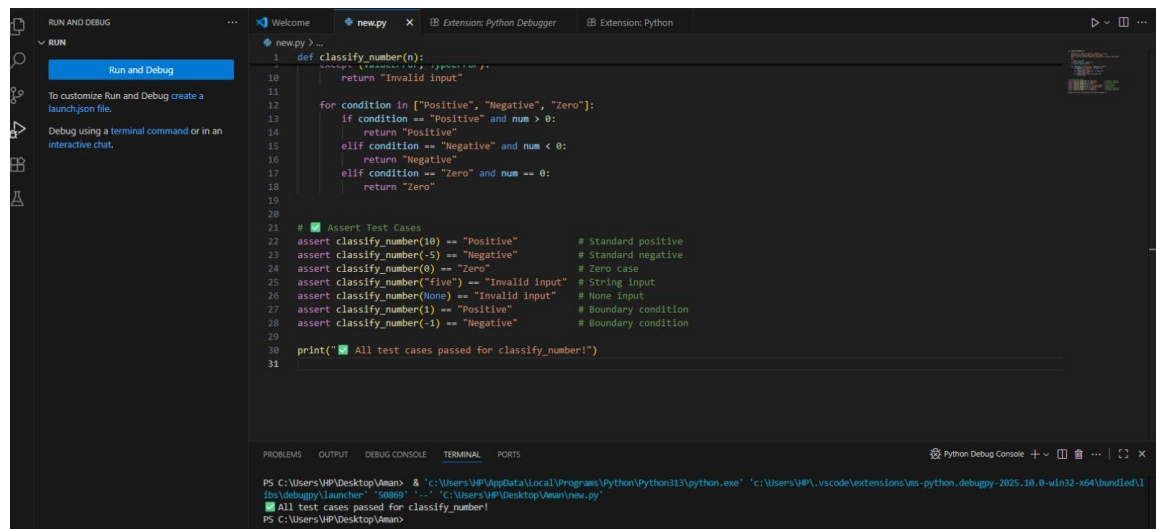
`assert classify_number(10) == "Positive"`

`assert classify_number(-5) == "Negative"`

`assert classify_number(0) == "Zero"`

### Expected Output #2:

- Classification logic passing all assert tests.



The screenshot shows a VS Code editor with a Python file named 'new.py'. The code defines a function 'classify\_number(n)' that returns 'Invalid input' for non-numeric values, 'Positive' for numbers greater than 0, 'Negative' for numbers less than 0, and 'Zero' for numbers equal to 0. Below the function, there are several assert test cases for various inputs, including standard positive/negative numbers, zero, string inputs, and boundary conditions. The script concludes with a print statement indicating that all test cases passed.

```
1 def classify_number(n):
2     if not isinstance(n, (int, float)):
3         return "Invalid input"
4
5     for condition in ["Positive", "Negative", "Zero"]:
6         if condition == "Positive" and num > 0:
7             return "Positive"
8         elif condition == "Negative" and num < 0:
9             return "Negative"
10        elif condition == "Zero" and num == 0:
11            return "Zero"
12
13 # Assert Test Cases
14 assert classify_number(10) == "Positive" # Standard positive
15 assert classify_number(-5) == "Negative" # Standard negative
16 assert classify_number(0) == "Zero" # Zero case
17 assert classify_number("five") == "Invalid input" # String input
18 assert classify_number(None) == "Invalid input" # None input
19 assert classify_number(1) == "Positive" # Boundary condition
20 assert classify_number(-1) == "Negative" # Boundary condition
21
22 print("All test cases passed for classify_number!")
23
```

The terminal at the bottom shows the command prompt and the output of the script, confirming that all test cases passed.

## 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:**

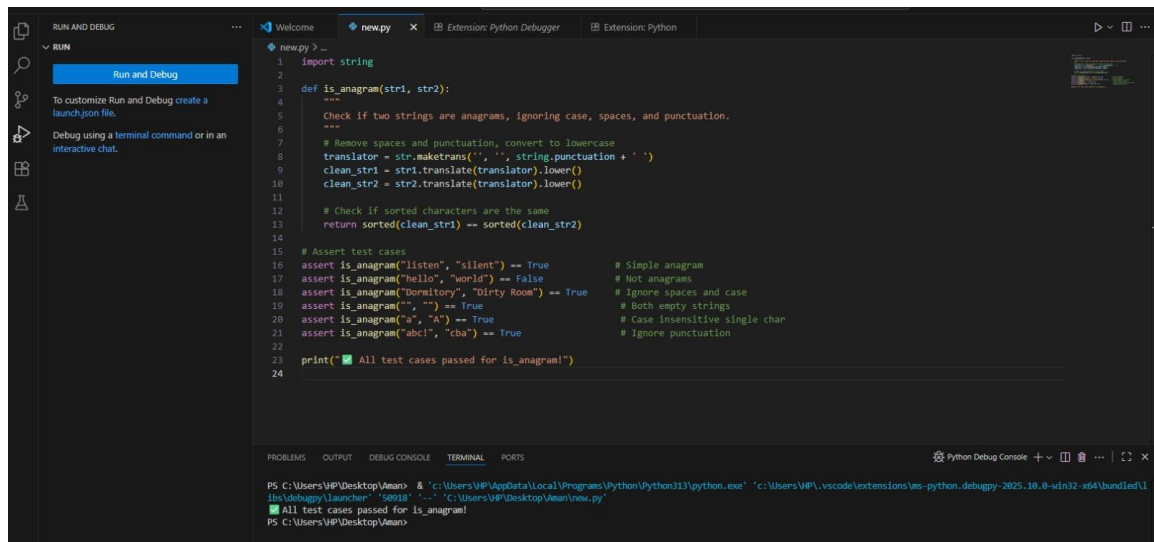
```
assert is_anagram("listen", "silent") == True
```

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

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

**Expected Output #3:**

- Function correctly identifying anagrams and passing all AI-generated tests.



## 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:**

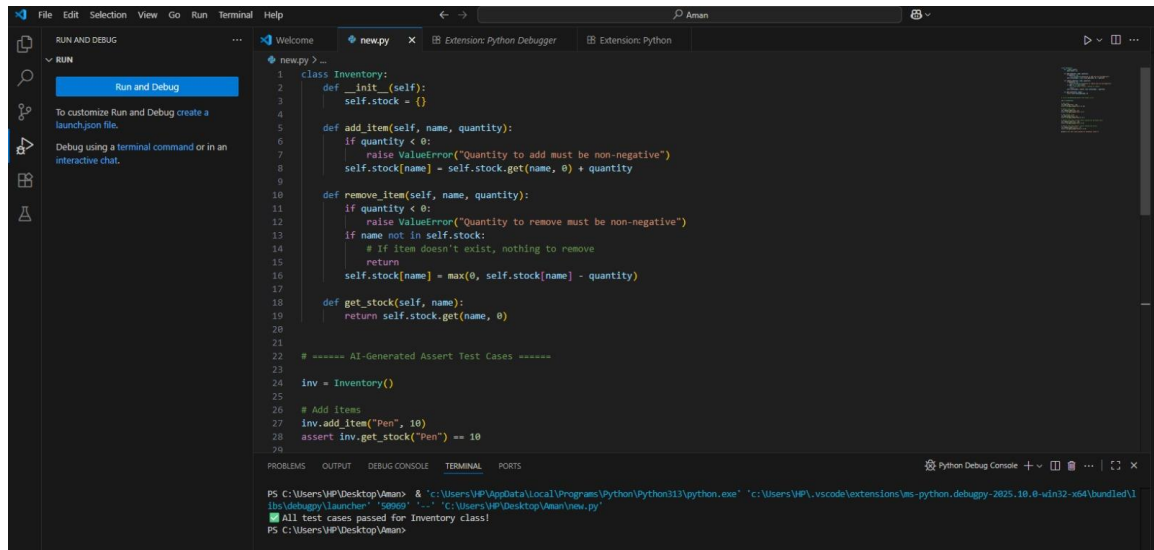
- o `add_item(name, quantity)`
- o `remove_item(name, quantity)`
- o `get_stock(name)`

**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



```
1 class Inventory:
2     def __init__(self):
3         self.stock = {}
4
5     def add_item(self, name, quantity):
6         if quantity < 0:
7             raise ValueError("Quantity to add must be non-negative")
8         self.stock[name] = self.stock.get(name, 0) + quantity
9
10    def remove_item(self, name, quantity):
11        if quantity < 0:
12            raise ValueError("Quantity to remove must be non-negative")
13        if name not in self.stock:
14            # If item doesn't exist, nothing to remove
15            return
16        self.stock[name] = max(0, self.stock[name] - quantity)
17
18    def get_stock(self, name):
19        return self.stock.get(name, 0)
20
21
22    # ===== AI-Generated Assert Test Cases =====
23
24    inv = Inventory()
25
26    # Add Items
27    inv.add_item("Pen", 10)
28    assert inv.get_stock("Pen") == 10
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
PS C:\Users\VP\Desktop\Aman> & "c:\Users\VP\AppData\Local\Programs\Python\Python313\python.exe" "c:\Users\VP\.vscode\extensions\ms-python.debugpy-2025.10.0-sln32-x64\bundle1\ins\debugpy_launcher" "newpy" - "c:\Users\VP\Desktop\Aman\new.py"
All test cases passed for Inventory class!
PS C:\Users\VP\Desktop\Aman>
```

## 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"
```

Expected Output #5:

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

```

1 from datetime import datetime
2
3 def validate_and_format_date(date_str):
4     """
5     Validates the date string in MM/DD/YYYY format.
6     Returns date in YYYY-MM-DD if valid, otherwise "Invalid Date".
7     """
8     try:
9         # Parse date using specified format
10        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
11        # Format date to YYYY-MM-DD and return
12        return date_obj.strftime("%Y-%m-%d")
13    except ValueError:
14        # Raised if date_str is invalid format or invalid date
15        return "Invalid Date"
16
17 # ===== AI-Generated Assert Test Cases =====
18
19 assert validate_and_format_date("10/15/2023") == "2023-10-15" # Valid date
20 assert validate_and_format_date("02/20/2023") == "Invalid Date" # Invalid day in February
21 assert validate_and_format_date("01/01/2024") == "2024-01-01" # Valid date (leap year check irrelevant here)
22 assert validate_and_format_date("13/01/2023") == "Invalid Date" # Invalid month (13)
23 assert validate_and_format_date("04/31/2023") == "Invalid Date" # April 31 does not exist
24 assert validate_and_format_date("abc") == "Invalid Date" # Completely invalid string
25
26 print("✅ All test cases passed for validate_and_format_date!")
27

```

Python Debug Console

```

PS C:\Users\VP\Desktop\Aman> & "c:\Users\VP\AppData\Local\Programs\Python\Python313\python.exe" "c:\Users\VP\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundle1\
StdDebugpy\launcher" -s8000 -p "c:\Users\VP\Desktop\Aman\new.py"
✅ All test cases passed for validate_and_format_date!
PS C:\Users\VP\Desktop\Aman>

```

## Observation:

### **Task 1 – Password Strength Validator**

The function successfully validated password strength using rules for length, uppercase, lowercase, digits, special characters, and no spaces. All test cases passed.

### **Task 2 – Number Classification**

The function correctly classified numbers as Positive, Negative, or Zero, and handled invalid inputs like strings and None. Boundary conditions (-1, 0, 1) worked as expected.

### **Task 3 – Anagram Checker**

The function correctly identified anagrams while ignoring spaces, case, and punctuation. Edge cases like empty strings and identical words were handled properly.

#### **Task 4 – Inventory Class**

The inventory system supported adding, removing, and checking stock. It also handled invalid quantities, missing items, and insufficient stock. All assertions passed.

#### **Task 5 – Date Validation & Formatting**

The function validated dates in MM/DD/YYYY format and converted them to YYYY-MM-DD. Invalid dates (like Feb 30, month=0, wrong day count) were rejected successfully.