**Assignment-8.1(AI ASSITED CODING)**

**2403A51251**

**B.Rakshith**

**batch:11**

**TITLE: Test-Driven Development with AI – Generating and Working with Test Cases**

**Task 1: Password Strength Validator**

**Prompt to AI:  
Generate a Python function is\_strong\_password(password) with rules for length, uppercase, lowercase, digit, special char, and no spaces. Add 3 assert test cases.**

**AI-Generated Test Cases:**

assert is\_strong\_password("Abcd@123") == True

assert is\_strong\_password("abcd123") == False

assert is\_strong\_password("ABCD@1234") == True

**AI-Generated Code:**

import re

def is\_strong\_password(password):

    if len(password) < 8 or " " in password:

        return False

    if not re.search(r"[A-Z]", password):

        return False

    if not re.search(r"[a-z]", password):

        return False

    if not re.search(r"\d", password):

        return False

    if not re.search(r"[!@#$%^&\*(),.?\":{}|<>]", password):

        return False

    return True

**My Explanation:  
Copilot used regex to check all conditions. The function works correctly and passes the assert tests.**

**Output:  
All test cases passed.**

**Task 2: Number Classification with Loops**

**Prompt to AI:  
Write a function classify\_number(n) that checks if n is positive, negative, or zero. Handle invalid inputs too. Give at least 3 assert test cases.**

**AI-Generated Test Cases:**

assert classify\_number(10) == "Positive"

assert classify\_number(-5) == "Negative"

assert classify\_number(0) == "Zero"

**AI-Generated Code:**

def classify\_number(n):

    if not isinstance(n, int):

        return "Invalid Input"

    if n > 0:

        return "Positive"

    elif n < 0:

        return "Negative"

    else:

        return "Zero"

**My Explanation:  
The bug handling is done using type check. Edge cases like 0 and negative values are covered.**

**Output:  
All test cases passed.**

**Task 3: Anagram Checker**

**Prompt to AI:  
Make a function is\_anagram(str1, str2) that ignores spaces, case, and punctuation. Provide 3 assert test cases.**

**AI-Generated Test Cases:**

assert is\_anagram("listen", "silent") == True

assert is\_anagram("hello", "world") == False

assert is\_anagram("Dormitory", "Dirty Room") == True

**AI-Generated Code:**

import re

def is\_anagram(str1, str2):

    s1 = re.sub(r"[^a-z]", "", str1.lower())

    s2 = re.sub(r"[^a-z]", "", str2.lower())

    return sorted(s1) == sorted(s2)

**My Explanation:  
The AI cleaned spaces/punctuation with regex and compared sorted strings. This handles edge cases well.**

**Output:  
All test cases passed.**

**Task 4: Inventory Class**

**Prompt to AI:  
Create an Inventory class with add\_item, remove\_item, and get\_stock methods. Add 3 assert tests for stock updates.**

**AI-Generated 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

**AI-Generated Code:**

class Inventory:

    def \_\_init\_\_(self):

        self.stock = {}

    def add\_item(self, name, quantity):

        self.stock[name] = self.stock.get(name, 0) + quantity

    def remove\_item(self, name, quantity):

        if name in self.stock and self.stock[name] >= quantity:

            self.stock[name] -= quantity

        else:

            return "Error: Not enough stock"

    def get\_stock(self, name):

        return self.stock.get(name, 0)

**My Explanation:  
The class works for adding and removing stock. Assert tests confirmed correctness.**

**Output:  
All test cases passed.**

**Task 5: Date Validation & Formatting**

**Prompt to AI:  
Write a function validate\_and\_format\_date(date\_str) to check MM/DD/YYYY format, handle invalid dates, and convert to YYYY-MM-DD. Add 3 assert tests.**

**AI-Generated 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"

**AI-Generated Code:**

from datetime import datetime

def validate\_and\_format\_date(date\_str):

    try:

        date = datetime.strptime(date\_str, "%m/%d/%Y")

        return date.strftime("%Y-%m-%d")

    except ValueError:

        return "Invalid Date"

**My Explanation:  
The AI used Python’s datetime library to validate and format. It catches invalid dates correctly.**

**Output:  
All test cases passed.**

**Conclusion**

**In this lab I practiced TDD with AI. Writing test cases first helped me focus on requirements. AI gave useful suggestions but I also checked and corrected them. I feel more confident now in debugging and test-based development.**