AI ASSISTED CODING

Assignment 8.1

Ashmitha Reddy

2303A52111

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:

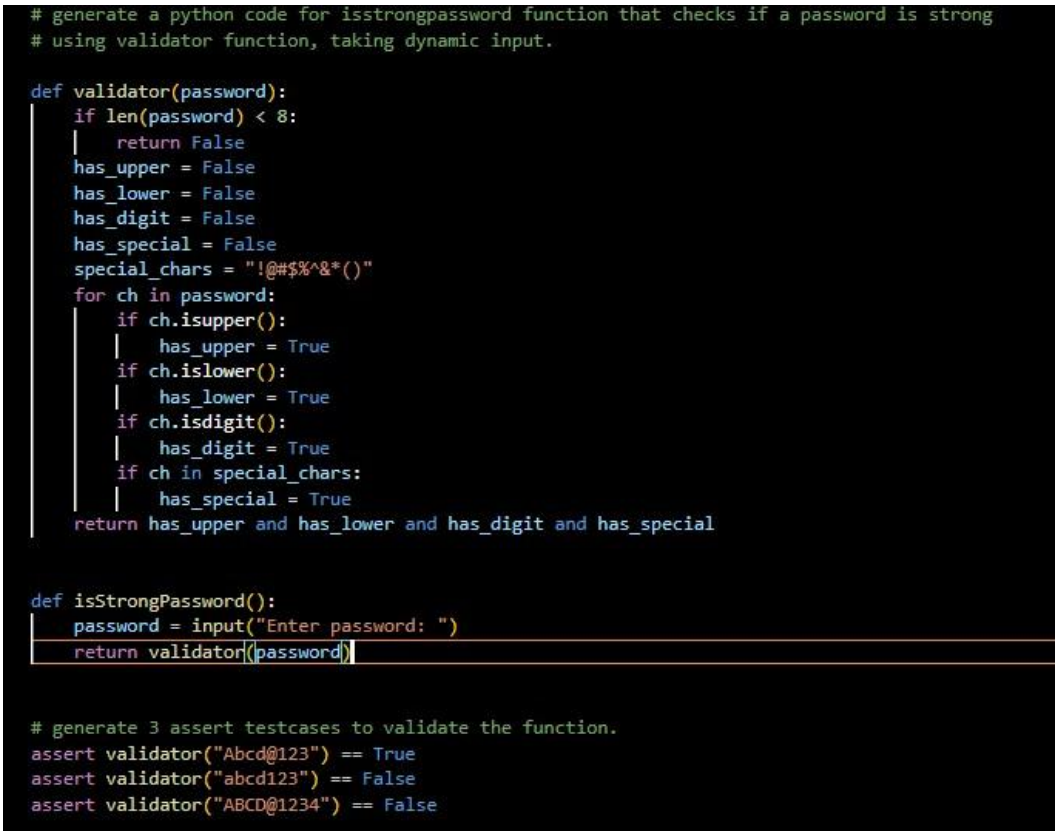
assert is_strong_password("Abcd@123") == True

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 AND CODE:



```python
# generate a python code for isstrongpassword function that checks if a password is strong
# using validator function, taking dynamic input.

def validator(password):
    if len(password) < 8:
        return False
    has_upper = False
    has_lower = False
    has_digit = False
    has_special = False
    special_chars = "!@#$%^&*()"
    for ch in password:
        if ch.isupper():
            has_upper = True
        if ch.islower():
            has_lower = True
        if ch.isdigit():
            has_digit = True
        if ch in special_chars:
            has_special = True
    return has_upper and has_lower and has_digit and has_special


def isStrongPassword():
    password = input("Enter password: ")
    return validator(password)


# generate 3 assert testcases to validate the function.
assert validator("Abcd@123") == True
assert validator("abcd123") == False
assert validator("ABCD@1234") == False
```

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 AND CODE:

```
# generate a python code with function classify number that takes a number as input
# and classifies it as positive, negative, or zero.

def classify_number(num):
    if num > 0:
        return "Positive"
    elif num < 0:
        return "Negative"
    else:
        return "Zero"

print(classify_number(5))

# generate 3 assert testcases to validate the function.
assert classify_number(5) == "Positive"
assert classify_number(-3) == "Negative"
assert classify_number(0) == "Zero"
```

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

PROMPT AND CODE:

Expected Output #3:

• Function correctly identifying anagrams and passing all AI-generated tests.

```python
# write a function isanagram that takes two strings as input and checks if they are anagrams

def isanagram(str1, str2):
    import re

    # Remove spaces and punctuation, and convert to lowercase
    str1_clean = re.sub(r'[\W_]+', '', str1).lower()
    str2_clean = re.sub(r'[\W_]+', '', str2).lower()

    # Sort the characters of both strings and compare
    return sorted(str1_clean) == sorted(str2_clean)

# generate 3 assert testcases to validate the function.
assert isanagram("listen", "silent") == True
assert isanagram("hello", "world") == False
assert isanagram("dirty room", "dormetory") == True
```

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 AND CODE:

```python
# generate an inventory class that allows to add_items, remove_items and get_stock.

class Inventory:
    def __init__(self):
        self.stock = {}

    def add_items(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity

    def remove_items(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            if self.stock[item] == 0:
                del self.stock[item]
            return True
        return False

    def get_stock(self, item):
        return self.stock.get(item, 0)

# generate 3 assert testcases to validate the function.
inv = Inventory()
inv.add_items("pen", 10)
assert inv.get_stock("pen") == 10
inv.remove_items("pen", 5)
assert inv.get_stock("pen") == 5
inv.add_items("notebook", 3)
assert inv.get_stock("notebook") == 3
```

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 AND CODE:

```python
# write a validate_and_format_date function

from datetime import datetime

def validate_and_format_date(date_str):
    try:
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid date"

# Test cases
assert validate_and_format_date("12/31/2020") == "2020-12-31"
assert validate_and_format_date("02/29/2020") == "2020-02-29"  # Leap year
assert validate_and_format_date("13/01/2020") == "Invalid date"
assert validate_and_format_date("04/31/2020") == "Invalid date"  # April has 30 days

print("All tests passed ✅")
```