2303A510B7 - Harini.S
Assignment 8.1
Batch - 14

Test-Driven Development with AI – Generating and Working with Test Cases

TASK 1: Password Strength Validator – Apply AI in Security Context

Prompt:

- Now implement the Python function is_strong_password(password) so that it passes all the previously generated assert test cases. Handle invalid inputs properly and ensure clean error handling. Return only the function implementation.
- Generate at least 3 assert-based test cases for a Python function is_strong_password(password).
- Requirements: Minimum 8 characters, Must include uppercase, lowercase, digit, and special character, Must not contain spaces and Include both valid and invalid test cases. Return only assert statements.

Code:

```
def is_strong_password(password):
    if not isinstance(password, str):
        raise ValueError("Input must be a string.")
    if len(password) < 8:
        return False
    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(not c.isalnum() for c in password)
    return has_upper and has_lower and has_digit and has_special
print(is_strong_password("StrongPass1!"))
print(is_strong_password("weakpass"))
print(is_strong_password("NoSpecialChar1"))
print(is_strong_password("Short1!"))
print(is_strong_password("ValidPass123$"))
print(is_strong_password("NoDigits!"))
print(is_strong_password("NoUppercase1!"))
print(is_strong_password("NoLowercase1!"))
print(is_strong_password("Has Space1!"))
print(is_strong_password(12345678))
```

Output:

```
True
False
False
False
True
False
True
True
True
Traceback (most recent call last):
  File "c:\Users\akshi\OneDrive\Dokumen\New folder\AI-AC.py", line 20, in <module>
    print(is_strong_password(12345678))
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\akshi\OneDrive\Dokumen\New folder\AI-AC.py", line 3, in is_strong_password
    raise ValueError("Input must be a string.")
ValueError: Input must be a string.
```

TASK 2: Number Classification with Loops – Apply AI for Edge Case Handling

Prompt:

- Generate at least 3 assert test cases for a Python function classify_number(n).
- Requirements: Return "Positive", "Negative", or "Zero", Handle boundary cases (-1, 0, 1), Handle invalid inputs like strings and None and Return only assert statements.
- Implement classify_number(n) using loops. The function must: Classify numbers as Positive, Negative, or Zero,

Return "Invalid Input" for non-numeric values. Ensure it passes all previously generated test cases. Return only the function code.

Code:

```
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
print(classify_number(5))
print(classify_number(-3))
print(classify_number(0))
print(classify_number(-1))
print(classify_number(1))
print(classify_number("string"))
print(classify_number(None))
```

Output:

```
Positive
Negative
Zero
Negative
Positive
Invalid Input
Invalid Input
```

TASK 3: Anagram Checker – Apply AI for String Analysis

Prompt:

- Generate at least 3 assert test cases for a function is_anagram(str1, str2). Implement is_anagram(str1, str2) in Python.
- Requirements: Ignore case differences, Ignore spaces and punctuation, Handle empty strings, Include both True and False cases and Return only assert statements.
- Requirements: Ignore case, spaces, and punctuation, Return True if strings are anagrams, Handle edge cases like empty strings. Ensure it passes all generated test cases. Return only the function implementation.

Code:

```
import re
def is_anagram(str1, str2):
    str1_cleaned = re.sub(r'[\s\W]+', '', str1).lower()
    str2_cleaned = re.sub(r'[\s\W]+', '', str2).lower()
    return sorted(str1_cleaned) == sorted(str2_cleaned)
print(is_anagram("listen", "silent"))
print(is_anagram("Triangle", "Integral"))
print(is_anagram("Dormitory", "Dirty Room"))
print(is_anagram("Hello", "World"))
print(is_anagram("Software", "Swear oft"))
print(is_anagram("Anagram", "Nag a ram"))
print(is_anagram("Not an anagram", "Definitely not"))
print(is_anagram("The eyes", "They see"))
print(is_anagram("Software", "Swear oft"))
print(is_anagram("Anagram", "Nag a ram"))
print(is_anagram("Not an anagram", "Definitely not"))
```

Output:

```
True
True
True
False
True
True
False
True
True
True
False
```

TASK 4: Inventory Class – Apply AI to Simulate Real - World Inventory System

Prompt:

- Generate at least 3 assert-based test cases for a Python class Inventory.
- Methods: add_item(name, quantity), remove_item(name, quantity), get_stock(name)
- Include: Adding new items, Removing stock, Removing more than available stock. Return only the assert test cases.
- Implement a Python class Inventory with methods: add_item(name, quantity), remove_item(name, quantity), get_stock(name)

- Requirements: Prevent negative stock, Handle non-existing items safely. Ensure it passes all previously generated assert test cases. Return only the class implementation.

Code:

```
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, name, quantity):
        if quantity < 0:
            raise ValueError("Quantity cannot be negative.")
        if name in self.stock:
            self.stock[name] += quantity
        else:
            self.stock[name] = quantity
    def remove_item(self, name, quantity):
        if name not in self.stock:
            raise KeyError("Item does not exist.")
        if quantity < 0:
            raise ValueError("Quantity cannot be negative.")
        if self.stock[name] < quantity:
            raise ValueError("Not enough stock to remove.")
        self.stock[name] -= quantity
    def get_stock(self, name):
        return self.stock.get(name, 0)
inventory = Inventory()
inventory.add_item("apple", 10)
print(inventory.get_stock("apple"))
inventory.remove_item("apple", 5)
print(inventory.get_stock("apple"))
inventory.add_item("grape", 10)
print(inventory.get_stock("grape"))
```

Output:

```
10
5
10
```

TASK 5: Date Validation & Formatting – Apply AI for Data Validation

Prompt:

- Generate at least 3 assert test cases for a Python function validate_and_format_date(date_str).
- Requirements: Input format: "MM/DD/YYYY", Convert valid dates to "YYYY-MM-DD", Return "Invalid Date" for invalid formats or impossible dates, Include edge cases. Return only assert statements.
- Implement validate_and_format_date(date_str) in Python.
- Requirements: Validate input format "MM/DD/YYYY", Handle invalid dates properly, Convert valid dates to "YYYY-MM-DD". Ensure the function passes all previously generated assert tests. Return only the function implementation.

Code:

```
def validate_and_format_date(date_str):
    import re
    from datetime import datetime
    if not re.match(r'^\d{2}/\d{2}/\d{4}$', date_str):
        return "Invalid Date"
    try:
        date_obj = datetime.strptime(date_str, '%m/%d/%Y')
        return date_obj.strftime('%Y-%m-%d')
    except ValueError:
        return "Invalid Date"
print(validate_and_format_date("12/31/2020"))
print(validate_and_format_date("13/01/2020"))
print(validate_and_format_date("00/10/2020"))
print(validate_and_format_date("02/29/2020"))
print(validate_and_format_date("abc"))
print(validate_and_format_date("12/31/20"))
print(validate_and_format_date("12-31-2020"))
print(validate_and_format_date("12/31/2020 "))
print(validate_and_format_date(" 12/31/2020"))
print(validate_and_format_date("12/31/2020/"))
print(validate_and_format_date("02/29/2028"))
print(validate_and_format_date("02/29/2023"))
print(validate_and_format_date("01/01/0000"))
print(validate_and_format_date("01/01/10000"))
```

Output:

```
2020-12-31
Invalid Date
Invalid Date
2020-02-29
Invalid Date
Invalid Date
Invalid Date
Invalid Date
Invalid Date
Invalid Date
2028-02-29
Invalid Date
Invalid Date
Invalid Date
```