

AI Assistant Coding

Assignment 8.1

Name : N. Eshwar

HT. No : 2303A52072

Batch: 32

Task Description #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 char.isspace():  
            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("Password123!") == True
assert password_valid("Abcd@123") == True
assert password_valid("abcd123") == False
assert password_valid("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.

Code:

```
def classify_number(n):
    invalid_inputs = ["", None, " ", "abc", [], {}, 3.14]
    for invalid in invalid_inputs:
        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("hello") == "Invalid input"
```

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-generated tests.

Code:

```
def is_anagram(s1, s2):  
    # Remove spaces and convert to lowercase  
    s1 = s1.replace(" ", "").lower()  
    s2 = s2.replace(" ", "").lower()  
  
    # Sort the characters of both strings and compare  
    return sorted(s1) == sorted(s2)  
# Test cases  
assert is_anagram("listen", "silent") == True  
assert is_anagram("triangle", "integral") == True  
assert is_anagram("Dormitory", "Dirty Room") == True  
assert is_anagram("hello", "world") == False
```

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.

Code:

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

    def add_item(self, item_name, quantity):
        if item_name in self.items:
            self.items[item_name] += quantity
        else:
            self.items[item_name] = quantity

    def remove_item(self, item_name, quantity):
        if item_name in self.items and self.items[item_name] >= quantity:
            self.items[item_name] -= quantity
            if self.items[item_name] == 0:
                del self.items[item_name]
        else:
            print(f"Not enough {item_name} in inventory to remove.")

    def get_stock(self, item_name):
        return self.items.get(item_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("notebook", 20)
assert inv.get_stock("notebook") == 20
inv.remove_item("notebook", 25) # Should print a warning
assert inv.get_stock("notebook") == 20
```

Output:

```
Not enough notebook in inventory to remove
```

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.

Code:

```
#Validate "MM/DD/YYYY" format and convert to "YYYY-MM-DD".
def validate_and_format_date(date_str):
    try:
        if not isinstance(date_str, str):
            return "Invalid Date"

        parts = date_str.split('/')
        if len(parts) != 3:
            return "Invalid Date"

        expected_lengths = [2, 2, 4]
        for part, length in zip(parts, expected_lengths):
            if len(part) != length:
                return "Invalid Date"
            for ch in part:
                if not ch.isdigit():
                    return "Invalid Date"

        month, day, year = map(int, parts)
        if year < 1 or month < 1 or month > 12 or day < 1:
            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 ValueError:
    return "Invalid Date"

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"
assert validate_and_format_date("02/29/2024") == "2024-02-29"
assert validate_and_format_date("02/29/2023") == "Invalid Date"
assert validate_and_format_date("13/01/2024") == "Invalid Date"
```