AI Assistant Coding

Assignment-8.1

Name:K.Trinay Prasad

HT No:2303A52070

Batch:32

Task 1:

(Password Strength Validator – Apply AI in Security

Context)

• **Task:** Apply AI to generate at least 3 assert test cases for is_password _strong

(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

Code:

```
def password_valid(password):
    if len(password) < 8:
        return False
    for char in password:
        if not char.isalnum():
            return False
    has_upper = False
    has_lower = False
    has_digit = False
    has_special = False
    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 "!@#$%^&*()-_=+[]{}|;:'\",.<>?/":
            has_special = True
    return has_upper and has_lower and has_digit and has_special

#Test cases
assert password_valid("Abcd@123") == True
assert password_valid("abcd1234") == False
assert password_valid("ABCD@1234") == True
```

Task 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.

Code:

```
def classify_number(n):
    invalid_input = ["", None, " ","abc", [], {}, 3.14]
    for invalid in invalid_input:
        if n == invalid:
            return "Invalid input"
    for _ in [n]:
        if not isinstance(n, int):
            return "Invalid input"
    if n < 0:
        return "Negative"
    if n == 0:
        return "Zero"
    return "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number(10) == "Positive"
assert classify_number(-1) == "Negative"
assert classify_number(1) == "Positive"
assert classify_number("") == "Invalid input"
assert classify_number(None) == "Invalid input"
assert classify_number("abc") == "Invalid input"
```

Task 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 AIgenerated tests.

Code:

```
def is_anagram(str1, str2):
    # Remove spaces and convert to lowercase
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()
    # Sort the characters of both strings and compare
    return sorted(str1) == sorted(str2)
#Test cases
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("racecar", "carrace") == True
assert is_anagram("Dormitory", "Dirty Room") == True
```

Task 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.

Code:

```
class inventory:
    def __init__(self):
        self.items = {}

    def add_item(self, name, quantity):
        if name in self.items:
            self.items[name] += quantity
        else:
            self.items[name] = quantity
    def remove_item(self, name, quantity):
        if name in self.items and self.items[name] >= quantity:
            self.items[name] -= quantity
            if self.items[name] == 0:
                del self.items[name]
        else:
            print(f"Not enough {name} in inventory to remove.")
    def get_stock(self, name):
        return self.items.get(name, 0)
#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", 20)
assert inv.get_stock("book") == 20
inv.remove_item("book", 15)# Should print a warning message
assert inv.get_stock("book") == 5
```

Task 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.

Code:

```
#Validate "MM/DD/YYYY" format and convert to "YYYY-MM-DD".
def validate_date(date_str):
    try:
        if not isinstance(date_str, str):
            return "Invalid input"
        parts = date_str.split("/")
        if len(parts) != 3:
            return "Invalid date"
        excepted_length = [2, 2, 4]
        for part, length in zip(parts, excepted_length):
            if len(part) != length:
                return "Invalid date"
            for char in part:
                if not char.isdigit():
                    return "Invalid date"
        month, day, year = map(int, parts)
        if year < 1 or month < 1 or month > 12 or day < 1 or day > 31:
            return "Invalid date"
        if month in [1, 3, 5, 7,8, 10, 12]:
            if day > 31:
                return "Invalid date"
        elif month in [4, 6, 9, 11]:
            if day > 30:
                return "Invalid date"
        else: # February
            if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
                if day > 29:
                    return "Invalid date"
            else:
                if day > 28:
                    return "Invalid date"
        return f"{year:04d}-{month:02d}-{day:02d}"
    except Exception as e:
        return "Invalid input"
```

```python
#Test cases
assert validate_date("02/29/2020") == "2020-02-29"
assert validate_date("13/02/2020") == "Invalid date"
assert validate_date("11/30/2020") == "2020-11-30"
```