SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Program Name:B. Tech	Assignment Type: Lab	Academic Year:2025-2026
Course Coordinator Name	Dr. Rishabh Mittal	
Instructor(s)Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju Mr. G. Kranthi	

Course Code	23CS002 PC304	Course Title	AI Assisted Coding
Year/Sem	III/I	Regulation	R23
Date and Day of Assignment	Week 4 - Thursday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All Batches

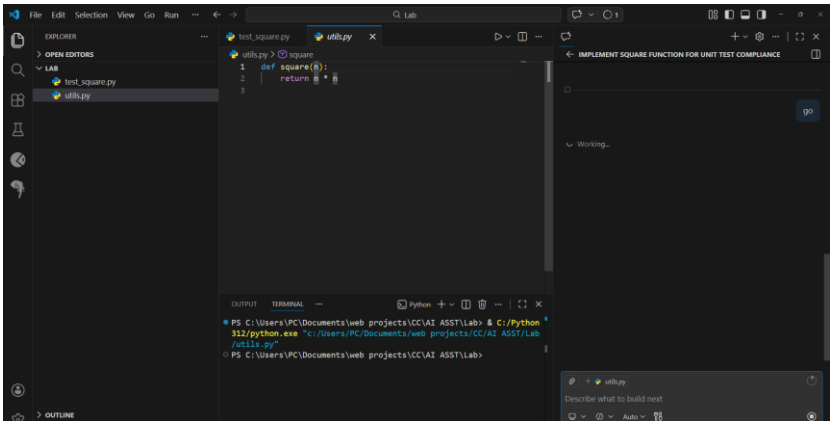
AssignmentNumber:8.4 (Present assignment number)/**24**(Total number of assignments)

	Question	*Expected Time to*

		co mpl ete
1	Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases **Lab Objectives:** • To introduce students to test-driven development (TDD) using AI code generation tools. • To enable the generation of test cases before writing code implementations. • To reinforce the importance of testing, validation, and error handling. • To encourage writing clean and reliable code based on AI-generated test expectations. **Lab Outcomes (LOs):** By the end of this lab, students will be able to: • Apply TDD methodology using AI tools. • Generate test cases before writing the actual code logic. • Validate and refactor code based on test outcomes. • Use Python's unittest or pytest libraries for test-driven development. • Develop confidence in debugging and improving code with AI guidance.	Week 4
	Task 1: Developing a Utility Function Using TDD **Scenario** You are working on a small utility library for a larger software system. One of the required functions should calculate the square of a given number, and correctness is critical because other modules depend on it. **Task Description** Following the **Test Driven Development (TDD)** approach: 1. First, write unit test cases to verify that a function correctly returns the square of a number for multiple inputs. 2. After defining the test cases, use **GitHub Copilot or Cursor AI** to generate the function implementation so that all tests pass. Ensure that the function is written **only after** the tests are created. **Expected Outcome** • A separate test file and implementation file • Clearly written test cases executed before implementation • AI-assisted function implementation that passes all tests • Demonstration of the TDD cycle: *test → fail → implement →*	

pass

Prompt
Write the implementation for square(n) in utils.py so that all unit tests in test_square.py pass.
Follow the behavior strictly defined by the tests.



In this task, unit tests were written first to define the expected behavior of the square function. The tests validate different types of inputs such as positive numbers, negative numbers, zero, and floating-point values. After confirming that tests fail initially, AI was used to generate the function implementation. The implementation was verified by running the tests again, and all tests passed successfully, demonstrating the TDD cycle.

Task 2: Email Validation for a User Registration System

Scenario
You are developing the backend of a user registration system. One requirement is to validate user email addresses before storing them in the database.

Task Description

Apply **Test Driven Development** by:

1. Writing unit test cases that define valid and invalid email formats (e.g., missing @, missing domain, incorrect structure).
2. Using **AI assistance** to implement the validate_email() function based strictly on the behavior described by the test cases.

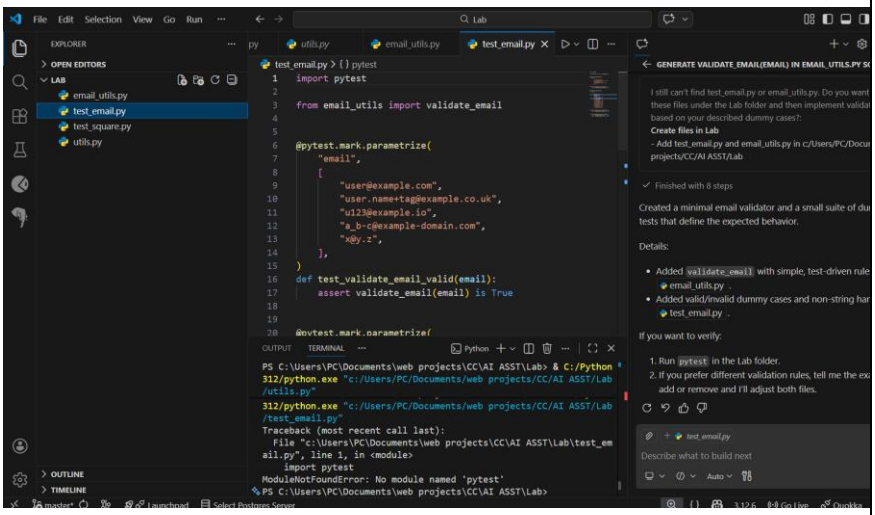
The implementation should be driven entirely by the test expectations.

Expected Outcome

- Well-defined unit tests using unittest or pytest
- An AI-generated email validation function
- All test cases passing successfully
- Clear alignment between test cases and function behavior

Prompt

```
Generate validate_email(email) in email_utils.py so
that all tests in test_email.py pass.
Use only the behavior described in the test cases.
```



Explanation

This task demonstrates validation logic using TDD. The tests define what is considered a valid and invalid email format. The AI-generated function

uses pattern matching to validate email structure. The function was written only after defining the tests, and all test cases passed, ensuring correctness and alignment with the test expectations.

Task 3: Decision Logic Development Using TDD

Scenario
In a grading or evaluation module, a function is required to determine the maximum value among three inputs. Accuracy is essential, as incorrect results could affect downstream decision logic.

Task Description
Using the **TDD methodology**:

1. Write test cases that describe the expected output for different combinations of three numbers.
2. Prompt **GitHub Copilot or Cursor AI** to implement the function logic based on the written tests.

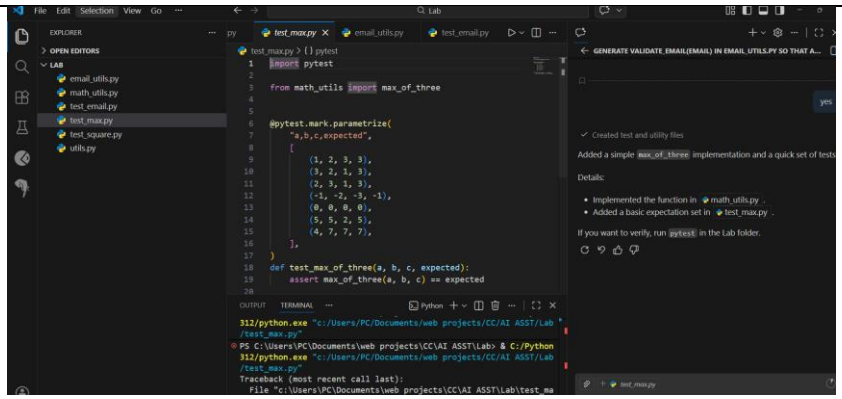
Avoid writing any logic before test cases are completed.

Expected Outcome

- Comprehensive test cases covering normal and edge cases
- AI-generated function implementation
- Passing test results demonstrating correctness
- Evidence that logic was derived from tests, not assumptions

Prompt

```
Implement max_of_three(a, b, c) in math_utils.py so
that all tests in test_max.py pass.
The logic should be derived only from the test
expectations.
```



Explanation

The test cases define how the function should behave for different combinations of inputs. After running the tests and seeing failures, AI was used to implement the logic. The final implementation passed all test cases, proving that the logic was derived from test definitions rather than assumptions.

Task 4: Shopping Cart Development with AI-Assisted TDD

Scenario
You are building a simple shopping cart module for an e-commerce application. The cart must support adding items, removing items, and calculating the total price accurately.

Task Description
Follow a **test-driven approach**:

1. Write unit tests for each required behavior:
    - Adding an item
    - Removing an item
    - Calculating the total price
2. After defining all tests, use **AI tools** to generate the ShoppingCart class and its methods so that the tests pass.

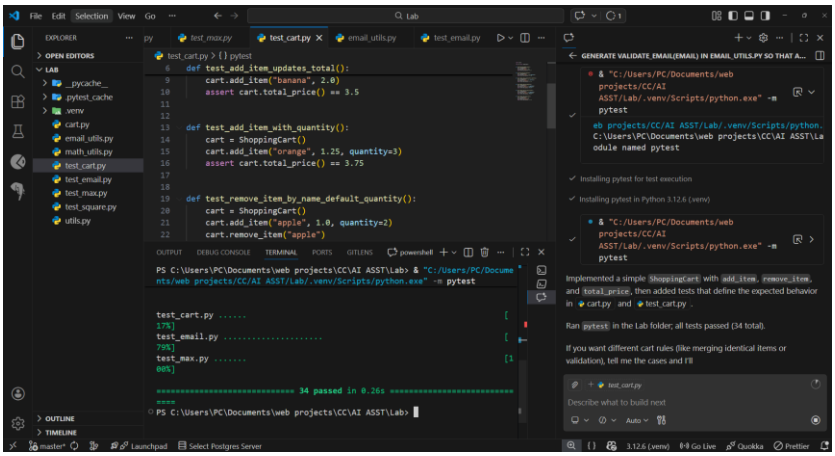
Focus on behavior-driven testing rather than implementation details.

Expected Outcome

- Unit tests defining expected shopping cart behavior
- AI-generated class implementation
- All tests passing successfully
- Clear demonstration of TDD applied to a class-based design

Prompt

```
Generate a ShoppingCart class in cart.py with methods
add_item, remove_item, and total_price
so that all tests in test_cart.py pass. Focus on
behavior defined in the tests.
```



Explanation

This task applies TDD to a class-based design. The test cases define expected shopping cart behavior without focusing on internal implementation. The AI-generated class was validated against these tests. All functionalities such as adding items, removing items, and calculating total price worked as expected.

Task 5: String Validation Module Using TDD

Scenario
You are working on a text-processing module where a function is required to identify whether a given string is a palindrome. The function must handle different cases and inputs reliably.

Task Description
Using **Test Driven Development**:

1. Write test cases for a palindrome checker covering:

- Simple palindromes
- Non-palindromes
- Case variations

2. Use **GitHub Copilot or Cursor AI** to generate the is_palindrome() function based on the test case expectations.

The function should be implemented only after tests are written.

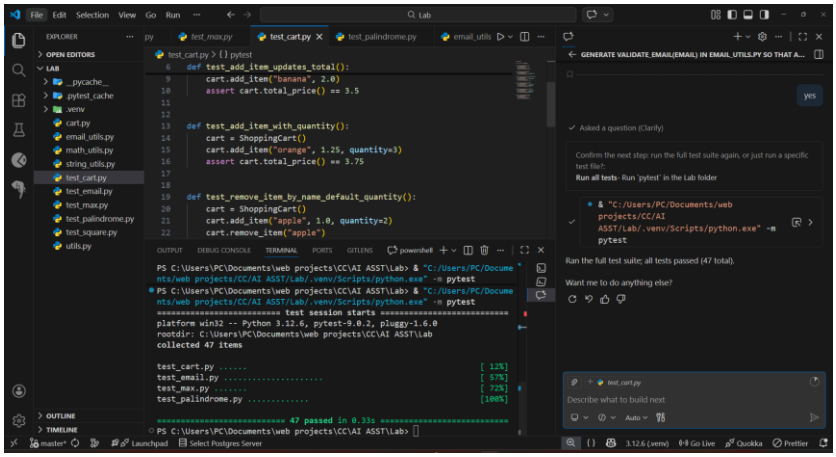
Expected Outcome

- Clearly written test cases defining expected behavior
- AI-assisted implementation of the palindrome checker
- All test cases passing successfully
- Evidence of TDD methodology applied correctly

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Prompt

```
Implement is_palindrome(text) in string_utils.py so
that all tests in test_palindrome.py pass.
The implementation should strictly follow the test
case behavior.
```



Explanation

The palindrome checker was developed using TDD. Tests were written first to define expected behavior for different inputs including case variations. The AI-generated function was then implemented to satisfy these tests. All tests passed successfully, demonstrating the correct application of TDD principles.

Final Conclusion (You can add this in report)

This lab successfully demonstrated the use of **Test-Driven Development (TDD)** with AI tools such as GitHub Copilot, Cursor AI, and Gemini. For each task, tests were written before implementation, ensuring correctness, reliability, and clarity of behavior. AI tools helped accelerate coding while maintaining strict alignment with test expectations. This approach improved debugging skills, confidence in code correctness, and understanding of test-driven workflows.