2403a51290

T.Manikanta

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.