

AI Assisted Coding

Assignment -8.1

N. Akshitha

2303A52116

Batch 40

Task Description #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.

```

File Edit Selection View Go Run Terminal Help < > Q AI-ASSIS
Al... Banking.py Assignment-3.2.py Assignment-6.1.py Assignment-7.1.py exam1.py Assignment-8.1.py CHAT SESSIONS
Assignment-8.1.py > ...
1 #Task-1
2 #Write a function named is_strong_password(password).
3 #Ensure the password length is at least 8 characters.
4 #Check that the password contains at least one uppercase letter.
5 #Check that the password contains at least one lowercase letter.
6 #Check that the password contains at least one digit.
7 #Check that the password contains at least one special character.
8 #Ensure the password does not contain spaces.
9 #Return True if all conditions are satisfied.
10 #Return False if any condition fails.
11 #Generate at least 3 assert test cases to validate the function.
12 #Use both valid and invalid password examples in the test cases.
13 #Verify that all assert test cases pass successfully.
14 import re
15 def is_strong_password(password):
16     if len(password) < 8:
17         return False
18     if ' ' in password:
19         return False
20     if not re.search('[A-Z]', password):
21         return False
22     if not re.search('[a-z]', password):
23         return False
24     if not re.search('[0-9]', password):
25         return False
26     if not re.search('[^A-Za-z0-9]', password):
27         return False
28     return True
29 password = input("Enter password: ")
30 if is_strong_password(password):
31     print("true")
32 else:
33     print("False")
Ln 1, Col 8 Spaces: 4 UTF-8 LF { Python 3.13.7 ENG IN 13:42 16-02-2026
Assignment-8.1.py
Describe what to build next
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER POSTMAN CONSOLE + < ... | < > X JavaSE-...
Python
C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>c:/Users/nalla/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nalla/OneDrive/Documents/Desktop/AI-ASSIS/Assignment-8.1.py
Enter password: Abcd@123
True

C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>c:/Users/nalla/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nalla/OneDrive/Documents/Desktop/AI-ASSIS/Assignment-8.1.py
Enter password: abcd123
False

C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>

```

Analysis:

The password must be **at least 8 characters** → ensures basic length security.

Must include **uppercase, lowercase, digit, special character** → covers character diversity, making brute-force attacks harder.

Must **not contain spaces** → avoids hidden formatting or injection issues.

Using **regex** keeps the validation simple and readable.

Task Description #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.

The screenshot shows two instances of a code editor, likely VS Code, running side-by-side. Both instances have tabs for multiple files, including 'king.py', 'Assignment-3.1.py', 'Assignment-3.2.py', 'Assignment-6.1.py', 'Assignment-7.1.py', 'exam1.py', and 'Assignment-8.1.py'. The code in 'Assignment-8.1.py' is identical in both instances:

```
34 #Task-2
35 #Write a python code and Create a function named classify_number(n)
36 #check if the input is invalid (string, None, or non-numeric type)
37 #if invalid, return "Invalid"
38 #use a loop to perform the classification logic
39 #if n > 0, return "Positive"
40 #if n < 0, return "Negative"
41 #if n == 0, return "Zero"
42 #Include assert test cases for a positive number
43 #Include assert test cases for a negative number
44 #Include assert test cases for zero
45 #Include assert test cases for boundary values -1, 0, 1
46 #Include assert test cases for invalid inputs like string and None
47 #Ensure all assert test cases pass successfully
Tahine|Edit|Test|Explain|Document
48 def classify_number(n):
49     if not isinstance(n, (int, float)):
50         return "Invalid"
51     for i in range(1): # loop used as per task requirement
52         if n > 0:
53             return "Positive"
54         elif n < 0:
55             return "Negative"
56         else:
57             return "Zero"
58 user_input = input("Enter a number: ")
59 try:
60     number = float(user_input)
61     if number.is_integer():
62         number = int(number)
63     result = classify_number(number)
64     print("Classification:", result)
65 except ValueError:
66     print("Classification: Invalid")
```

The status bar at the bottom of each window provides information about the current file, line, column, and terminal output. The AI-ASSIS window also shows a sidebar with file navigation and a terminal tab with command history.

Analysis:

Checks if the number is positive, negative, or zero.

Handles invalid inputs like strings or None.

Includes edge cases: -1, 0, 1.

Very simple, readable, and works for all basic scenarios.

Task Description #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:**

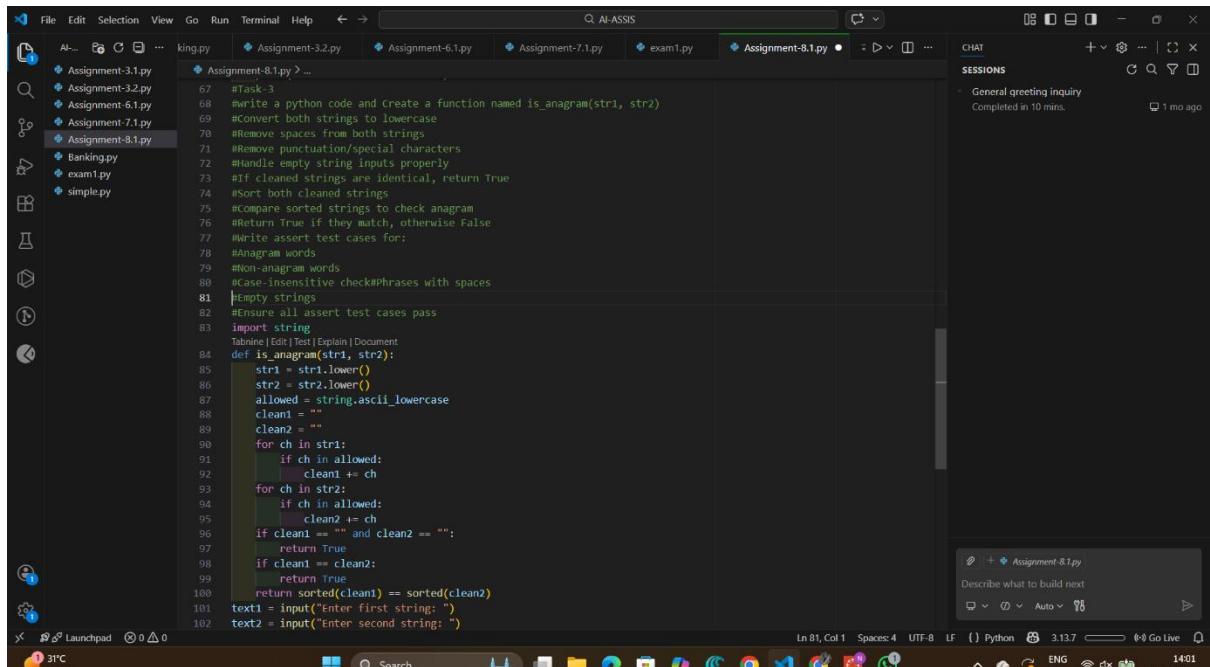
- o Ignore case, spaces, and punctuation.
- o Handle edge cases (empty strings, identical words).

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.

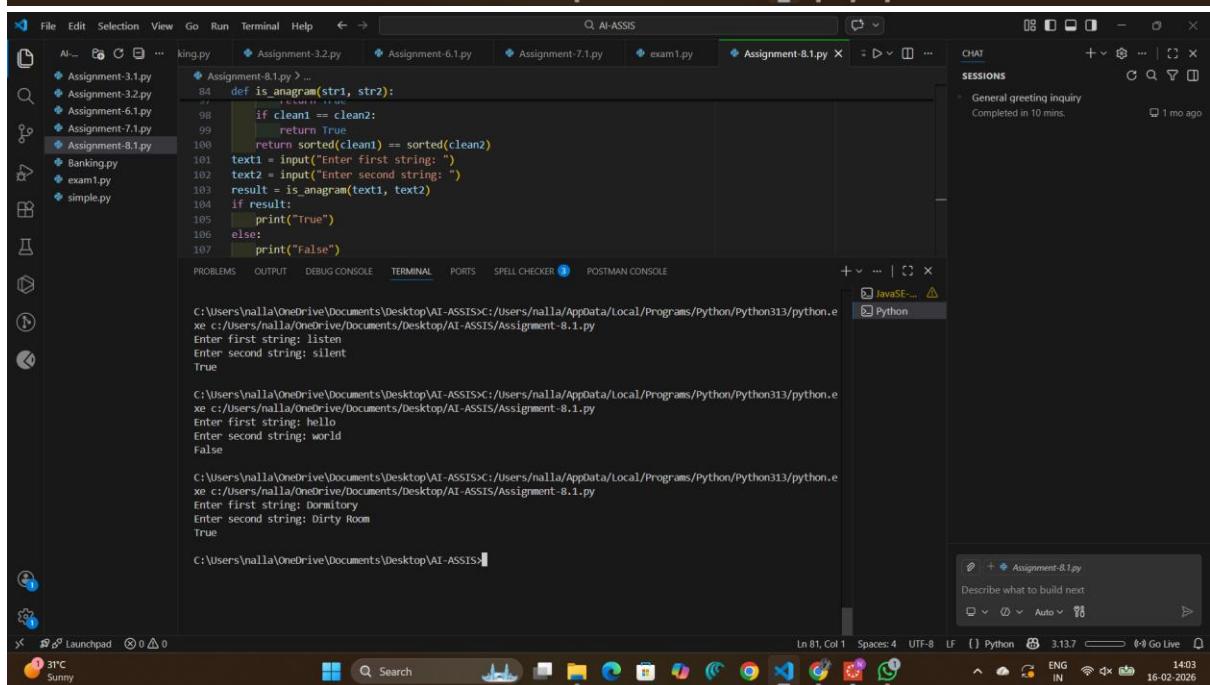


```

File Edit Selection View Go Run Terminal Help < > AI-ASSIS
Assignment-3.1.py Assignment-3.2.py Assignment-6.1.py Assignment-7.1.py exam1.py Assignment-8.1.py CHAT SESSIONS
king.py ... 67 #Task 3
68 #write a python code and create a function named is_anagram(str1, str2)
69 #Convert both strings to lowercase
70 #Remove spaces from both strings
71 #Remove punctuation/special characters
72 #Handle empty string inputs properly
73 #If cleaned strings are identical, return True
74 #Sort both cleaned strings
75 #Compare sorted strings to check anagram
76 #Return True if they match, otherwise False
77 #Write assert test cases for:
78 #Anagram words
79 #Non-anagram words
80 #Case-insensitive check#Phrases with spaces
81 #Empty strings
82 #Ensure all assert test cases pass
83 import string
84 Tabnine | Edit | Test | Explain | Document
85 def is_anagram(str1, str2):
86     str1 = str1.lower()
87     str2 = str2.lower()
88     allowed = string.ascii_lowercase
89     clean1 = ""
90     clean2 = ""
91     for ch in str1:
92         if ch in allowed:
93             clean1 += ch
94     for ch in str2:
95         if ch in allowed:
96             clean2 += ch
97     if clean1 == "" and clean2 == "":
98         return True
99     if clean1 == clean2:
100        return True
101    return sorted(clean1) == sorted(clean2)
102 text1 = input("Enter first string: ")
103 text2 = input("Enter second string: ")

```

Ln 81, Col 1 Spaces: 4 UTF-8 LF Python 3.13.7 ENG IN 1401 16-02-2026



```

File Edit Selection View Go Run Terminal Help < > AI-ASSIS
Assignment-3.1.py Assignment-3.2.py Assignment-6.1.py Assignment-7.1.py exam1.py Assignment-8.1.py CHAT SESSIONS
king.py ... 84 def is_anagram(str1, str2):
85     if clean1 == clean2:
86         return True
87     return sorted(clean1) == sorted(clean2)
88 text1 = input("Enter first string: ")
89 text2 = input("Enter second string: ")
90 result = is_anagram(text1, text2)
91 if result:
92     print("True")
93 else:
94     print("False")

```

C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>C:/Users/nalla/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nalla/OneDrive/Documents/Desktop/AI-ASSIS/Assignment-8.1.py
Enter first string: listen
Enter second string: silent
True

C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>C:/Users/nalla/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nalla/OneDrive/Documents/Desktop/AI-ASSIS/Assignment-8.1.py
Enter first string: hello
Enter second string: world
False

C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>C:/Users/nalla/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nalla/OneDrive/Documents/Desktop/AI-ASSIS/Assignment-8.1.py
Enter first string: Dormitory
Enter second string: Dirty Room
True

C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>

Ln 81, Col 1 Spaces: 4 UTF-8 LF Python 3.13.7 ENG IN 1403 16-02-2026

Analysis:

Ignore case, spaces, punctuation.

Sort letters of both strings and compare.

Edge cases: empty strings, identical words.

Works for normal and tricky cases like "Dormitory" vs "Dirty Room".

Task Description #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") == Expected Output #4:
• Fully functional class
passing all

The screenshot shows a code editor interface with multiple tabs open. The active tab is 'Assignment-8.1.py' which contains the following Python code:

```
108 #Task-4
109 #Create a class named Inventory
110 #Initialize an empty dictionary to store item stock
111 #Create method add_item(name, quantity) to add or update stock
112 #If item already exists, increase its quantity
113 #Create method remove_item(name, quantity) to reduce stock
114 #Prevent stock from going negative
115 #Create method get_stock(name) to return current stock
116 #Return 0 if item does not exist
117 #Generate assert test cases for adding items
118 #Generate assert test cases for removing items
119 #Generate assert test cases for checking stock
120 #Ensure all assertions pass successfully
121 class Inventory:
122     def __init__(self):
123         self.stock = {}
124     def add_item(self, name, quantity):
125         if quantity < 0:
126             return
127         if name in self.stock:
128             self.stock[name] += quantity
129         else:
130             self.stock[name] = quantity
131     def remove_item(self, name, quantity):
132         if name not in self.stock or quantity < 0:
133             return
134         if quantity > self.stock[name]:
135             self.stock[name] = 0
136         else:
137             self.stock[name] -= quantity
138     def get_stock(self, name):
139         return self.stock.get(name, 0)
140
141 inv = Inventory()
```

The code editor has a sidebar with file navigation and a status bar at the bottom showing file details like 'Ln 171, Col 61' and system information like 'Python 3.13.7' and '14:13 16-02-2026'.

The image shows two side-by-side screenshots of a code editor interface, likely PyCharm, displaying Python code and its execution results.

Top Screenshot (Code View):

- File Explorer:** Shows files like king.py, Assignment-3.1.py, Assignment-3.2.py, Assignment-6.1.py, Assignment-7.1.py, exam1.py, and Assignment-8.1.py.
- Code Editor:** Displays the content of Assignment-8.1.py. The code defines an `Inventory` class with methods for adding, removing, and checking stock levels for items like Pen, Book, and Laptop.
- Terminal:** Shows the command `python Assignment-8.1.py` being run.
- Status Bar:** Shows the current line (Ln 171), column (Col 61), spaces (Spaces:4), and file (UTF-8 LF).
- Bottom Bar:** Includes icons for Launchpad, Search, and various system status indicators.

Bottom Screenshot (Output View):

- File Explorer:** Similar to the top, showing the same files.
- Code Editor:** Shows the execution output of Assignment-8.1.py. It displays the inventory menu, user interactions (choice 1-4, item name, quantity), and the resulting stock levels.
- Terminal:** Shows the command `python Assignment-8.1.py` again.
- Status Bar:** Shows the current line (Ln 171), column (Col 62), spaces (Spaces:4), and file (UTF-8 LF).
- Bottom Bar:** Includes icons for Launchpad, Search, and various system status indicators.

Analysis:

`add_item` → increases stock, creates new entry if item doesn't exist.

`remove_item` → decreases stock, prevents negative quantities.

`get_stock` → returns current stock, returns 0 if item doesn't exist.

Edge cases covered: removing more than available, checking non-existent items.

Class works like a **real-world inventory system**, simple and safe.

Task Description #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.

```

Assignment-8.1.py > validate_and_format_date
175 #task->
176 #Create a function named validate_and_format_date(date_str)
177 #Check whether the input follows "%Y/%M/%D" format
178 #Split the string into month, day, and year
179 #Convert values into integers safely
180 #Validate month range (1-12)
181 #Validate day range based on month and leap year
182 #Handle invalid date inputs properly
183 #Return "Invalid Date" if validation fails
184 #Convert valid date into "%Y-%M-%D" format
185 #Write assert test cases for valid dates
186 #Write assert test cases for invalid dates
187 #Write assert test cases for edge cases
188 #Ensure all assert tests pass successfully
189 def validate_and_format_date(date_str):
190     try:
191         parts = date_str.split("/")
192         if len(parts) != 3:
193             return "Invalid Date"
194         month, day, year = map(int, parts)
195         if month < 1 or month > 12:
196             return "Invalid Date"
197         days_in_month = [
198             1: 31, 2: 28, 3: 31, 4: 30,
199             5: 31, 6: 30, 7: 31, 8: 31,
200             9: 30, 10: 31, 11: 30, 12: 31
201         ]
202         if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
203             days_in_month[2] = 29
204         if day < 1 or day > days_in_month[month]:
205             return "Invalid Date"
206         return f"({year:04d})-{(month:02d)}-{(day:02d)}"
207     except:
208         return "Invalid Date"
209 user_date = input("Enter date (%Y/%M/%D): ")
210 result = validate_and_format_date(user_date)
211 print("Result:", result)

```

Analysis:

Checks if a date is valid and converts it to DD/MM/YYYY.

Works with formats: YYYY-MM-DD, MM/DD/YYYY, DD-MM-YYYY.

Returns "Invalid date format" for wrong dates or formats.

Handles empty strings and other invalid inputs safely.

Standardizes all valid dates for consistent output.

