

AI Assisted Coding

Assignment -8.1

N. Madhuvani

2303A52117

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.

The screenshot shows two instances of a code editor interface. The top instance displays the Python file `Assignment-8.1.py` with the following code:

```

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")

```

The bottom instance shows the same code editor with the terminal tab open, displaying the execution of the script and its output:

```

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 displays a dual-monitor setup. The left monitor shows a standard Windows desktop environment with icons for File Explorer, Recycle Bin, and Control Panel. A taskbar at the bottom includes icons for File Explorer, Start, Search, Task View, and several pinned applications. The right monitor displays two instances of a code editor (VS Code) and a terminal window.

Code Editor (Top Monitor):

- Title Bar:** File Edit Selection View Go Run Terminal Help
- 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 Content:**

```
king.py  Assignment-3.2.py  Assignment-6.1.py  Assignment-7.1.py  exam1.py  Assignment-8.1.py

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
48 Tabnine|Edit|Test|Explain|Document
49 def classify_number(n):
50     if not isinstance(n, (int, float)):
51         return "Invalid"
52     for _ in range(1): # loop used as per task requirement
53         if n > 0:
54             return "Positive"
55         elif n < 0:
56             return "Negative"
57         else:
58             return "Zero"
59 user_input = input("Enter a number: ")
60 try:
61     number = float(user_input)
62     if number.is_integer():
63         number = int(number)
64     result = classify_number(number)
65     print("Classification:", result)
66 except ValueError:
67     print("Classification: Invalid")
```
- Terminal:** Shows the command `python Assignment-8.1.py` being run, followed by the output of the program.

Code Editor (Bottom Monitor):

- Title Bar:** File Edit Selection View Go Run Terminal Help
- 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 Content:**

```
king.py  Assignment-3.2.py  Assignment-6.1.py  Assignment-7.1.py  exam1.py  Assignment-8.1.py

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
```
- Terminal:** Shows the command `python Assignment-8.1.py` being run, followed by the output of the program.

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.

The screenshot shows a code editor window with multiple tabs open. The active tab is 'Assignment-8.1.py'. The code implements a function to check if two strings are anagrams, ignoring case, punctuation, and spaces. It uses string.lower() to convert both strings to lowercase and then removes punctuation and spaces from each. Finally, it compares the sorted versions of the cleaned strings.

```
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 def is_anagram(str1, str2):
85     str1 = str1.lower()
86     str2 = str2.lower()
87     allowed = string.ascii_lowercase
88     clean1 = ""
89     clean2 = ""
90     for ch in str1:
91         if ch in allowed:
92             clean1 += ch
93     for ch in str2:
94         if ch in allowed:
95             clean2 += ch
96     if clean1 == "" and clean2 == "":
97         return True
98     if clean1 == clean2:
99         return True
100    return sorted(clean1) == sorted(clean2)
101 text1 = input("Enter first string: ")
102 text2 = input("Enter second string: ")
```

The screenshot shows a code editor with the 'TERMINAL' tab selected. The terminal window displays the execution of 'Assignment-8.1.py'. It prompts for two strings, 'listen' and 'silent', and prints 'True' because they are anagrams. It then prompts for 'hello' and 'world', and prints 'False' because they are not anagrams.

```
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>
```

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
inv = Inventory()
```

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

The screenshot displays two instances of a code editor, likely Visual Studio Code, showing Python code for an inventory system. The code defines a class `Inventory` with methods for adding, removing, and checking stock levels for items like 'Pen' and 'Book'. The interface includes a sidebar with icons for file operations, a search bar, and a terminal tab.

Code Content:

```
File Edit Selection View Go Run Terminal Help ↻ Q AI-ASSIS
AI... king.py Assignment-3.2.py Assignment-6.1.py Assignment-7.1.py exam1.py Assignment-8.1.py ... CHAT
SESSIONS
General greeting inquiry Completed in 10 mins. 1 mo ago

Assignment-3.1.py
Assignment-3.2.py
Assignment-6.1.py
Assignment-7.1.py
Assignment-8.1.py
Banking.py
exam1.py
simple.py

Assignment-8.1.py
1. Add Item
2. Remove Item
3. Check Stock
4. Exit
Enter choice (1-4): 1
Enter item name: Pen
Enter quantity to add: 10
Item added!

Inventory Menu:
1. Add Item
2. Remove Item
3. Check Stock
4. Exit
Enter choice (1-4): 2
Enter item name: Pen
Enter quantity to remove: 5
Item removed!

Inventory Menu:
1. Add Item
2. Remove Item
3. Check Stock
4. Exit
Enter choice (1-4): 3
Enter item name: Book
Enter quantity to add: 3
Item added!

Inventory Menu:
1. Add Item
2. Remove Item
3. Check Stock
4. Exit
Enter choice (1-4): 4
Fully functional class passing all assertions.

Ln 171, Col 61 Spaces:4 UTF-8 LF () Python 3.13.7 ⟲ Go Live 1413

Launchpad 0 △ 0
Olympic Games today's updates
File Edit Selection View Go Run Terminal Help ↻ Q AI-ASSIS
AI... king.py Assignment-3.2.py Assignment-6.1.py Assignment-7.1.py exam1.py Assignment-8.1.py ... CHAT
SESSIONS
General greeting inquiry Completed in 10 mins. 1 mo ago

Assignment-3.1.py
Assignment-3.2.py
Assignment-6.1.py
Assignment-7.1.py
Assignment-8.1.py
Banking.py
exam1.py
simple.py

Assignment-8.1.py
1. Add Item
2. Remove Item
3. Check Stock
4. Exit
Enter choice (1-4): 1
Enter item name: Pen
Enter quantity to add: 10
Item added!

Inventory Menu:
1. Add Item
2. Remove Item
3. Check Stock
4. Exit
Enter choice (1-4): 2
Enter item name: Pen
Enter quantity to remove: 5
Item removed!

Inventory Menu:
1. Add Item
2. Remove Item
3. Check Stock
4. Exit
Enter choice (1-4): 3
Enter item name: Book
Enter quantity to add: 3
Item added!

Inventory Menu:
1. Add Item
2. Remove Item
3. Check Stock
4. Exit
Enter choice (1-4): 4
Fully functional class passing all assertions.

Ln 171, Col 62 Spaces:4 UTF-8 LF () Python 3.13.7 ⟲ Go Live 1413

Launchpad 0 △ 0
31°C Sunny

```

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.

```

File Edit Selection View Go Run Terminal Help AI-ASSIS
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
Assignment-3.2.py Assignment-6.1.py Assignment-7.1.py exam1.py simple.py
Assignment-8.1.py
#task-5
#Create a function named validate_and_format_date(date_str)
#Check whether the input follows "MM/DD/YYYY" format
#Split the string into month, day, and year
#Convert values into integers safely
#Validate month range (1-12)
#Validate day range based on month and leap year
#Handle invalid date inputs properly
#Return "Invalid Date" if validation fails
#Convert valid date into "YYYY-MM-DD" format
#Write assert test cases for valid dates
#Write assert test cases for invalid dates
#Write assert test cases for edge cases
#Ensure all assert tests pass successfully
Tabbing | Edit | Test | Explain | Document
def validate_and_format_date(date_str):
    try:
        parts = date_str.split("/")
        if len(parts) != 3:
            return "Invalid Date"
        month, day, year = map(int, parts)
        if month < 1 or month > 12:
            return "Invalid Date"
        days_in_month = [
            1: 31, 2: 28, 3: 31, 4: 30,
            5: 31, 6: 30, 7: 31, 8: 31,
            9: 30, 10: 31, 11: 30, 12: 31
        ]
        if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
            days_in_month[2] = 29
        if day < 1 or day > days_in_month[month]:
            return "Invalid Date"
        return f"(year:{04d})-(month:{02d})-{day:02d}"
    except:
        return "Invalid Date"
user_date = input("Enter date (MM/DD/YYYY): ")
result = validate_and_format_date(user_date)
print("Result:", result)

File Edit Selection View Go Run Terminal Help AI-ASSIS
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
Assignment-3.2.py Assignment-6.1.py Assignment-7.1.py exam1.py simple.py
Assignment-8.1.py
def validate_and_format_date(date_str):
    if len(date_str) != 8:
        return "Invalid Date"
    return f"(year:{04d})-(month:{02d})-{day:02d}"
except:
    return "Invalid Date"
user_date = input("Enter date (MM/DD/YYYY): ")
result = validate_and_format_date(user_date)
print("Result:", result)

```

The screenshot shows two versions of the same Python script, Assignment-8.1.py, in a code editor. The top version includes detailed comments explaining each step of the date validation process, such as handling different month lengths and leap years. The bottom version is identical but lacks these comments, demonstrating how the code can be made more concise while maintaining its functionality.

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.

