Lab Assignmnet-8.1

G.Datha Narayana

2303A51754

Batch-11

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

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:

```
1   #Generate a python code for password validation function that checks if a password is at least 8 characters long,
2   # contains at least one uppercase letter, one lowercase letter, one digit, and one special character.
3   import re
4   def validate_password(password):
5       if len(password) < 8:
6           return False
7       if not re.search(r'[A-Z]', password):
8           return False
9       if not re.search(r'[a-z]', password):
10          return False
11      if not re.search(r'\d', password):
12          return False
13      if not re.search(r'[!@#$%^&*()-+]', password):
14          return False
15      return True
16  #Generate test cases for high end password checker
17  def test_validate_password():
18      assert validate_password("Password1!") == True
19      assert validate_password("pass") == False
20      assert validate_password("PASSWORD1!") == False
21      assert validate_password("password1!") == False
22      assert validate_password("Password!") == False
23      assert validate_password("Password1") == False
24      assert validate_password("P@ssw0rd") == True
25      assert validate_password("P@ss") == False
26      assert validate_password("12345678") == False
27      assert validate_password("Passw0rd!") == True
28      print("All test cases pass")
29  test_validate_password()
30
```

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:

```
33    # Task2
34    #Generate a python code for number classification with loops
35    def classify_numbers(numbers):
36        classification = {}
37        for number in numbers:
38            if number < 0:
39                classification[number] = 'Negative'
40            elif number == 0:
41                classification[number] = 'Zero'
42            else:
43                classification[number] = 'Positive'
44        return classification
45    #Generate assert statements for number classification function
46    def test_classify_numbers():
47        assert classify_numbers([-10, 0, 5]) == {-10: 'Negative', 0: 'Zero', 5: 'Positive'}
48        assert classify_numbers([1, 2, 3]) == {1: 'Positive', 2: 'Positive', 3: 'Positive'}
49        assert classify_numbers([-1, -2, -3]) == {-1: 'Negative', -2: 'Negative', -3: 'Negative'}
50        assert classify_numbers([0]) == {0: 'Zero'}
51        assert classify_numbers([-5, 0, 5, 10, -10]) == {-5: 'Negative', 0: 'Zero', 5: 'Positive', 10: 'Positive', -10: 'Negative'}
52        print("All test cases for number classification passed.")
53    test_classify_numbers()
54
55
```

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:

```
56    #Task3
57    #Python code for anagram checker function and give test cases using assert statements
58    def are_anagrams(str1, str2):
59        return sorted(str1) == sorted(str2)
60    def test_are_anagrams():
61        assert are_anagrams("listen", "silent") == True
62        assert are_anagrams("triangle", "integral") == True
63        assert are_anagrams("apple", "pale") == False
64        assert are_anagrams("rat", "car") == False
65        assert are_anagrams("aabbcc", "abcabc") == True
66        assert are_anagrams("", "") == True
67        assert are_anagrams("a", "a") == True
68        assert are_anagrams("abcd", "dcba") == True
69        assert are_anagrams("abcd", "abce") == False
70        print("All test cases for anagram checker passed.")
71    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:

```
74    #Task4
75    #Python code for inventory management system using classes and give test cases using assert statements
76    class Inventory:
77        def __init__(self):
78            self.items = {}
79
80        def add_item(self, item_name, quantity):
81            if item_name in self.items:
82                self.items[item_name] += quantity
83            else:
84                self.items[item_name] = quantity
85
86        def remove_item(self, item_name, quantity):
87            if item_name in self.items and self.items[item_name] >= quantity:
88                self.items[item_name] -= quantity
89                if self.items[item_name] == 0:
90                    del self.items[item_name]
91                return True
92            return False
93
94        def get_inventory(self):
95            return self.items
96    def test_inventory():
97        inventory = Inventory()
98        inventory.add_item("apple", 10)
99        inventory.add_item("banana", 5)
100       assert inventory.get_inventory() == {"apple": 10, "banana": 5}
101       inventory.add_item("apple", 5)
102       assert inventory.get_inventory() == {"apple": 15, "banana": 5}
103       assert inventory.remove_item("banana", 3) == True
104       assert inventory.get_inventory() == {"apple": 15, "banana": 2}
105       assert inventory.remove_item("banana", 2) == True
106       assert inventory.get_inventory() == {"apple": 15}
107       assert inventory.remove_item("banana", 1) == False
108       assert inventory.get_inventory() == {"apple": 15}
109       print("All test cases for inventory management system passed.")
110   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:

```
112
113     #Task5
114     #Python code date validation and formatting function with test cases using assert statements
115     from datetime import datetime
116
117     def is_valid_date(date_string, date_format):
118         try:
119             datetime.strptime(date_string, date_format)
120             return True
121         except ValueError:
122             return False
123
124     def format_date(date_string, input_format, output_format):
125         date_obj = datetime.strptime(date_string, input_format)
126         return date_obj.strftime(output_format)
127
128     def test_date_functions():
129         assert is_valid_date("2023-12-25", "%Y-%m-%d") == True
130         assert is_valid_date("2023-13-25", "%Y-%m-%d") == False
131         assert is_valid_date("25/12/2023", "%d/%m/%Y") == True
132         assert is_valid_date("25/13/2023", "%d/%m/%Y") == False
133
134         assert format_date("2023-12-25", "%Y-%m-%d", "%d/%m/%Y") == "25/12/2023"
135         assert format_date("1990-07-15", "%Y-%m-%d", "%B %d, %Y") == "July 15, 1990"
136         print("All test cases for date validation and formatting passed.")
137
138     test_date_functions()
139
```

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.