

## 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:**

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows the Explorer view with a folder named 'AI - ASSIST' containing several Python files: Assignment-5.4.py, Assignment-1.py, Assignment-2.py, Assignment-4.4.py, Assignment-6.4.py, Assignment-7.1.py, and Assignment-8.1.py. The main editor area displays a Python script named 'Assignment-8.1.py'. The code defines a function 'is\_strong\_password' that checks if a password is strong based on length, uppercase letters, lowercase letters, digits, and special characters. It also includes assert test cases for various password types. The bottom status bar shows the terminal output: 'PS C:\Users\arjun\OneDrive\Desktop\AI - Assist> & C:/Python313/python.exe "c:/Users/arjun/OneDrive/Desktop/AI - Assist/Assignment-8.1.py"' followed by 'True'. The bottom navigation bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, GITLENS, and SPELL CHECKER.

```
4 # Must include uppercase, lowercase, digit, and special character.
5 # Must not contain spaces.
6 import re
7 def is_strong_password(password):
8     if len(password) < 8:
9         return False
10    if not re.search('[A-Z]', password):
11        return False
12    if not re.search('[a-z]', password):
13        return False
14    if not re.search('[0-9]', password):
15        return False
16    if not re.search('[\$!%*?&]', password):
17        return False
18    if ' ' in password:
19        return False
20    return True
21 #Assert test cases
22 assert is_strong_password("StrongPass1!") == True # Valid strong password
23 assert is_strong_password("weakpass") == False # No uppercase, digit, or
24 assert is_strong_password("short1!") == False # Less than 8 characters
25 assert is_strong_password("NoSpecialChar!") == False # No special character
26 assert is_strong_password("Has Space1!") == False # Contains space
27 print(is_strong_password("StrongPass1!"))
```

## 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.

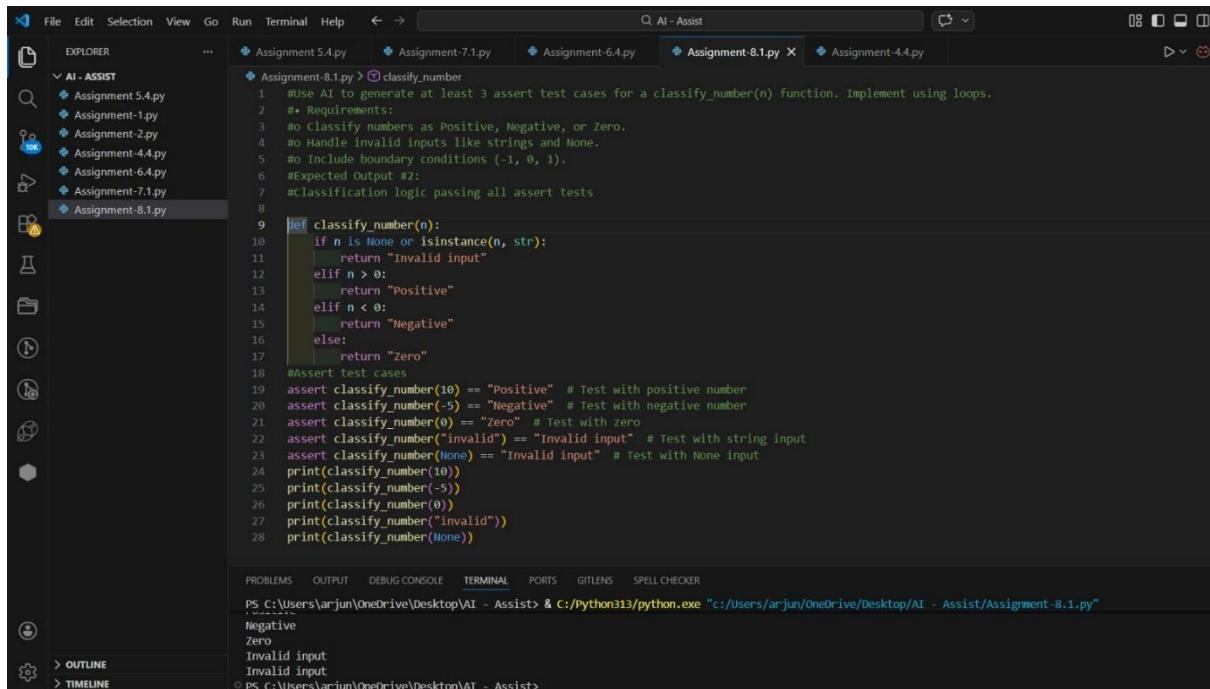
## Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)

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:



The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows files including Assignment 5.4.py, Assignment-1.py, Assignment-2.py, Assignment-4.4.py, Assignment-6.4.py, Assignment-7.1.py, Assignment-8.1.py, and Assignment-4.4.py.
- Code Editor:** Displays Python code for a `classify_number` function. The code handles various input types and values, including None, strings, and integers, and asserts the correctness of the classification logic.
- Terminal:** Shows command-line output from running the script, displaying results for positive, negative, zero, and invalid inputs.
- Status Bar:** Shows the current file path as "Assignment-8.1.py" and the command "PS C:\Users\arjun\OneDrive\Desktop\AI - Assist & C:/Python313/python.exe".

```
#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).
#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))
```

## 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.

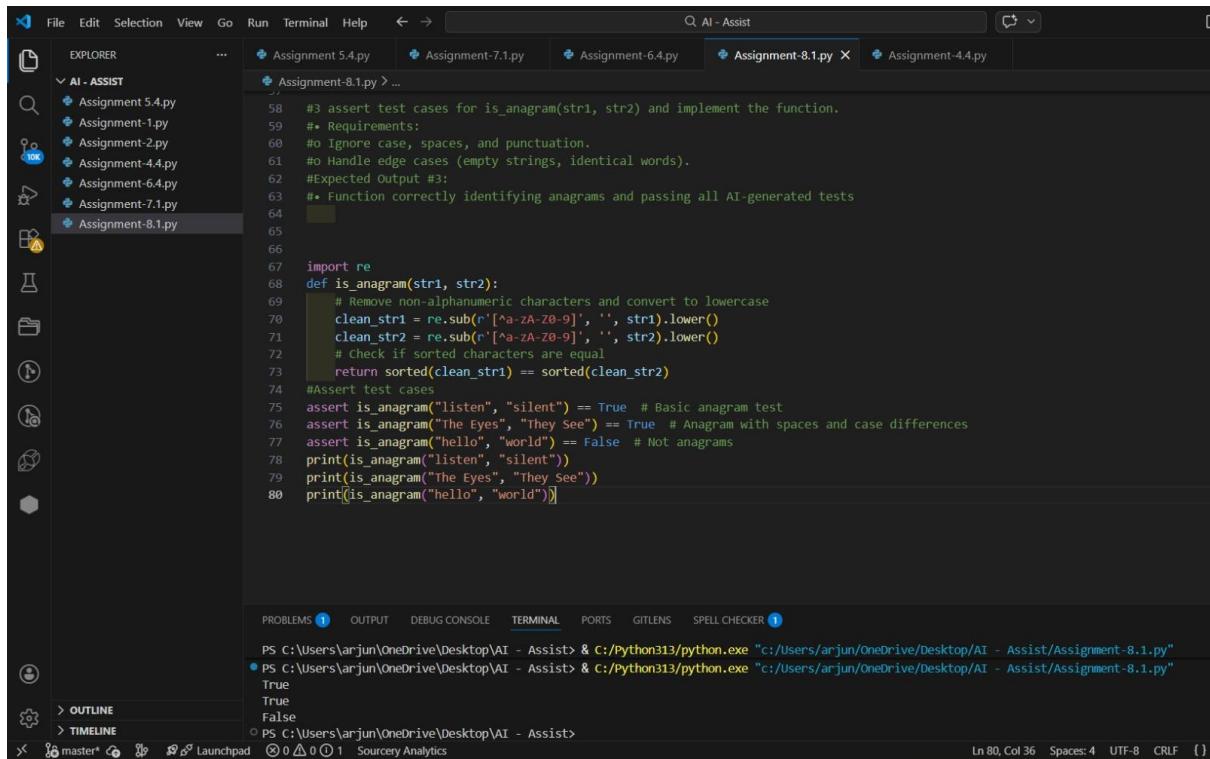
# Task Description #3 (Anagram Checker – Apply AI for String Analysis)

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:



The screenshot shows a code editor interface with a dark theme. The left sidebar has icons for file operations like Open, Save, Find, and Refresh. The Explorer panel shows several Python files: Assignment 5.4.py, Assignment-1.py, Assignment-2.py, Assignment-4.4.py, Assignment-6.4.py, Assignment-7.1.py, and Assignment-8.1.py. The main editor area contains the following Python code:

```
58 # 3 assert test cases for is_anagram(str1, str2) and implement the function.
59 #+ Requirements:
60 #+ Ignore case, spaces, and punctuation.
61 #+ Handle edge cases (empty strings, identical words).
62 #Expected Output #3:
63 #+ Function correctly identifying anagrams and passing all AI-generated tests
64
65
66
67 import re
68 def is_anagram(str1, str2):
69     # Remove non-alphanumeric characters and convert to lowercase
70     clean_str1 = re.sub(r'[^a-zA-Z0-9]', '', str1).lower()
71     clean_str2 = re.sub(r'[^a-zA-Z0-9]', '', str2).lower()
72     # Check if sorted characters are equal
73     return sorted(clean_str1) == sorted(clean_str2)
74 #Assert test cases
75 assert is_anagram("listen", "silent") == True # Basic anagram test
76 assert is_anagram("The Eyes", "They See") == True # Anagram with spaces and case differences
77 assert is_anagram("hello", "world") == False # Not anagrams
78 print(is_anagram("listen", "silent"))
79 print(is_anagram("The Eyes", "They See"))
80 print(is_anagram("hello", "world"))
```

The bottom status bar shows the terminal output:

```
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
```

Other tabs in the bottom bar include PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, GITLENS, and SPELL CHECKER.

## 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:**

```

File Edit Selection View Go Run Terminal Help < > Q AI - Assist
EXPLORER ... Assignment-5.4.py Assignment-7.1.py Assignment-6.4.py Assignment-8.1.py X Assignment-4.4.py
AI - ASSIST Assignment-8.1.py > ...
82 Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.
83 ## Methods:
84 # add_item(name, quantity)
85 #o remove_item(name, quantity)
86 #o get_stock(name)
87 #Expected Output #4:
88 ## Fully functional class passing all assertions.
89
90 class Inventory:
91     def __init__(self):
92         self.stock = {}
93     def add_item(self, name, quantity):
94         if name in self.stock:
95             self.stock[name] += quantity
96         else:
97             self.stock[name] = quantity
98     def remove_item(self, name, quantity):
99         if name in self.stock and self.stock[name] >= quantity:
100             self.stock[name] -= quantity
101             return True
102         return False
103     def get_stock(self, name):
104         return self.stock.get(name, 0)
105 #Assert test cases
106 inventory = Inventory()
107 inventory.add_item("apple", 10)
108 assert inventory.get_stock("apple") == 10 # Test adding items
109 assert inventory.remove_item("apple", 5) == True # Test removing items
110 assert inventory.get_stock("apple") == 5 # Test stock after removal
111 assert inventory.remove_item("apple", 10) == False # Test removing more than stock
112 assert inventory.get_stock("apple") == 5 # Stock should remain unchanged
113 print(inventory.get_stock("apple"))
114 print(inventory.remove_item("apple", 5))
115 print(inventory.get_stock("apple"))
116 print(inventory.remove_item("apple", 10))
117 print(inventory.get_stock("apple"))
118 print(inventory.get_stock("banana")) # Test getting stock for non-existent item
119 print(inventory.remove_item("banana", 1)) # Test removing non-existent item
120 print(inventory.get_stock("banana"))
121 # Should return 0 for non-existent item

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SPELL CHECKER

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

File Edit Selection View Go Run Terminal Help < > Q AI - Assist

EXPLORER ... Assignment-5.4.py Assignment-7.1.py Assignment-6.4.py Assignment-8.1.py X Assignment-4.4.py

AI - ASSIST Assignment-8.1.py > ...

```

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
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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SPELL CHECKER

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

< OUTLINE > TIMELINE

## 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:**

```

File Edit Selection View Go Run Terminal Help ⏮ ⏯ Q AI - Assist
EXPLORER ... Assignment-5.4.py Assignment-7.1.py Assignment-6.4.py Assignment-8.1.py Assignment-4.4.py Assignment-1.py Assignment-2.py Assignment-3.py Assignment-4.py Assignment-5.py Assignment-6.py Assignment-7.py Assignment-8.py
AI - ASSIST
Assignment-8.1.py > ...
124 #Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.
125 # Requirements:
126 # Validate "MM/DD/YYYY" format.
127 # Handle invalid dates.
128 # Convert valid dates to "YYYY-MM-DD".
129 #Expected Output #5:
130 #Function passes all AI-generated assertions and handles edge cases
131
132 import re
133 def validate_and_format_date(date_str):
134     # Validate date format
135     if not re.match(r'^\d{2}/\d{2}/\d{4}$', date_str):
136         return "Invalid date format"
137     month, day, year = map(int, date_str.split('/'))
138     # Validate date values
139     if month < 1 or month > 12 or day < 1 or day > 31:
140         return "Invalid date"
141     # Handle February and leap years
142     if month == 2:
143         if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
144             if day > 29:
145                 return "Invalid date"
146             else:
147                 if day > 28:
148                     return "Invalid date"
149         # Handle months with 30 days
150         if month in [4, 6, 9, 11] and day > 30:
151             return "Invalid date"
152         # Convert to "YYYY-MM-DD" format
153         return f"(year:04d)-(month:02d)-(day:02d)"
154     #Assert test cases
155     assert validate_and_format_date("12/25/2020") == "2020-12-25" # Valid date
156     assert validate_and_format_date("02/29/2020") == "2020-02-29" # Valid leap year date
157     assert validate_and_format_date("02/30/2020") == "Invalid date" # Invalid date
158     assert validate_and_format_date("13/01/2020") == "Invalid date" # Invalid month
159     assert validate_and_format_date("12/31/2020") == "2020-12-31" # Valid date
160     assert validate_and_format_date("invalid") == "Invalid date format" # Invalid format

```

```

File Edit Selection View Go Run Terminal Help ⏮ ⏯ Q AI - Assist
EXPLORER ... Assignment-5.4.py Assignment-7.1.py Assignment-6.4.py Assignment-8.1.py Assignment-4.4.py Assignment-1.py Assignment-2.py Assignment-3.py Assignment-4.py Assignment-5.py Assignment-6.py Assignment-7.py Assignment-8.py
AI - ASSIST
Assignment-8.1.py > ...
133 def validate_and_format_date(date_str):
134     if not re.match(r'^\d{2}/\d{2}/\d{4}$', date_str):
135         return "Invalid date"
136     else:
137         if day > 28:
138             return "Invalid date"
139         # Handle months with 30 days
140         if month in [4, 6, 9, 11] and day > 30:
141             return "Invalid date"
142         # Convert to "YYYY-MM-DD" format
143         return f"(year:04d)-(month:02d)-(day:02d)"
144     #Assert test cases
145     assert validate_and_format_date("12/25/2020") == "2020-12-25" # Valid date
146     assert validate_and_format_date("02/29/2020") == "2020-02-29" # Valid leap year date
147     assert validate_and_format_date("02/30/2020") == "Invalid date" # Invalid date
148     assert validate_and_format_date("13/01/2020") == "Invalid date" # Invalid month
149     assert validate_and_format_date("12/31/2020") == "2020-12-31" # Valid date
150     assert validate_and_format_date("invalid") == "Invalid date format" # Invalid format
151     print(validate_and_format_date("12/25/2020"))
152     print(validate_and_format_date("02/29/2020"))
153     print(validate_and_format_date("02/30/2020"))
154     print(validate_and_format_date("13/01/2020"))
155     print(validate_and_format_date("12/31/2020"))
156     print(validate_and_format_date("invalid"))

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SPELL CHECKER 1
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.