

# **AI Assisted Coding**

## **Lab Assignment- 8.1**

**Name : G.Abhiram**

**Ht.No : 2303A51087**

**Batch No : 02**

### **1. 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:

    Password must have at least 8 characters.

    Must include uppercase, lowercase, digit, and special character.

    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.

```
#write a function for strong password check and demonstrate it using assert statements with 3 test cases

def is_strong_password(password):
    if len(password) < 8:
        return False
    has_upper = False
    has_lower = False
    has_digit = False
    has_special = False
    special_characters = "!@#$%^&*()-+"

    for char in password:
        if char.isupper():
            has_upper = True
        elif char.islower():
            has_lower = True
        elif char.isdigit():
            has_digit = True
        elif char in special_characters:
            has_special = True

    return has_upper and has_lower and has_digit and has_special
```

```
lab-8.1.py > is_strong_password
1  #write a function for strong password check and demonstrate it using assert statements with 3 test cases
2
3  def is_strong_password(password):
4      if len(password) < 8:
5          return False
6      has_upper = False
7      has_lower = False
8      has_digit = False
9      has_special = False
10     special_characters = "!@#$%^&*()-+"
11
12     for char in password:
13         if char.isupper():
14             has_upper = True
15         elif char.islower():
16             has_lower = True
17         elif char.isdigit():
18             has_digit = True
19         elif char in special_characters:
20             has_special = True
21
22     return has_upper and has_lower and has_digit and has_special
23
24 assert is_strong_password("Abcd@123") == True
25 assert is_strong_password("abcd1234") == False
26 assert is_strong_password("ABCD@123") == True
27
```

## 2. 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:

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.

```
lab-8.1.py > ...  
32  
33     def classify_number(n):  
34         if n > 0:  
35             return "positive"  
36         elif n < 0:  
37             return "negative"  
38         else:  
39             return "zero"  
40  
41     assert classify_number(5) == "positive"  
42     assert classify_number(-3) == "negative"  
43     assert classify_number(0) == "zero"
```

### 3 . 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:

Ignore case, spaces, and punctuation.

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.

```
# lab-8.1.py > ...
46  # write a python function to check anagram strings and validate its implementation using assert statements
47  def are_anagrams(str1, str2):
48      return sorted(str1) == sorted(str2)
49
50  assert are_anagrams("listen", "silent") == True, "Test case 1 failed"
51  assert are_anagrams("hello", "world") == False, "Test case 2 failed"
52  assert are_anagrams("python", "typhon") == True, "Test case 3 failed"
```

## 4. 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:

```
add_item(name, quantity)
remove_item(name, quantity)
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

```

lab-8.1.py > ...
54  # write a function to check inventory management system and validate its implementation using assert statement
55  class Inventory:
56      def __init__(self):
57          self.stock = {}
58
59      def add_item(self, item, quantity):
60          if item in self.stock:
61              self.stock[item] += quantity
62          else:
63              self.stock[item] = quantity
64
65      def remove_item(self, item, quantity):
66          if item in self.stock and self.stock[item] >= quantity:
67              self.stock[item] -= quantity
68          else:
69              raise ValueError("Not enough stock to remove")
70
71      def get_stock(self, item):
72          return self.stock.get(item, 0)
73
74 #test cases
75 inv = Inventory()
76 inv.add_item("Pen", 10)
77 assert inv.get_stock("Pen") == 10, "Test case 1 failed"
78 inv.remove_item("Pen", 5)
79 assert inv.get_stock("Pen") == 5, "Test case 2 failed"
80 inv.add_item("Book", 3)
81 assert inv.get_stock("Book") == 3, "Test case 3 failed"
81

```

## 5. 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:

Validate "MM/DD/YYYY" format.

Handle invalid dates.

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.

```
lab-8.1.py > ...
o2
83 # write a function to validate and format date in the format "MM/DD/YYYY" and validate its implementation using assert
84 from datetime import datetime
85 def validate_and_format_date(date_str):
86     try:
87         date_obj = datetime.strptime(date_str, "%m/%d/%Y")
88         return date_obj.strftime("%Y-%m-%d")
89     except ValueError:
90         return "Invalid Date"
91
92 # test cases
93 assert validate_and_format_date("10/15/2023") == "2023-10-15", "Test case 1 failed"
94 assert validate_and_format_date("02/30/2023") == "Invalid Date", "Test case 2 failed"
95 assert validate_and_format_date("01/01/2024") == "2024-01-01", "Test case 3 failed"
96
```