

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		<table border="1"> <tr><td>Mr. S Naresh Kumar</td><td></td></tr> <tr><td>Ms. B. Swathi</td><td></td></tr> <tr><td>Dr. Sasanko Shekhar Gantayat</td><td></td></tr> <tr><td>Mr. Md Sallauddin</td><td></td></tr> <tr><td>Dr. Mathivanan</td><td></td></tr> <tr><td>Mr. Y Srikanth</td><td></td></tr> <tr><td>Ms. N Shilpa</td><td></td></tr> <tr><td>Dr. Rishabh Mittal (Coordinator)</td><td></td></tr> <tr><td>Dr. R. Prashant Kumar</td><td></td></tr> <tr><td>Mr. Ankushavali MD</td><td></td></tr> <tr><td>Mr. B Viswanath</td><td></td></tr> <tr><td>Ms. Sujitha Reddy</td><td></td></tr> <tr><td>Ms. A. Anitha</td><td></td></tr> <tr><td>Ms. M. Madhuri</td><td></td></tr> <tr><td>Ms. Katherashala Swetha</td><td></td></tr> <tr><td>Ms. Velpula sumalatha</td><td></td></tr> <tr><td>Mr. Bingi Raju</td><td></td></tr> <tr><td>Mr. G. Kranthi</td><td></td></tr> </table>		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	
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																																				
Assignment Number: 8.4 (Present assignment number)/24 (Total number of assignments)																																							
	Question		Expected Time to																																				

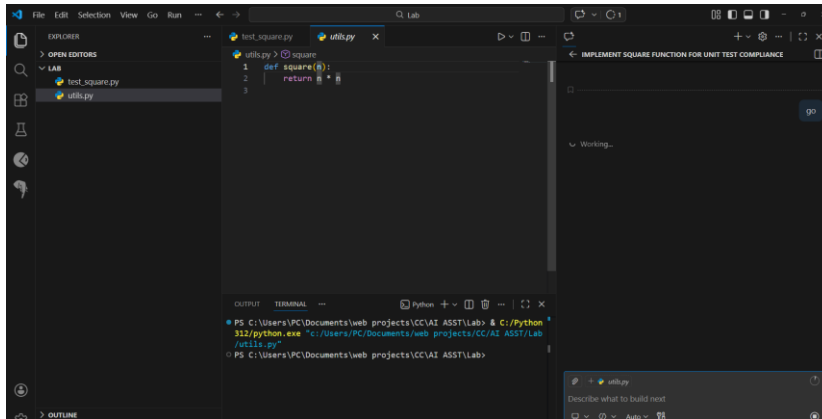
		complete	
1	<p>Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases</p> <p>Lab Objectives:</p> <ul style="list-style-type: none"> To introduce students to test-driven development (TDD) using AI code generation tools. To enable the generation of test cases before writing code implementations. To reinforce the importance of testing, validation, and error handling. To encourage writing clean and reliable code based on AI-generated test expectations. <p>Lab Outcomes (LOs):</p> <p>By the end of this lab, students will be able to:</p> <ul style="list-style-type: none"> 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	
	<p>Task 1: Developing a Utility Function Using TDD</p> <p>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.</p> <p>Task Description Following the Test Driven Development (TDD) approach:</p> <ol style="list-style-type: none"> First, write unit test cases to verify that a function correctly returns the square of a number for multiple inputs. After defining the test cases, use GitHub Copilot or Cursor AI to generate the function implementation so that all tests pass. <p>Ensure that the function is written only after the tests are created.</p> <p>Expected Outcome</p> <ul style="list-style-type: none"> 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: <i>test</i> → <i>fail</i> → <i>implement</i> → 		

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.

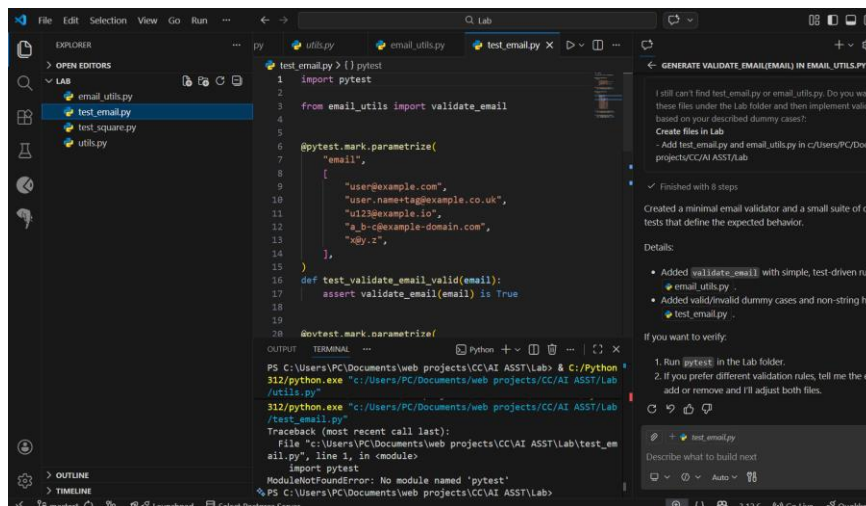
The implementation should be driven entirely by the test expectations.

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.



The screenshot shows a VS Code editor with a file explorer on the left showing a project named 'LAB'. The main editor window displays a file named 'test_email.py' with the following content:

```
1 import pytest
2
3 from email_utils import validate_email
4
5
6 @pytest.mark.parametrize(
7     "email",
8     [
9         "user@example.com",
10        "user.name+tag@example.co.uk",
11        "u123@example.io",
12        "a_b-c@example-domain.com",
13        "x@y.z",
14    ],
15 )
16 def test_validate_email_valid(email):
17     assert validate_email(email) is True
18
19
20 @pytest.mark.parametrize(
21     "email",
22     [
23         "user@example.com",
24         "user.name+tag@example.co.uk",
25         "u123@example.io",
26         "a_b-c@example-domain.com",
27         "x@y.z",
28     ],
29 )
30 def test_validate_email_invalid(email):
31     assert validate_email(email) is False
```

The terminal window at the bottom shows the command `python -m pytest` being executed, resulting in a `ModuleNotFoundError: No module named 'pytest'`. The right sidebar shows a chat window with the 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." The chat response states: "I still can't find test_email.py or email_utils.py. Do you want these files under the Lab folder and then implement validate_email based on your described dummy cases?" It then provides instructions to create files in the 'LAB' folder and lists the steps: 1. Run `pytest` in the Lab folder. 2. If you prefer different validation rules, tell me the expected behavior and I'll adjust both files.

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.

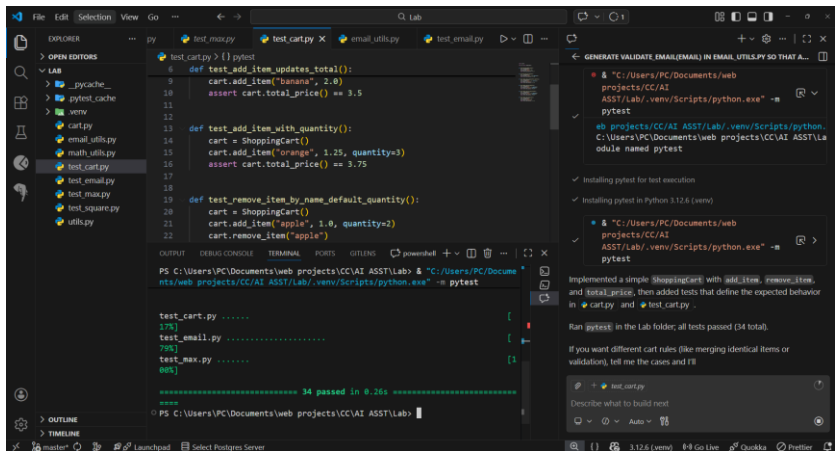
Focus on behavior-driven testing rather than implementation details.

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.



```
def test_add_item_updates_total():
    cart.add_item("banana", 2.0)
    assert cart.total_price() == 3.5

def test_add_item_with_quantity():
    cart = ShoppingCart()
    cart.add_item("orange", 1.25, quantity=3)
    assert cart.total_price() == 3.75

def test_remove_item_by_name_default_quantity():
    cart = ShoppingCart()
    cart.add_item("apple", 1.0, quantity=2)
    cart.remove_item("apple")
```

```
test_cart.py ..... [17s]
test_email.py ..... [79s]
test_max.py ..... [1s]

===== 34 passed in 0.26s =====
```

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.

The screenshot shows a code editor with the following content:

```
test_cart.py > {} pytest
6 def test_add_item_updates_total():
7     cart.add_item("banana", 2.0)
8     assert cart.total_price() == 3.5
9
10
11
12
13 def test_add_item_with_quantity():
14     cart = ShoppingCart()
15     cart.add_item("orange", 1.25, quantity=3)
16     assert cart.total_price() == 3.75
17
18
19
20 def test_remove_item_by_name_default_quantity():
21     cart = ShoppingCart()
22     cart.add_item("apple", 1.0, quantity=2)
23     cart.remove_item("apple")
```

The terminal output shows the following:

```
PS C:\Users\PC\Documents\web projects\CCVAI ASST\Lab> & "C:/Users/PC/Docu
nts/web projects/CCVAI ASST/lab/.venv/scripts/python.exe" -m pytest
* PS C:\Users\PC\Documents\web projects\CCVAI ASST\Lab> & "C:/Users/PC/Docu
nts/web projects/CCVAI ASST/lab/.venv/scripts/python.exe" -m pytest
===== Test session starts =====
platform win32 -- Python 3.12.6, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\PC\Documents\web projects\CCVAI ASST\Lab
collected 47 items

test_cart.py ..... [ 12%]
test_email.py ..... [ 37%]
test_max.py ..... [ 72%]
test_palindrome.py ..... [100%]

===== 47 passed in 0.33s =====
PS C:\Users\PC\Documents\web projects\CCVAI ASST\Lab>
```


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.