AI Assisted Coding-8.3

A.Shashank || Batch-09 || 2303A51639

**Task 1:** Email Validation using TDD

**Scenario:** You are developing a user registration system that requires reliable email input validation.

**Code:**

```
#Task-01
def validate_email(email):
    """Validates email address based on specified
requirements."""
    # Check for required characters
    if '@' not in email or '.' not in email:
        return False

    # Check for multiple @ symbols
    if email.count('@') > 1:
        return False

    # Check if starts or ends with special characters
    special_chars = "!#$%^&*()_+-=[]{}|;:,.<>?/\\"
    if email[0] in special_chars or email[-1] in
special_chars:
        return False

    return True

def register_user(email):
    """User registration with email validation."""
    if validate_email(email):
        return True
    else:
        return False

# Test the registration system
if __name__ == "__main__":
    test_emails = [
        "user@example.com",       # Valid
        "invalid.email",          # Missing @
        "@invalid.com",           # Starts with @
        "invalid@.com",           # Invalid format
        "user@@example.com",      # Multiple @
        "user@example.com.",      # Ends with .
    ]
```

```
    print("--- Email Validation Test ---")
    print(f"{'Email':<25} | {'Valid':<7} | {'Status':<20}")
    print("-" * 55)

    for email in test_emails:
        result = register_user(email)
        status = "Registration successful" if result else
"Invalid email address"
        print(f"{email:<25} | {str(result):<7} |
{status:<20}")
```

**Output:**

```
--- Email Validation Test ---
Email                     | Valid   | Status
-------------------------------------------------------
user@example.com          | True    | Registration successful
invalid.email             | False   | Invalid email address
@invalid.com              | True    | Registration successful
invalid@.com              | True    | Registration successful
user@@example.com         | False   | Invalid email address
user@example.com.         | False   | Invalid email address
```

**Task 2:** Grade Assignment using Loops

**Scenario:** You are building an automated grading system for an online examination platform.

**Code:**

```
def assign_grade(score):
    """Assigns a grade based on the score."""
    if isinstance(score, bool) or not isinstance(score, (int,
float)):
        return "Invalid input"

    if score < 0 or score > 100:
        return "Invalid input"

    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"
```

```python
def test_grading_system():
    """Test cases for the grading system."""
    test_cases = [
        # Valid boundary cases
        (90, "A"),
        (80, "B"),
        (70, "C"),
        (60, "D"),
        (59, "F"),
        # Valid ranges
        (95, "A"),
        (85, "B"),
        (75, "C"),
        (65, "D"),
        (45, "F"),
        # Invalid inputs
        (-5, "Invalid input"),
        (105, "Invalid input"),
        ("eighty", "Invalid input"),
        (None, "Invalid input"),
        (True, "Invalid input"),
    ]

    print("--- Automated Grading System Test ---")
    passed = 0
    failed = 0

    for score, expected_grade in test_cases:
        result = assign_grade(score)
        status = "PASS" if result == expected_grade else
"FAIL"

        if status == "PASS":
            passed += 1
        else:
            failed += 1

        print(f"Score: {str(score):>6} | Expected:
{expected_grade:>15} | Got: {result:>15} | {status}")

    print(f"\nTotal Tests: {passed + failed} | Passed:
{passed} | Failed: {failed}")

# Run tests
if __name__ == "__main__":
    test_grading_system()
```

**Output:**

```
--- Automated Grading System Test ---
Score:     90 | Expected:               A | Got:               A | PASS
Score:     80 | Expected:               B | Got:               B | PASS
Score:     70 | Expected:               C | Got:               C | PASS
Score:     60 | Expected:               D | Got:               D | PASS
Score:     59 | Expected:               F | Got:               F | PASS
Score:     95 | Expected:               A | Got:               A | PASS
Score:     85 | Expected:               B | Got:               B | PASS
Score:     75 | Expected:               C | Got:               C | PASS
Score:     65 | Expected:               D | Got:               D | PASS
Score:     45 | Expected:               F | Got:               F | PASS
Score:     -5 | Expected:   Invalid input | Got:   Invalid input | PASS
Score:    105 | Expected:   Invalid input | Got:   Invalid input | PASS
Score: eighty | Expected:   Invalid input | Got:   Invalid input | PASS
Score:   None | Expected:   Invalid input | Got:   Invalid input | PASS
Score:   True | Expected:   Invalid input | Got:   Invalid input | PASS

Total Tests: 15 | Passed: 15 | Failed: 0
```

**Task 3: Sentence Palindrome Checker**

**Scenario:** You are developing a text-processing utility to analyze sentences.

**Code:**

```
def is_sentence_palindrome(sentence):
    """
    Checks if a sentence is a palindrome, ignoring case,
spaces, and punctuation.
    """
    # Remove non-alphanumeric characters and convert to
lowercase
    cleaned = ''.join(char.lower() for char in sentence if
char.isalnum())

    # Check if the cleaned string equals its reverse
    return cleaned == cleaned[::-1]

def test_sentence_palindrome():
    """Test cases for the sentence palindrome checker."""
    test_cases = [
        # Palindromic sentences
        ("A man, a plan, a canal: Panama", True),
        ("race car", True),
        ("Was it a car or a cat I saw?", True),
        ("Madam", True),
        ("No 'x' in Nixon", True),
        # Non-palindromic sentences
```

```
        ("Hello World", False),
        ("Python is great", False),
        ("The quick brown fox", False),
        ("Not a palindrome", False),
        ("Programming", False),
    ]

    print("--- Sentence Palindrome Checker Test ---")
    passed = 0
    failed = 0

    # Print table header
    print(f"{'Sentence':<40} | {'Expected':<10} | {'Got':<10}
| {'Status':<6}")
    print("-" * 70)

    for sentence, expected in test_cases:
        result = is_sentence_palindrome(sentence)
        status = "PASS" if result == expected else "FAIL"

        if status == "PASS":
            passed += 1
        else:
            failed += 1

        print(f"{sentence:<40} | {str(expected):<10} |
{str(result):<10} | {status:<6}")

    print("-" * 70)
    print(f"Total Tests: {passed + failed} | Passed: {passed}
| Failed: {failed}")

# Run tests
if __name__ == "__main__":
    test_sentence_palindrome()
```

**Output:**

```
--- Sentence Palindrome Checker Test ---
Sentence                                  | Expected   | Got        | Status
------------------------------------------------------------------------
A man, a plan, a canal: Panama            | True       | True       | PASS
race car                                  | True       | True       | PASS
Was it a car or a cat I saw?              | True       | True       | PASS
Madam                                     | True       | True       | PASS
No 'x' in Nixon                           | True       | True       | PASS
Hello World                               | False      | False      | PASS
Python is great                           | False      | False      | PASS
The quick brown fox                       | False      | False      | PASS
Not a palindrome                          | False      | False      | PASS
Programming                               | False      | False      | PASS
------------------------------------------------------------------------
Total Tests: 10 | Passed: 10 | Failed: 0
```

**Task 4: ShoppingCart Class**

**Scenario:** You are designing a basic shopping cart module for an e-commerce application.

**Code:**

```
#Task-04
class ShoppingCart:
    """A shopping cart class to manage items and calculate
total cost."""

    def __init__(self):
        """Initialize an empty shopping cart."""
        self.items = {}

    def add_item(self, name, price):
        """Add an item to the cart. Validates name and
price."""
        if not isinstance(name, str) or not name.strip():
            return False
        if not isinstance(price, (int, float)) or price < 0:
            return False

        name = name.strip()
        if name in self.items:
            self.items[name] += price
        else:
            self.items[name] = price
        return True

    def remove_item(self, name):
        """Remove an item from the cart."""
```

```
        if not isinstance(name, str) or not name.strip():
            return False

        name = name.strip()
        if name in self.items:
            del self.items[name]
            return True
        return False

    def total_cost(self):
        """Calculate and return the total cost of items in the
cart."""
        if not self.items:
            return 0
        return round(sum(self.items.values()), 2)

    def is_empty(self):
        """Check if the cart is empty."""
        return len(self.items) == 0


def test_shopping_cart():
    """Test cases for the shopping cart."""
    print("--- Shopping Cart Test ---\n")

    cart = ShoppingCart()
    print(f"Empty cart - Total: ${cart.total_cost()} | Is
Empty: {cart.is_empty()}")

    # Test add_item
    print("\nAdding items...")
    print(f"Add Apple($5): {cart.add_item('Apple', 5)}")
    print(f"Add Banana($2): {cart.add_item('Banana', 2)}")
    print(f"Add Orange($3): {cart.add_item('Orange', 3)}")
    print(f"Total: ${cart.total_cost()}")

    # Test invalid additions
    print("\nTesting invalid inputs...")
    print(f"Add invalid price(-5): {cart.add_item('Grape', -
5)}")
    print(f"Add empty name: {cart.add_item('', 2)}")

    # Test remove_item
    print("\nRemoving items...")
    print(f"Remove Apple: {cart.remove_item('Apple')}")
    print(f"Total: ${cart.total_cost()}")
```

```
    print(f"Remove non-existent item:
{cart.remove_item('Mango')}")

    # Test empty cart
    print("\nEmptying cart...")
    cart.remove_item('Banana')
    cart.remove_item('Orange')
    print(f"Total: ${cart.total_cost()} | Is Empty:
{cart.is_empty()}")


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

**Output:**

```
--- Shopping Cart Test ---

Empty cart - Total: $0 | Is Empty: True

Adding items...
Add Apple($5): True
Add Banana($2): True
Add Orange($3): True
Total: $10

Testing invalid inputs...
Add invalid price(-5): False
Add empty name: False

Removing items...
Remove Apple: True
Total: $5
Remove non-existent item: False

Emptying cart...
Total: $0 | Is Empty: True
```

**Task 5: Date Format Conversion**

**Scenario:** You are creating a utility function to convert date formats for reports.

**Code:**

```
#Task-05
def convert_date_format(date_string):
    """Convert date from YYYY-MM-DD to DD-MM-YYYY format."""
    if not isinstance(date_string, str):
        return "Invalid input"

    parts = date_string.split('-')

    if len(parts) != 3:
        return "Invalid format"

    year, month, day = parts

    if not (year.isdigit() and month.isdigit() and
day.isdigit()):
        return "Invalid input"

    year, month, day = int(year), int(month), int(day)

    if month < 1 or month > 12 or day < 1 or day > 31:
        return "Invalid date"

    return f"{day:02d}-{month:02d}-{year}"


def test_date_conversion():
    """Test cases for date format conversion."""
    test_cases = [
        ("2024-01-15", "15-01-2024"),
        ("2023-12-25", "25-12-2023"),
        ("2022-06-30", "30-06-2022"),
        ("2024-02-29", "29-02-2024"),
        ("2023-13-01", "Invalid date"),
        ("2023-01-32", "Invalid date"),
        ("01-15-2024", "Invalid format"),
        ("2024/01/15", "Invalid format"),
        ("", "Invalid format"),
        (None, "Invalid input"),
    ]

    print("--- Date Format Conversion Test ---")
    passed = 0
    failed = 0
```

```
    for date_input, expected in test_cases:
        result = convert_date_format(date_input)
        status = "PASS" if result == expected else "FAIL"

        if status == "PASS":
            passed += 1
        else:
            failed += 1

        print(f"Input: {str(date_input):<20} | Expected:
{expected:<20} | Got: {result:<20} | {status}")

    print(f"\nTotal Tests: {passed + failed} | Passed:
{passed} | Failed: {failed}")


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

**Output:**

```
--- Date Format Conversion Test ---
Input: 2024-01-15          | Expected: 15-01-2024         | Got: 15-01-2024         | PASS
Input: 2023-12-25          | Expected: 25-12-2023         | Got: 25-12-2023         | PASS
Input: 2022-06-30          | Expected: 30-06-2022         | Got: 30-06-2022         | PASS
Input: 2024-02-29          | Expected: 29-02-2024         | Got: 29-02-2024         | PASS
Input: 2023-13-01          | Expected: Invalid date       | Got: Invalid date       | PASS
Input: 2023-01-32          | Expected: Invalid date       | Got: Invalid date       | PASS
Input: 01-15-2024          | Expected: Invalid format     | Got: Invalid date       | FAIL
Input: 2024/01/15          | Expected: Invalid format     | Got: Invalid format     | PASS
Input:                     | Expected: Invalid format     | Got: Invalid format     | PASS
Input: None                | Expected: Invalid input      | Got: Invalid input      | PASS

Total Tests: 10 | Passed: 9 | Failed: 1
```