Lab Assignmnet-8.1

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

T Hrushikesh

2303A51788

Batch-12

Task 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.

Prompt:

#Generate a python code for password validation function that checks if a password is at least 8 characters long, contains at least one uppercase letter, one lowercase letter, one digit, and one special character.

#Generate test cases for high end password checker

Code:

```python
#Generate a python code for password validation function that checks if a password is at least 8 characters long,
# contains at least one uppercase letter, one lowercase letter, one digit, and one special character.
import re
def validate_password(password):
    if len(password) < 8:
        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
#Generate test cases for high end password checker
def test_validate_password():
    assert validate_password("Password1!") == True
    assert validate_password("pass") == False
    assert validate_password("PASSWORD1!") == False
    assert validate_password("password1!") == False
    assert validate_password("Password!") == False
    assert validate_password("Password1") == False
    assert validate_password("P@ssw0rd") == True
    assert validate_password("P@ss") == False
    assert validate_password("12345678") == False
    assert validate_password("Passw0rd!") == True
    print("All test cases pass")
test_validate_password()

```

Output:

```
PS C:\Users\chari> & C:/Users/chari/AppData/Local/Programs/Python/Python311/python.exe "
.py"
All test cases passed as per requirements.
```

Explaination: This task involves creating a password validation function that ensures a password meets specific security criteria. The function checks if the password is at least 8 characters long and contains at least one uppercase letter, one lowercase letter, one digit, and one special character.By running these tests, we can confirm that the password validation logic is robust and reliable.

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.

Prompt:

#Generate a python code for number classification with loops

#Generate assert statements for number classification function

Code:

```python
# Task2
#Generate a python code for number classification with loops
def classify_numbers(numbers):
    classification = {}
    for number in numbers:
        if number < 0:
            classification[number] = 'Negative'
        elif number == 0:
            classification[number] = 'Zero'
        else:
            classification[number] = 'Positive'
    return classification
#Generate assert statements for number classification function
def test_classify_numbers():
    assert classify_numbers([-10, 0, 5]) == {-10: 'Negative', 0: 'Zero', 5: 'Positive'}
    assert classify_numbers([1, 2, 3]) == {1: 'Positive', 2: 'Positive', 3: 'Positive'}
    assert classify_numbers([-1, -2, -3]) == {-1: 'Negative', -2: 'Negative', -3: 'Negative'}
    assert classify_numbers([0]) == {0: 'Zero'}
    assert classify_numbers([-5, 0, 5, 10, -10]) == {-5: 'Negative', 0: 'Zero', 5: 'Positive', 10: 'Positive', -10: 'Negative'}
    print("All test cases for number classification passed.")
test_classify_numbers()
```

Output:

```
PS C:\Users\chari> & C:/Users/chari/AppData/Local/Programs/Pytho
.py"
.py"
All test cases for number classification passed.
```

Explaination: This task focuses on classifying numbers into three categories: negative, zero, and positive. The function iterates through a list of numbers and assigns each number to its respective category based on its value. By validating the function with these tests, we can ensure its accuracy and reliability in handling diverse inputs.

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.

Prompt:

#Python code for anagram checker function and give test cases using assert statements

Code:

```python
#Task3
#Python code for anagram checker function and give test cases using assert statements
def are_anagrams(str1, str2):
    return sorted(str1) == sorted(str2)
def test_are_anagrams():
    assert are_anagrams("listen", "silent") == True
    assert are_anagrams("triangle", "integral") == True
    assert are_anagrams("apple", "pale") == False
    assert are_anagrams("rat", "car") == False
    assert are_anagrams("aabbcc", "abcabc") == True
    assert are_anagrams("", "") == True
    assert are_anagrams("a", "a") == True
    assert are_anagrams("abcd", "dcba") == True
    assert are_anagrams("abcd", "abce") == False
    print("All test cases for anagram checker passed.")
test_are_anagrams()
```

Output:

```
PS C:\Users\chari> & C:/Users/chari/AppData/Local/Programs/Python/Python31
All test cases for anagram checker passed.
```

Explaination: This task is to develop an anagram checker function that determines whether two strings are anagrams of each other. Anagrams are words or phrases formed by rearranging the letters of another, and this function checks for that by sorting the characters in both strings and comparing them. By running these tests, we can confirm the function's correctness and reliability in identifying anagram relationships.

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.

Prompt:

#Python code for inventory management system using classes and give test cases using assert statements

Code:

```python
#Task4
#Python code for inventory management system using classes and give test cases using assert statements
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]
            return True
        return False

    def get_inventory(self):
        return self.items
def test_inventory():
    inventory = Inventory()
    inventory.add_item("apple", 10)
    inventory.add_item("banana", 5)
    assert inventory.get_inventory() == {"apple": 10, "banana": 5}
    inventory.add_item("apple", 5)
    assert inventory.get_inventory() == {"apple": 15, "banana": 5}
    assert inventory.remove_item("banana", 3) == True
    assert inventory.get_inventory() == {"apple": 15, "banana": 2}
    assert inventory.remove_item("banana", 2) == True
    assert inventory.get_inventory() == {"apple": 15}
    assert inventory.remove_item("banana", 1) == False
    assert inventory.get_inventory() == {"apple": 15}
    print("All test cases for inventory management system passed.")
test_inventory()
```

Output:

```
PS C:\Users\chari> & C:/Users/chari/AppData/Local/Programs/Python/Python311/python.exe "c
All test cases for inventory management system passed.
```

Explaination: This task involves creating an inventory management system using a class-based approach. The Inventory class allows for adding and removing items, as well as retrieving the current inventory state. By validating the class with these tests, we can ensure that the inventory management system operates reliably and accurately reflects the current stock status.

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.

Prompt:

#Python code date validation and formatting function with test cases using assert statements

Code:

```
#Task5
#Python code date validation and formatting function with test cases using assert statements
from datetime import datetime

def is_valid_date(date_string, date_format):
    try:
        datetime.strptime(date_string, date_format)
        return True
    except ValueError:
        return False

def format_date(date_string, input_format, output_format):
    date_obj = datetime.strptime(date_string, input_format)
    return date_obj.strftime(output_format)

def test_date_functions():
    assert is_valid_date("2023-12-25", "%Y-%m-%d") == True
    assert is_valid_date("2023-13-25", "%Y-%m-%d") == False
    assert is_valid_date("25/12/2023", "%d/%m/%Y") == True
    assert is_valid_date("25/13/2023", "%d/%m/%Y") == False

    assert format_date("2023-12-25", "%Y-%m-%d", "%d/%m/%Y") == "25/12/2023"
    assert format_date("1990-07-15", "%Y-%m-%d", "%B %d, %Y") == "July 15, 1990"
    print("All test cases for date validation and formatting passed.")

test_date_functions()
```

Output:

```
PS C:\Users\chari> & C:/Users/chari/AppData/Local/Programs/Python/Python311/python.exe
All test cases for date validation and formatting passed.
```

Explaination: This task focuses on date validation and formatting functions. The is_valid_date function checks whether a given date string conforms to a specified format, ensuring that only valid dates are accepted. By running these tests, we can confirm that the date functions work correctly and handle a variety of input formats reliably.