2403A51291
G.Arun

Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases

Task 1: Password Strength Validator

AI-Generated Test Cases:

```
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True
```

Implementation:

```
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    pattern = r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).+$'
    return bool(re.match(pattern, password))
```

Analysis:

All test cases passed. Handles length, case, digit, and special character requirements.

Task 2: Number Classification with Loops

AI-Generated Test Cases:

```
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
```

Implementation:

```
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"
```

Analysis:

Works for integers and floats. Rejects invalid inputs like strings or None.

Task 3: Anagram Checker

AI-Generated Test Cases:

```
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
```

Implementation:

```
import re

def is_anagram(str1, str2):
    clean1 = re.sub(r'[^a-zA-Z]', '', str1).lower()
    clean2 = re.sub(r'[^a-zA-Z]', '', str2).lower()
    return sorted(clean1) == sorted(clean2)
```

Analysis:

Correctly ignores spaces/punctuation. Edge cases like empty strings handled.

Task 4: Inventory Class

AI-Generated 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
```

Implementation:

```
class Inventory:
  def __init__(self):
    self.stock = {}

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

  def remove_item(self, name, quantity):
    if name in self.stock and self.stock[name] >= quantity:
      self.stock[name] -= quantity
    else:
      return "Not enough stock"

  def get_stock(self, name):
    return self.stock.get(name, 0)
```

Analysis:

Supports add/remove operations. Prevents removing more than available.

Task 5: Date Validation & Formatting

AI-Generated 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"
```

Implementation:

```
from datetime import datetime

def validate_and_format_date(date_str):
  try:
    date_obj = datetime.strptime(date_str, "%m/%d/%Y")
    return date_obj.strftime("%Y-%m-%d")
  except ValueError:
    return "Invalid Date"
```

Analysis:

Correctly validates format. Invalid dates (like Feb 30) rejected.