Course Title	AI Assisted Coding

Name: G. Snehith Roll Number: 2303A52324 Batch: 45

Assignment Number:8.3(Present assignment number)/**24**(Total number of assignments)

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

Lab Objectives

- Introduce TDD using AI
- Generate test cases before implementation
- Emphasize testing and validation
- Encourage clean, reliable code

Lab Outcomes

Students will be able to:

- Write AI-generated test cases
- Implement code using test-first approach
- Validate using unittest
- Analyze test coverage
- Compare AI vs manual tests

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 Python developer following Test-Driven Development (TDD).

Task: Implement Email Validation for a user registration system using a test-first approach.

Step 1:

Generate comprehensive AI-created test cases using Python's unittest framework BEFORE writing the implementation.

Test Case Requirements:

- Include multiple valid email formats.
- Include multiple invalid email formats.
- Cover edge cases.

• Tests must clearly check both True and False outcomes.

Validation Rules:

1. Email must contain exactly one '@' symbol.

2. Email must contain at least one '.' character.

3. Email must not start with special characters.

4. Email must not end with special characters.

5. Email must not allow multiple '@' symbols.

6. '@' must not be the first or last character.

7. '.' must appear after '@' in the domain part.

8. Username and domain must not be empty.

Invalid Examples to Test:

• Missing '@'

• Missing '.'

• Multiple '@'

• Starts with special character

• Ends with special character

• '@' at beginning

• '@' at end

• No username

• No domain

• Dot immediately before or after '@'

Step 2:

Implement the function:

```
def is_valid_email(email):
```

Implementation Requirements:

• Do NOT use regular expressions.

• Use basic string methods only.

• Keep the code clean and readable.

• Ensure all AI-generated test cases pass.

Step 3:

Include a main block to execute unittest.

Output Format:

1. AI-generated unittest test cases

2. Implementation of is_valid_email(email)

3. Main block to run tests

The final output must be complete, clean, and runnable Python code.

Code:

```python
"""2303A52324"""
import unittest
def is_valid_email(email):
    # Must contain exactly one '@'
    if email.count('@') != 1:
        return False
    # Must contain at least one '.'
    if '.' not in email:
        return False
    # Must not start or end with special characters
    special_chars = "@."
    if email[0] in special_chars or email[-1] in special_chars:
        return False
    # Split into username and domain
    username, domain = email.split('@')
    # Username and domain must not be empty
    if not username or not domain:
        return False
    # Domain must contain '.'
    if '.' not in domain:
        return False
    # Dot must not be immediately before or after '@'
    if username.endswith('.') or domain.startswith('.'):
        return False
    return True
```

Output:

```
PS D:\3-2\AI Assitant Coding>  d:; cd 'd:\3-2\AI Assitant Coding';
e' 'c:\Users\SNEHITH\.vscode\extensions\ms-python.debugpy-2025.18.0
itant Coding\Task-8.3.py'
user@example.com --> VALID (True)
john.doe@gmail.com --> VALID (True)
student123@college.edu --> VALID (True)
userexample.com --> INVALID (False)
user@examplecom --> INVALID (False)
user@@example.com --> INVALID (False)
@example.com --> INVALID (False)
user@example.com. --> INVALID (False)
user@ --> INVALID (False)
user.@example.com --> INVALID (False)
user@.example.com --> INVALID (False)
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
PS D:\3-2\AI Assitant Coding>
```

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 Python developer following Test-Driven Development (TDD).

Task: Implement a grade assignment system for an online examination platform.

Step 1:

First generate comprehensive AI-created test cases using Python's unittest framework BEFORE writing the implementation.

Grading Rules:

• 90–100 → A

• 80–89 → B

• 70–79 → C

• 60–69 → D

• Below 60 → F

Requirements:

1. Include boundary value tests:

   - 60, 70, 80, 90

   - Also test 59, 69, 79, 89, 100

2. Include invalid inputs:

   - Negative values (e.g., -5)

   - Values greater than 100 (e.g., 105)

   - Non-numeric input (e.g., "eighty")

3. Invalid inputs must be handled gracefully.

- Return "Invalid Input" instead of crashing.

4. Use loops where appropriate to test multiple values.

5. Implement function:

def assign_grade(score):

6. Ensure:

- Correct grade is returned
- Boundary values handled correctly
- Invalid inputs handled safely
- All test cases pass successfully

Output Format:

1. AI-generated unittest test cases

2. Implementation of assign_grade(score)

3. Main block to run tests

4. Also print each score and its assigned grade clearly

5. Final output must be complete, clean, and runnable Python code.

Code:

```
"""2303A52324"""
import unittest
def assign_grade(score):
    # Handle non-numeric input
    if not isinstance(score, (int, float)):
        return "Invalid Input"
    # Handle invalid range
    if score < 0 or score > 100:
        return "Invalid Input"
    # Grade assignment
    if 90 <= score <= 100:
        return "A"
    elif 80 <= score <= 89:
        return "B"
    elif 70 <= score <= 79:
        return "C"
    elif 60 <= score <= 69:
        return "D"
    else:
        return "F"
class TestGradeAssignment(unittest.TestCase):
    def test_valid_grades(self):
        test_cases = {
            95: "A",
            90: "A",      # Boundary
```

Output:

```
PS D:\3-2\AI Assitant Coding>  d:; cd 'd:\3-2\AI Assitant Coding';
e' 'c:\Users\SNEHITH\.vscode\extensions\ms-python.debugpy-2025.18.
itant Coding\Task-8.3.py'
Score: -5 --> Grade: Invalid Input
Score: 105 --> Grade: Invalid Input
Score: eighty --> Grade: Invalid Input
.Score: 95 --> Grade: A
Score: 90 --> Grade: A
Score: 85 --> Grade: B
Score: 80 --> Grade: B
Score: 75 --> Grade: C
Score: 70 --> Grade: C
Score: 65 --> Grade: D
Score: 60 --> Grade: D
Score: 55 --> Grade: F
Score: 59 --> Grade: F
Score: 69 --> Grade: D
Score: 79 --> Grade: C
Score: 89 --> Grade: B
Score: 100 --> Grade: A
.
----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

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 Python developer following Test-Driven Development (TDD).

Task: Implement a sentence palindrome checker.

Function to implement:

def is_sentence_palindrome(sentence)

Requirements:

1. First, generate AI-created test cases using Python's unittest framework BEFORE writing the implementation.

2. Ignore:

- Case differences
- Spaces
- Punctuation

3. Only consider alphanumeric characters.

4. The function must return:

- True → if the sentence is a palindrome
- False → if not

Test Requirements:

- Include multiple palindromic sentences.
- Include multiple non-palindromic sentences.
- Include edge cases:
  - Empty string
  - Single character
  - Only punctuation
- Example:

  "A man a plan a canal Panama" → True

  "Hello World" → False

Implementation Rules:

- Do NOT use external libraries.
- You may use string methods and loops.
- Ensure all test cases pass.
- Print each sentence and whether it is a palindrome.

Output Format:

1. AI-generated unittest test cases

2. Implementation of is_sentence_palindrome

3. Main block to run tests

4. Printed output showing each sentence and result

Code:

```python
"""2303A52324"""
import unittest
import string
def is_sentence_palindrome(sentence):
    # Remove non-alphanumeric characters and convert to lowercase
    cleaned = ""
    for char in sentence:
        if char.isalnum():
            cleaned += char.lower()
    # Check palindrome
    return cleaned == cleaned[::-1]
class TestSentencePalindrome(unittest.TestCase):
    def test_palindromes(self):
        palindromes = [
            "A man a plan a canal Panama",
            "Madam",
            "Was it a car or a cat I saw",
            "No lemon no melon",
            "",
            "A"
        ]
        for sentence in palindromes:
            result = is_sentence_palindrome(sentence)
            print(f'"{sentence}" --> {result}')
            self.assertTrue(result)
```

Output:

```
PS D:\3-2\AI Assitant Coding>  d:; cd 'd:\3-2\AI Assitant Coding';
e' 'c:\Users\SNEHITH\.vscode\extensions\ms-python.debugpy-2025.18.
itant Coding\Task-8.3.py'
"Hello World" --> False
"Python Programming" --> False
"OpenAI ChatGPT" --> False
"Palindrome Test" --> False
."A man a plan a canal Panama" --> True
"Madam" --> True
"Was it a car or a cat I saw" --> True
"No lemon no melon" --> True
"" --> True
"A" --> True
.
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
PS D:\3-2\AI Assitant 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 Python developer following Test-Driven Development (TDD).

Task: Implement a ShoppingCart class for an e-commerce application.

Step 1:

First generate comprehensive AI-created test cases using Python's unittest framework BEFORE writing the implementation.

Class Requirements:

The ShoppingCart class must include:

1. add_item(name, price)

   - Adds an item with its price to the cart.

   - Price must be positive.

2. remove_item(name)

   - Removes an item from the cart.

   - If item does not exist, handle gracefully.

3. total_cost()

   - Returns total price of all items in the cart.

Test Case Requirements:

- Test adding multiple items.
- Test removing items.
- Test total cost calculation.
- Test removing non-existing item.

• Test empty cart total cost (should return 0).

• Test adding invalid price (negative or non-numeric).

Implementation Requirements:

• Use a dictionary or list to store items.

• Ensure clean, readable code.

• Handle empty cart scenario.

• All AI-generated test cases must pass.

• Print cart contents and total cost during testing.

Output Format:

1. AI-generated unittest test cases

2. Implementation of ShoppingCart class

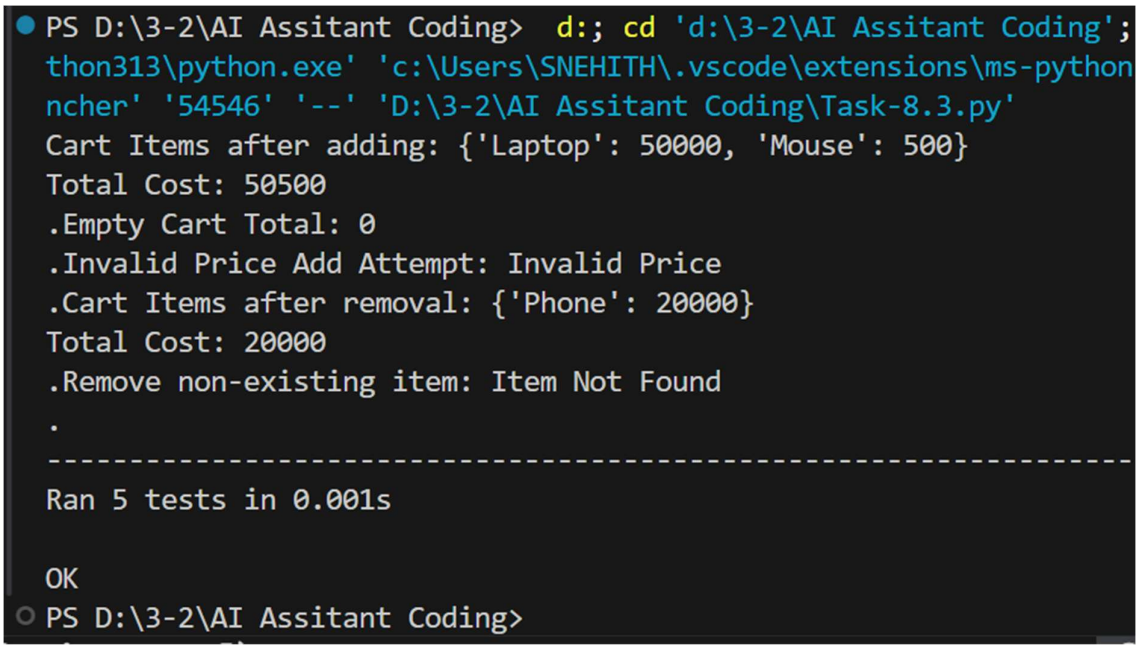
3. Main block to run tests

4. Clear printed output showing operations

Code:

```
"""2303A52324"""
import unittest
class ShoppingCart:
    def __init__(self):
        self.items = {}
    def add_item(self, name, price):
        if not isinstance(price, (int, float)) or price <= 0:
            return "Invalid Price"
        self.items[name] = price
    def remove_item(self, name):
        if name in self.items:
            del self.items[name]
        else:
            return "Item Not Found"
    def total_cost(self):
        return sum(self.items.values())
class TestShoppingCart(unittest.TestCase):
    def test_add_and_total(self):
        cart = ShoppingCart()
        cart.add_item("Laptop", 50000)
        cart.add_item("Mouse", 500)
        print("Cart Items after adding:", cart.items)
        print("Total Cost:", cart.total_cost())
        self.assertEqual(cart.total_cost(), 50500)
    def test_remove_item(self):
```

Output:



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 Python developer following Test-Driven Development (TDD).

Task: Implement a date format conversion function.

Function to implement:

def convert_date_format(date_str)

Requirements:

1. First generate AI-created test cases using Python's unittest framework BEFORE writing the implementation.

2. Input format must strictly be "YYYY-MM-DD".

3. Output format must be "DD-MM-YYYY".

4. Example:

"2023-10-15" → "15-10-2023"

Test Case Requirements:

- Include multiple valid date inputs.
- Include boundary cases like:
  - "2000-01-01"
  - "1999-12-31"
- Include invalid inputs:
  - Incorrect format (e.g., "15-10-2023")
  - Missing parts
  - Non-string input
  - Invalid month/day values
- Invalid inputs must be handled gracefully.
  - Return "Invalid Date Format"

Implementation Requirements:

- Use string methods (do not use datetime module).
- Validate format structure (YYYY-MM-DD).
- Ensure month is 01–12 and day is 01–31.
- Return converted format if valid.
- All AI-generated test cases must pass.
- Print input date and converted result.

Output Format:

1. AI-generated unittest test cases

2. Implementation of convert_date_format(date_str)

3. Main block to run tests

4. Printed output showing conversion results

Code:

```
"""2303A52324"""
import unittest
def convert_date_format(date_str):
    # Check if input is string
    if not isinstance(date_str, str):
        return "Invalid Date Format"
    # Check format length
    if len(date_str) != 10:
        return "Invalid Date Format"
    # Check structure YYYY-MM-DD
    parts = date_str.split("-")
    if len(parts) != 3:
        return "Invalid Date Format"
    year, month, day = parts
    # Ensure numeric values
    if not (year.isdigit() and month.isdigit() and day.isdigit()):
        return "Invalid Date Format"
    # Validate month and day range
    month = int(month)
    day = int(day)
    if not (1 <= month <= 12 and 1 <= day <= 31):
        return "Invalid Date Format"
    # Return converted format
    return f"{day:02d}-{month:02d}-{year}"
class TestDateConversion(unittest.TestCase):
```

Output:

```
PS D:\3-2\AI Assitant Coding>  d:; cd 'd:\3-2\AI Assitant Coding';
thon313\python.exe' 'c:\Users\SNEHITH\.vscode\extensions\ms-python.
ncher' '55403' '--' 'D:\3-2\AI Assitant Coding\Task-8.3.py'
15-10-2023 --> Invalid Date Format
2023/10/15 --> Invalid Date Format
2023-13-01 --> Invalid Date Format
2023-10-40 --> Invalid Date Format
20231015 --> Invalid Date Format
20231015 --> Invalid Date Format
.2023-10-15 --> 15-10-2023
2000-01-01 --> 01-01-2000
1999-12-31 --> 31-12-1999
.
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
PS D:\3-2\AI Assitant Coding>
```