

AAC ASSIGNMENT 8.3

M.AKHIL REDDY

2303A52315

BATCH-45

Task 1: Email Validation using TDD

Scenario

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

Requirements

- Must contain @ and . characters
- Must not start or end with special characters
- Should not allow multiple @ symbols
- AI should generate test cases covering valid and invalid email formats
- Implement is_valid_email(email) to pass all AI-generated test cases

Expected Output

- Python function for email validation
- All AI-generated test cases pass successfully
- Invalid email formats are correctly rejected
- Valid email formats return True

Prompt:

You are a senior Python developer following Test-Driven Development (TDD).

Task:

Implement email validation using TDD.

Requirements:

1. Email must contain exactly one '@' symbol.
2. Email must contain at least one '.' after the '@'.
3. Email must not start or end with special characters like '.', '@', '!', '_'.
4. Email must not contain multiple '@' symbols.
5. Email should have characters before and after '@'.
6. Domain part must contain at least one '.'.

7. No spaces allowed in the email.

Instructions:

1. First, generate comprehensive test cases (both valid and invalid emails).
 2. Cover edge cases.
 3. Then implement the function:

```
def is_valid_email(email: str) -> bool
```

4. Follow TDD approach:
 - Write test cases first
 - Then implement the function
 - Ensure all tests pass

5. Print results of all test cases clearly showing:

Email: <email> → Expected: <True/False> → Actual: <True/False>

6. Use only Python standard library (no external libraries).

Output format:

- Test cases list
 - Function implementation
 - Test execution results

Output:

Task 2: Grade Assignment using Loops

Scenario

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

Requirements

- AI should generate test cases for `assign_grade(score)` where:
 - 90–100 → A
 - 80–89 → B
 - 70–79 → C
 - 60–69 → D
 - Below 60 → F
- Include boundary values (60, 70, 80, 90)
- Include invalid inputs such as -5, 105, "eighty"
- Implement the function using a test-driven approach

Expected Output

- Grade assignment function implemented in Python
- Boundary values handled correctly
- Invalid inputs handled gracefully
- All AI-generated test cases pass

Prompt:

You are a senior Python developer following Test-Driven Development (TDD).

Task:

Build a grade assignment system.

Function to implement:

```
def assign_grade(score)
```

Grading Rules:

90–100 → A

80–89 → B

70-79 → C

60-69 → D

Below 60 → F

Requirements:

1. Generate comprehensive test cases first.
 2. Include boundary values: 60, 70, 80, 90.
 3. Include edge values: 59, 69, 79, 89, 100.
 4. Include invalid inputs:
 - Negative numbers (-5)
 - Numbers greater than 100 (105)
 - Non-numeric input ("eighty")
 5. Handle invalid inputs gracefully (return "Invalid Input").
 6. Implement the function using loops where appropriate.
 7. Print test results clearly in the format:

Score: <score> → Expected: <grade> → Actual: <grade>
 8. Ensure all test cases pass.
 9. Use only Python standard library.

Output:

Task 3: Sentence Palindrome Checker

Scenario

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

Requirements

- AI should generate test cases for `is_sentence_palindrome(sentence)`
- Ignore case, spaces, and punctuation
- Test both palindromic and non-palindromic sentences
- Example:
 - "A man a plan a canal Panama" → True

Expected Output

- Function correctly identifies sentence palindromes
- Case and punctuation are ignored
- Returns True or False accurately
- All AI-generated test cases pass

Prompt:

You are a senior Python developer following Test-Driven Development (TDD).

Task:

Implement a function:

```
def is_sentence_palindrome(sentence: str) -> bool
```

Requirements:

1. Ignore:
 - Case (uppercase/lowercase)
 - Spaces
 - Punctuation
2. Only consider alphanumeric characters.
3. Generate comprehensive test cases first.
4. Include:
 - Simple palindromes
 - Sentence palindromes
 - Non-palindromes

- Empty string
- Only punctuation

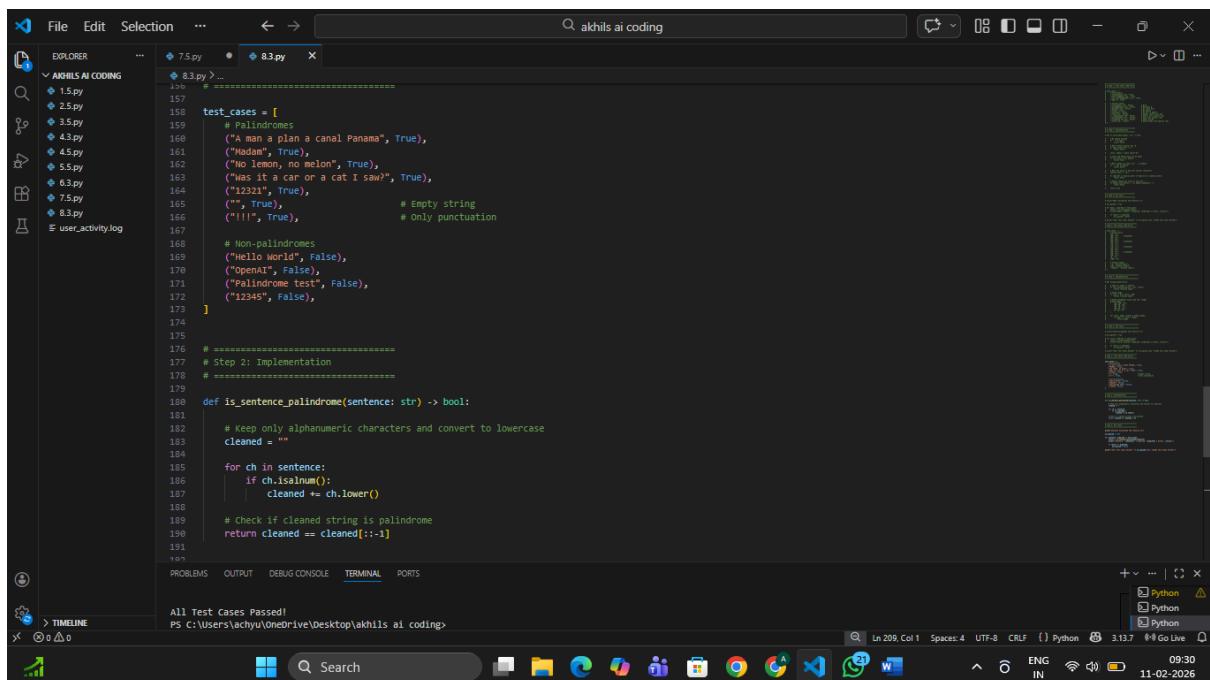
5. Print results in format:

Sentence: "<sentence>" → Expected: <True/False> → Actual: <True/False>

6. Use only Python standard library.

7. Ensure all test cases pass.

Output:



```

File: 8.3.py
1.5.py
2.5.py
3.5.py
4.3.py
4.5.py
5.5.py
6.3.py
7.5.py
8.3.py
user_activity.log

# =====#
157 test_cases = [
158     # Palindromes
159     ("A man a plan a canal Panama", True),
160     ("Madam", True),
161     ("No lemon, no melon", True),
162     ("Was it a car or a cat I saw?", True),
163     ("12321", True),
164     ("", True),           # Empty string
165     ("!!!", True),        # Only punctuation
166
167     # Non-palindromes
168     ("Hello World", False),
169     ("OpenIt", False),
170     ("Palindrome test", False),
171     ("12345", False),
172 ]
173
174 # =====#
175 # Step 2: Implementation
176 # =====#
177
178 def is_sentence_palindrome(sentence: str) -> bool:
179
180     # Keep only alphanumeric characters and convert to lowercase
181     cleaned = ""
182
183     for ch in sentence:
184         if ch.isalnum():
185             cleaned += ch.lower()
186
187     # Check if cleaned string is palindrome
188     return cleaned == cleaned[::-1]
189
190
191
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
All Test Cases Passed!
PS C:\Users\achyu\OneDrive\Desktop\akhilis ai coding>

```

Task 4: ShoppingCart Class

Scenario

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

Requirements

- AI should generate test cases for the ShoppingCart class
- Class must include the following methods:
 - add_item(name, price)
 - remove_item(name)
 - total_cost()

- Validate correct addition, removal, and cost calculation
- Handle empty cart scenarios

Expected Output

- Fully implemented ShoppingCart class
- All methods pass AI-generated test cases
- Total cost is calculated accurately
- Items are added and removed correctly

Prompt:

You are a senior Python developer following Test-Driven Development (TDD).

Task:

Design a ShoppingCart class.

Class Requirements:

1. add_item(name, price)
2. remove_item(name)
3. total_cost()

Rules:

- Item name must be string.
- Price must be positive number.
- Removing a non-existing item should be handled gracefully.
- total_cost() should return 0 for empty cart.
- Support multiple items.

Instructions:

1. First generate comprehensive test cases:

- Add single item
- Add multiple items
- Remove item
- Remove non-existing item
- Empty cart total
- Invalid price

2. Implement the ShoppingCart class.

3. Print test results clearly.

4. Ensure all test cases pass.

5. Use only Python standard library.

Output:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the "AKHILS AI CODING" folder, including 1.5.py, 2.5.py, 3.5.py, 4.3.py, 4.5.py, 5.5.py, 6.3.py, 7.5.py, 8.3.py, and user.activitylog.
- Code Editor:** Displays Python code for a `ShoppingCart` class and associated test cases. The code includes methods for initializing the cart, adding items, removing items, calculating total cost, and executing test cases. A comment indicates the use of the standard library.
- Terminal:** Shows the command "python -m unittest ShoppingCartTest.py" being run, with output indicating all tests executed successfully.
- Status Bar:** Shows system information like battery level, signal strength, and date/time (11-02-2026).

Task 5: Date Format Conversion

Scenario

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

Requirements

- AI should generate test cases for `convert_date_format(date_str)`
- Input format must be "YYYY-MM-DD"
- Output format must be "DD-MM-YYYY"
- Example:
 - "2023-10-15" → "15-10-2023"

Expected Output

- Date conversion function implemented in Python
- Correct format conversion for all valid inputs
- All AI-generated test cases pass successfully

Prompt:

You are a senior Python developer following Test-Driven Development (TDD).

Task:

Implement a function:

```
def convert_date_format(date_str: str) -> str
```

Requirements:

1. Input format must be exactly "YYYY-MM-DD".
2. Output format must be "DD-MM-YYYY".
3. Generate comprehensive test cases first.
4. Include:

- Valid dates
- Edge dates like "2023-01-01"
- Invalid formats like:

"15-10-2023"

"2023/10/15"

"2023-13-01"

"2023-00-10"

"abcd-ef-gh"

5. Handle invalid inputs gracefully (return "Invalid Date Format").

6. Print results in format:

Input: <date> → Expected: <output> → Actual: <output>

7. Use only Python standard library.

8. Ensure all test cases pass.

Output:

