NAME:CH.Kruthankiran H.NO:2303A51404 BATCH:26

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	23CS002PC304	**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)			

Q.No.	Question	*ExpectedTime to complete*
1	Lab 8: Test-Driven Development with AI – Generating and Working with Test	Week 4

	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.	
	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*	

AI 8.4.py > ...

```python
import unittest

class TestSquare(unittest.TestCase):

    def test_square_positive(self):
        self.assertEqual(square(2), 4)

    def test_square_negative(self):
        self.assertEqual(square(-3), 9)

    def test_square_zero(self):
        self.assertEqual(square(0), 0)

    def test_square_float(self):
        self.assertEqual(square(1.5), 2.25)


def square(n):
    return n ** 2


if __name__ == "__main__":
    unittest.main()

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS
PS C:\Users\kruth\OneDrive\Desktop\java>  & 'c:\Users\kruth
2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65328'
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
PS C:\Users\kruth\OneDrive\Desktop\java>
```

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

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	

AI 8.4(ii).py > ...

```python
import unittest

class TestMaxOfThree(unittest.TestCase):

    def test_max_all_positive(self):
        self.assertEqual(max_of_three(1, 2, 3), 3)

    def test_max_with_negatives(self):
        self.assertEqual(max_of_three(-1, -2, -3), -1)

    def test_max_mixed(self):
        self.assertEqual(max_of_three(-1, 5, 0), 5)

    def test_max_duplicates(self):
        self.assertEqual(max_of_three(2, 2, 2), 2)

    def test_max_first_max(self):
        self.assertEqual(max_of_three(10, 5, 7), 10)


def max_of_three(a, b, c):
    return max(a, b, c)


if __name__ == "__main__":
    unittest.main()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Users\kruth\OneDrive\Desktop\java>  & 'c:\Users\kruth\AppDa
2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65328' '--' '
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\
ers\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x6
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\
ers\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x6
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
PS C:\Users\kruth\OneDrive\Desktop\java>
```

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: o Adding an item o Removing an item o 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	

AI 8.4(iii).py > ShoppingCart > total_price

```python
import unittest
class TestShoppingCart(unittest.TestCase):
    def test_add_item(self):
        cart = ShoppingCart()
        cart.add_item("apple", 2, 3.0)   # 2 apples at $3.0 each
        self.assertEqual(cart.items["apple"]["quantity"], 2)
        self.assertAlmostEqual(cart.items["apple"]["price"], 3.0)
    def test_remove_item(self):
        cart = ShoppingCart()
        cart.add_item("banana", 1, 1.5)
        cart.remove_item("banana")
        self.assertNotIn("banana", cart.items)
    def test_remove_nonexistent_item(self):
        cart = ShoppingCart()
        # removing an item not in cart should do nothing, not crash
        cart.remove_item("ghost")
        self.assertEqual(len(cart.items), 0)
    def test_total_price_empty_cart(self):
        cart = ShoppingCart()
        self.assertEqual(cart.total_price(), 0.0)
    def test_total_price_multiple_items(self):
        cart = ShoppingCart()
        cart.add_item("apple", 2, 3.0)   # 6.0
        cart.add_item("banana", 3, 1.5)  # 4.5
        self.assertAlmostEqual(cart.total_price(), 10.5)
class ShoppingCart:
    def __init__(self):
        self.items = {}
    def add_item(self, name, quantity, price):
        if name in self.items:
            self.items[name]["quantity"] += quantity
        else:
            self.items[name] = {"quantity": quantity, "price": price}
    def remove_item(self, name):
        if name in self.items:
            del self.items[name]
    def total_price(self):
        total = 0.0
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
ers\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\OneDrive\Desktop\java';
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62031' '--' 'c:\Users\kruth\OneDrive\
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\OneDrive\Desktop\java';
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65090' '--' 'c:\Users\kruth\OneDrive\

.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
PS C:\Users\kruth\OneDrive\Desktop\java>
```

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: o Simple palindromes o Non-palindromes o 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	

AI 8.4(iv).py > ...

```python
import unittest

class TestPalindrome(unittest.TestCase):

    def test_simple_palindrome(self):
        self.assertTrue(is_palindrome("madam"))

    def test_non_palindrome(self):
        self.assertFalse(is_palindrome("hello"))

    def test_case_variation(self):
        self.assertTrue(is_palindrome("RaceCar"))

    def test_single_character(self):
        self.assertTrue(is_palindrome("a"))

    def test_empty_string(self):
        self.assertTrue(is_palindrome(""))


def is_palindrome(text):
    text = text.lower()
    return text == text[::-1]


if __name__ == "__main__":
    unittest.main()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\deb
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\
extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\deb

.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
PS C:\Users\kruth\OneDrive\Desktop\java>  c:; cd 'c:\Users\kruth\
extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\deb
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
PS C:\Users\kruth\OneDrive\Desktop\java>
```

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