AI Assisted Coding
ASSIGNMENT 8.1

Name: T.Akshaya
HT No: 2303A52017
Batch: 31

Question 1: Test-Driven Development with AI

Task

Apply AI to generate at least 3 assert test cases for is_strong_password(password) and implement the validator function.

Requirements

Password must have at least 8 characters. Must include uppercase, lowercase, digit, and special character. Must not contain spaces.

Prompt

Generate at least three assert test cases for a function is_strong_password(password) that checks whether a password has at least eight characters, contains uppercase, lowercase, digit, and special character, and does not contain spaces. Then implement the function so that all the generated test cases pass successfully.

Code

```python
def is_strong_password(password):
    """
    Check if a password is strong.
    Requirements:
    - At least 8 characters
    - Contains uppercase letter
    - Contains lowercase letter
    - Contains digit
    - Contains special character
    - Does not contain spaces
    """
    if len(password) < 8:
        return False
    if ' ' in password:
        return False
    if not any(c.isupper() for c in password):
        return False
    if not any(c.islower() for c in password):
        return False
    if not any(c.isdigit() for c in password):
        return False
    special_chars = "!@#$%^&*()_+-=[]{}|;:,.<>?"
    if not any(c in special_chars for c in password):
        return False
    return True


# Test cases
assert is_strong_password("StrongPass1!") == True
assert is_strong_password("weak") == False
assert is_strong_password("NoDigit!@ABC") == False
assert is_strong_password("NoSpecial12AB") == False
```

```
assert is_strong_password("Has Space1!@") == False
assert is_strong_password("UPPERCASE1!") == False
assert is_strong_password("lowercase1!") == False

print("All test cases passed!")
```

Output

```
All tests passed!
```

Explanation

The function checks whether a password satisfies all security conditions. It verifies minimum length, presence of uppercase, lowercase, digit, and special character, and ensures no spaces exist. AI-generated assert test cases validate both valid and invalid passwords. All test cases pass successfully, confirming correct implementation.

Question 2: Number Classification with Loops

Task

Use AI to generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.

Requirements

Classify numbers as Positive, Negative, or Zero. Handle invalid inputs like strings and None. Include boundary conditions (-1, 0, 1).

Prompt

Generate at least three assert test cases for a function classify_number(n) that classifies a number as Positive, Negative, or Zero. The function must use loops and handle invalid inputs such as strings and None, including boundary cases like minus one, zero, and one. Implement the function so all test cases pass.

Code

```
def classify_number(n):
    """Classifies a number as Positive, Negative, or Zero."""
    if n is None or isinstance(n, str):
        return "Invalid input"
    if not isinstance(n, (int, float)):
        return "Invalid input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"

def test_classify_number():
    assert classify_number(1) == "Positive"
    assert classify_number(5) == "Positive"
    assert classify_number(-1) == "Negative"
    assert classify_number(-5) == "Negative"
    assert classify_number(0) == "Zero"
    assert classify_number(None) == "Invalid input"
    assert classify_number("5") == "Invalid input"
    assert classify_number(1.5) == "Positive"
    assert classify_number(-1.5) == "Negative"
    assert classify_number(0.0) == "Zero"
    print("All tests passed!")

if __name__ == "__main__":
```

```
        test_classify_number()
```

Output

```
All tests passed!
```

Explanation

The function classifies a number as Positive, Negative, or Zero. It also handles invalid inputs such as strings and None. Boundary cases like minus one, zero, and one are tested. All assert test cases pass, proving proper classification and error handling.

Question 3: Anagram Checker

Task

Use AI to generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.

Requirements

Ignore case, spaces, and punctuation. Handle edge cases (empty strings, identical words).

Prompt

Generate at least three assert test cases for a function is_anagram(str1, str2) that checks whether two strings are anagrams. The function must ignore case, spaces, and punctuation, and handle edge cases such as empty strings and identical words. Implement the function so that all generated test cases pass.

Code

```python
def is_anagram(str1, str2):
    def clean(s):
        return sorted(c.lower() for c in s if c.isalnum())
    return clean(str1) == clean(str2)

def test_is_anagram_basic():
    assert is_anagram("listen", "silent") is True
    assert is_anagram("Triangle", "Integral") is True
    assert is_anagram("apple", "pale") is False

def test_is_anagram_with_spaces_and_case():
    assert is_anagram("Dormitory", "Dirty room") is True
    assert is_anagram("Conversation", "Voices rant on") is True

def test_is_anagram_with_punctuation():
    assert is_anagram("A gentleman!", "Elegant man.") is True
    assert is_anagram("Clint Eastwood", "Old West Action!") is True

def test_is_anagram_edge_cases():
    assert is_anagram("", "") is True
    assert is_anagram("a", "A") is True
    assert is_anagram("abc", "def") is False

if __name__ == "__main__":
    test_is_anagram_basic()
    test_is_anagram_with_spaces_and_case()
    test_is_anagram_with_punctuation()
    test_is_anagram_edge_cases()
    print("All tests passed!")
```

Output

```
All tests passed!
```

Explanation

The function checks whether two strings are anagrams by ignoring case, spaces, and punctuation. It cleans the strings, sorts characters, and compares them. Edge cases such as empty strings and identical words are handled. All AI-generated test cases pass successfully.

Question 4: Inventory Class

Task

Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.

Methods

add_item(name, quantity), remove_item(name, quantity), get_stock(name)

Prompt

Generate at least three assert based test cases for an Inventory class that manages stock using methods add_item(name, quantity), remove_item(name, quantity), and get_stock(name). Implement the class so that it simulates a real world inventory system and passes all generated assertions.

Code

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

    def add_item(self, name, quantity):
        if not isinstance(name, str) or not isinstance(quantity, int):
            return "Invalid input"
        if quantity < 0:
            return "Quantity cannot be negative"
        self._inventory[name] = self._inventory.get(name, 0) + quantity
        return f"Added {quantity} of {name} to the inventory"

    def remove_item(self, name, quantity):
        if not isinstance(name, str) or not isinstance(quantity, int):
            return "Invalid input"
        if quantity < 0:
            return "Quantity cannot be negative"
        if name not in self._inventory:
            return "Item not found in inventory"
        if quantity > self._inventory[name]:
            return "Not enough stock to remove"
        self._inventory[name] -= quantity
        return f"Removed {quantity} of {name} from the inventory"

    def get_stock(self, name):
        if not isinstance(name, str):
            return "Invalid input"
        return f"Current stock of {name}: {self._inventory.get(name, 0)}"

def test_remove_item():
    inv = Inventory()
    assert inv.add_item("apple", 10) == "Added 10 of apple to the inventory"
    assert inv.remove_item("apple", 3) == "Removed 3 of apple from the inventory"
    assert inv.get_stock("apple") == "Current stock of apple: 7"
    assert inv.remove_item("apple", 8) == "Not enough stock to remove"
    assert inv.remove_item("banana", 1) == "Item not found in inventory"
```

```
        print("All tests passed!")

if __name__ == "__main__":
    test_remove_item()
```

Output

```
All tests passed!
```

Explanation

The Inventory class simulates a real-world stock management system. It supports adding items, removing items, and checking stock. The class validates inputs and handles edge cases such as insufficient stock and invalid data types. All assert-based tests pass, confirming correct behavior.

Question 5: Date Validation & Formatting

Task

Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.

Requirements

Validate MM/DD/YYYY format. Handle invalid dates. Convert valid dates to YYYY-MM-DD.

Prompt

Generate at least three assert test cases for a function validate_and_format_date(date_str) that validates dates in MM/DD/YYYY format, handles invalid dates, and converts valid dates into YYYY-MM-DD format. Implement the function so that all generated test cases pass successfully.

Code

```
def validate_and_format_date(date_str):
    from datetime import datetime
    try:
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        return None

def test_validate_and_format_date_valid():
    assert validate_and_format_date("12/31/2023") == "2023-12-31"
    assert validate_and_format_date("01/01/2000") == "2000-01-01"

def test_validate_and_format_date_invalid_format():
    assert validate_and_format_date("2023-12-31") is None
    assert validate_and_format_date("31/12/2023") is None

def test_validate_and_format_date_invalid_date():
    assert validate_and_format_date("02/30/2023") is None
    assert validate_and_format_date("13/01/2023") is None
    assert validate_and_format_date("00/10/2023") is None

if __name__ == "__main__":
    test_validate_and_format_date_valid()
    test_validate_and_format_date_invalid_format()
    test_validate_and_format_date_invalid_date()
    print("All tests passed!")
```

Output

```
All tests passed!
```

Explanation

The function validates dates in MM/DD/YYYY format using datetime. If valid, it converts them to YYYY-MM-DD format. Invalid formats and incorrect dates are handled properly. All AI-generated test cases pass, ensuring accurate validation and formatting.