

# AI ASSISTED CODING

## ASSIGNMENT 8.1

2403A51263-Konda Shiva

Batch-11

**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.

### Prompt:

Task: Implement a password strength validator function

Requirements:

- Password must have at least 8 characters
- Must include uppercase, lowercase, digit, and special character
- Must not contain spaces

Also generate at least 3 assert test cases for `is_strong_password(password)`.

Expected Output:

Password validation logic passing all AI-generated test cases.

## Code and Output:

The screenshot shows a VS Code editor with a Python file named `1task.py`. The script defines a function `is_strong_password` that checks for password strength based on length, character types, and spaces. It also includes test cases to verify the function's behavior.

```

1 import re
2
3 def is_strong_password(password):
4     # Check length
5     if len(password) < 8:
6         return False
7     # Check for spaces
8     if ' ' in password:
9         return False
10    # Check for uppercase
11    if not re.search(r'[A-Z]', password):
12        return False
13    # Check for lowercase
14    if not re.search(r'[a-z]', password):
15        return False
16    # Check for digit
17    if not re.search(r'[0-9]', password):
18        return False
19    # Check for special character
20    if not re.search(r'[!@#$%^&*~`~\|;:,./? ]', password):
21        return False
22    return True
23
24 # Test cases
25 assert is_strong_password("Abcdef1") == True
26 assert is_strong_password("abcdef1") == False # No uppercase
27 assert is_strong_password("Abcdefgh") == False # No digit or special char
28 assert is_strong_password("Abcdef! ") == False # Contains space
29 print("All tests passed!")

```

The terminal at the bottom shows the command to run the script and the output:

```

PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8> & C:\Users\siris\AppData\Local\Microsoft\WindowsApps\python3.
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8> & C:\Users\siris\AppData\Local\Microsoft\WindowsApps\python3.
All tests passed!

```

## 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

Prompt:

### Task: Implement a Number Classification function with Loops

### Requirements:

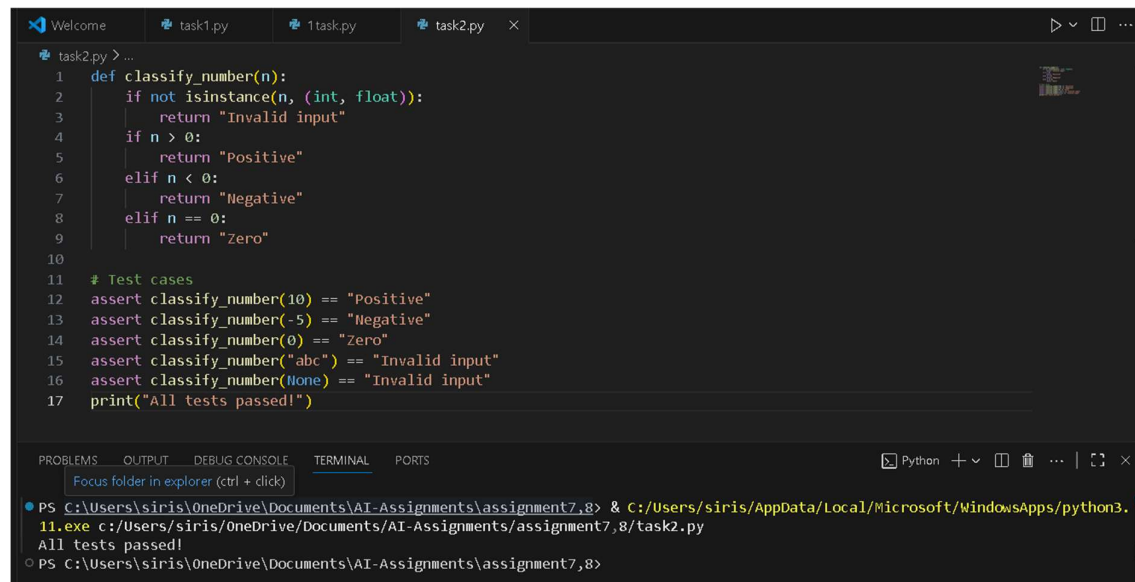
- Classify numbers as Positive, Negative, or Zero.
- Handle invalid inputs like strings and None.

Also generate at least 3 assert test cases for `classify_number(n)`

Expected Output:

Password validation logic passing all AI-generated test cases.

Code and output:



```

1 def classify_number(n):
2     if not isinstance(n, (int, float)):
3         return "Invalid input"
4     if n > 0:
5         return "Positive"
6     elif n < 0:
7         return "Negative"
8     elif n == 0:
9         return "Zero"
10
11 # Test cases
12 assert classify_number(10) == "Positive"
13 assert classify_number(-5) == "Negative"
14 assert classify_number(0) == "Zero"
15 assert classify_number("abc") == "Invalid input"
16 assert classify_number(None) == "Invalid input"
17 print("All tests passed!")

```

Terminal Output:

```

PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8> & C:/Users/siris/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/siris/OneDrive/Documents/AI-Assignments/assignment7,8/task2.py
All tests passed!
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8>

```

### 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-

prompt:

Task: Implement an Anagram Checker function `is_anagram(str1, str2)`

### Requirements:

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

Generate at least 3 assert test cases.

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

## Code and output:

```
task3.py > ...
1 import re
2
3 def is_anagram(str1, str2):
4     # Remove spaces and punctuation, convert to lowercase
5     clean1 = re.sub(r'[\W_]', '', str1).lower()
6     clean2 = re.sub(r'[\W_]', '', str2).lower()
7     # Edge case: both empty strings are anagrams
8     if clean1 == "" and clean2 == "":
9         return True
10    # Compare sorted characters
11    return sorted(clean1) == sorted(clean2)
12
13 # Test cases
14 assert is_anagram("Listen", "Silent") == True
15 assert is_anagram("Astronomer", "Moon starrer") == True
16 assert is_anagram("Hello", "Olelh!") == True
17 assert is_anagram("Test", "Taste") == False
18 assert is_anagram("", "") == True
19 print("All tests passed!")
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8> & C:\Users\siris\AppData\Local\Microsoft\WindowsApps\python3.11.exe c:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8\task3.py
All tests passed!
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8>
```

## 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") == 3
Expected Output #4:
```

- Fully functional class passing all assertions.

## Prompt:

Task: Implement an Inventory class with methods add\_item, remove\_item, get\_stock.

Requirements:

- Manage stock quantities for items
- Generate at least 3 assert test cases

Expected Output: Fully functional class passing all assertions.

## Code and output:

```
task4.py > ...
1  class Inventory:
2      def __init__(self):
3          self.stock = {}
4
5      def add_item(self, item, quantity):
6          if quantity < 0:
7              raise ValueError("Quantity cannot be negative")
8          self.stock[item] = self.stock.get(item, 0) + quantity
9
10     def remove_item(self, item, quantity):
11         if item not in self.stock or self.stock[item] < quantity or quantity < 0:
12             return False
13         self.stock[item] -= quantity
14         if self.stock[item] == 0:
15             del self.stock[item]
16         return True
17
18     def get_stock(self, item):
19         return self.stock.get(item, 0)
20
21     # Test cases
22     inv = Inventory()
23     inv.add_item("apple", 10)
24     assert inv.get_stock("apple") == 10
25
26     inv.add_item("banana", 5)
27     assert inv.get_stock("banana") == 5
28
29     assert inv.remove_item("apple", 3) == True
30     assert inv.get_stock("apple") == 7
31
32     assert inv.remove_item("banana", 10) == False # Not enough stock
33     assert inv.get_stock("banana") == 5
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [ ] [x] [ ] [x]

/WindowsApps/python3.11.exe c:/Users/siris/OneDrive/Documents/AI-Assignments/assignment7,8/task4.py  
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8>

## 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.

Prompt:

### Task: Implement `validate_and_format_date(date_str)`

### Requirements:

- Validate "MM/DD/YYYY" format
- Handle invalid dates
- Convert valid dates to "YYYY-MM-DD"

Generate at least 3 assert test cases

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

## Code and output:

```
task5.py > ...
1 from datetime import datetime
2
3 def validate_and_format_date(date_str):
4     try:
5         # Try to parse the date in MM/DD/YYYY format
6         dt = datetime.strptime(date_str, "%m/%d/%Y")
7         # Return in YYYY-MM-DD format
8         return dt.strftime("%Y-%m-%d")
9     except ValueError:
10        return "Invalid date"
11
12 # Test cases
13 assert validate_and_format_date("12/31/2022") == "2022-12-31"
14 assert validate_and_format_date("02/29/2021") == "Invalid date" # 2021 is not a leap year
15 assert validate_and_format_date("13/01/2022") == "Invalid date" # Invalid month
16 assert validate_and_format_date("04/31/2022") == "Invalid date" # April has only 30 days
17 assert validate_and_format_date("01/01/2020") == "2020-01-01"
```