SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName:B. Tech		**Assignment Type: Lab**	**AcademicYear:**2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
Instructor(s)Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week4 - Wednesday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:8.3(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 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.	Week4 - Wednesday

- 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):
After completing this lab, students will be able to:

- Use AI tools to write test cases for Python functions and classes.
- Implement functions based on test cases in a test-first development style.
- Use unittest or pytest to validate code correctness.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic

Task Description#1

Use AI to generate test cases for is_valid_email(email) and then implement the validator function.

Requirements:

- Must contain @ and . characters.
- Must not start or end with special characters.
- Should not allow multiple @.

Expected Output#1

- Email validation logic passing all test cases

PROMPT: write the python function that valid email(email) and then implement the validator function must contain@ and characters.Must not start or end with special characters.Should not allow multiple @.

CODE :

```
import re
def valid_email(email):
    # Check for exactly one @
    if email.count('@') != 1:
        return False
    # Check for allowed characters and structure
    pattern = r'^[A-Za-z0-9][A-Za-z0-9._%+-]*@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
    if not re.match(pattern, email):
        return False
    # Should not start or end with special characters
    if not (email[0].isalnum() and email[-1].isalnum()):
        return False
    return True
if __name__ == "__main__":
    emails = [
        # Test case1
        "Deeksha123.@gmail.com", #returns true
        # Test case2
        ".deeksha123.@gmail.com" #returns false
    ]
    for e in emails:
        print(f"{e}: {valid_email(e)}")
```

OUTPUT:

```
IAC/Lab-8/Task1.py
Deeksha123.@gmail.com: True
.deeksha123.@gmail.com: False
```

TEST CASES FORM VS:

Test_Task1.py > ...

```python
import unittest
from Task1 import valid_email

class TestValidEmail(unittest.TestCase):
    def test_valid_email_simple(self):
        email = "user@example.com"
        result = valid_email(email)
        print(f"{email}: {result}")
        self.assertTrue(result)

    def test_valid_email_with_numbers(self):
        email = "user123@example.com"
        result = valid_email(email)
        print(f"{email}: {result}")
        self.assertTrue(result)

    def test_valid_email_with_dot(self):
        email = "first.last@example.co.uk"
        result = valid_email(email)
        print(f"{email}: {result}")
        self.assertTrue(result)

    def test_valid_email_with_underscore(self):
        email = "first_last@example.org"
        result = valid_email(email)
        print(f"{email}: {result}")
```

OUTPUT:

```
user@example.com: True
.first.last@example.co.uk: True
.user-name@example.com: True
.user@example.technology: True
.user123@example.com: True
```

```
.first.last@example.co.uk: True
.user-name@example.com: True
.user@example.technology: True
.user123@example.com: True
.user+tag@example.com: True
.user@mail.example.com: True
.first_last@example.org: True
```

```
----------------------------
Ran 8 tests in 0.001s

OK
```

Task Description#2 (Loops)

- Ask AI to generate test cases for assign_grade(score) function. Handle boundary and invalid inputs.

Requirements

- AI should generate test cases for assign_grade(score) where: 90-100: A, 80-89: B, 70-79: C, 60-69: D, <60: F
- Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

Expected Output#2

Grade assignment function passing test suite

PROMPT: write the python function assign_grade(score).handle boundary and invalid inputs.

CODE:

```
def assign_grade(score):
    if not isinstance(score, (int, float)):
        return "Invalid input: score must be a number."
    if score < 0 or score > 100:
        return "Invalid input: score must be between 0 and 100."

    if score >= 90:
        return 'Grade A'
    elif score >= 80:
        return 'Grade B'
    elif score >= 70:
        return 'Grade C'
    elif score >= 60:
        return 'Grade D'
    else:
        return 'Fail'
# test cases 1:
print(assign_grade(97)) #returns 'A'
# test cases 2:
print(assign_grade(85)) #returns 'B'
# test cases 3:
print(assign_grade(55)) #returns 'F'
```

OUTPUT:

```
eDrive/Desktop/AIAC/Lab-8/Task2.py
Grade A
Grade B
Fail
```

TEST CASES FROM VS:

```python
import unittest
from Task2 import assign_grade

class TestAssignGrade(unittest.TestCase):
    def test_grade_A(self):
        for score in [ 97]:
            result = assign_grade(score)
            print(f"assign_grade({score}) = {result}")
            self.assertEqual(result, 'Grade A')

    def test_grade_B(self):
        for score in [89]:
            result = assign_grade(score)
            print(f"assign_grade({score}) = {result}")
            self.assertEqual(result, 'Grade B')

    def test_grade_C(self):
        for score in [ 75]:
            result = assign_grade(score)
            print(f"assign_grade({score}) = {result}")
            self.assertEqual(result, 'Grade C')

    def test_grade_D(self):
        for score in [66]:
            result = assign_grade(score)
            print(f"assign_grade({score}) = {result}")
            self.assertEqual(result, 'Grade D')

    def test fail(self):
```

```
    def test_fail(self):
        for score in [55]:
            result = assign_grade(score)
            print(f"assign_grade({score}) = {result}")
            self.assertEqual(result, 'Fail')

    def test_invalid_negative(self):
        result = assign_grade(-5)
        print(f"assign_grade(-5) = {result}")
        self.assertEqual(result, "Invalid input: score must be between 0 and 100.")

    def test_invalid_above_100(self):
        result = assign_grade(105)
        print(f"assign_grade(105) = {result}")
        self.assertEqual(result, "Invalid input: score must be between 0 and 100.")

    def test_invalid_string(self):
        result = assign_grade("eighty")
        print(f"assign_grade('eighty') = {result}")
        self.assertEqual(result, "Invalid input: score must be a number.")

    def test_invalid_none(self):
        result = assign_grade(None)
        print(f"assign_grade(None) = {result}")
        self.assertEqual(result, "Invalid input: score must be a number.")

if __name__ == '__main__':
    unittest.main()
```

OUTPUT:

```
assign_grade(55) = Fail
.assign_grade(97) = Grade A
.assign_grade(89) = Grade B
.assign_grade(75) = Grade C
.assign_grade(66) = Grade D
.assign_grade(105) = Invalid input: score must be between 0 and 100.

.assign_grade(-5) = Invalid input: score must be between 0 and 100.
.assign_grade(None) = Invalid input: score must be a number.
.assign_grade('eighty') = Invalid input: score must be a number.
.
----------------------------------------------------------------------
Ran 9 tests in 0.001s

OK
```

Task Description#3

- Generate test cases using AI for is_sentence_palindrome(sentence). Ignore case, punctuation, and spaces

Requirement

- Ask AI to create test cases for is_sentence_palindrome(sentence) (ignores case, spaces, and punctuation).

- Example:
  "A man a plan a canal Panama" → True

Expected Output#3

- Function returns True/False for cleaned sentences
- Implement the function to pass AI-generated tests.

- PROMPT : write the python function is_sentence_palindrome(sentence). Ignore case, punctuation, and spaces.

CODE:

```
import string

def is_sentence_palindrome(sentence):
    # Remove punctuation and spaces, convert to lowercase
    cleaned = ''.join(
        ch.lower() for ch in sentence if ch.isalnum()
    )
    return cleaned == cleaned[::-1]
#test cases1:
#"A man, a plan, a canal, Panama" #is a palindrome
#test cases2:
# Was it a car or a cat I saw? #is not a palindrome
sentence = input("Enter a sentence: ")
if is_sentence_palindrome(sentence):
    print("The sentence is a palindrome.")
else:
    print("The sentence is not a palindrome.")
```

OUTPUT:

```
Enter a sentence: "A man, a plan,a canal, panama"
The sentence is a palindrome.
```

```
Enter a sentence: was it a car or cat i saw?
The sentence is not a palindrome.
```

TEST CASES FROM VS:

```python
import unittest
from Task3 import is_sentence_palindrome

class TestIsSentencePalindrome(unittest.TestCase):
    def test_simple_palindrome(self):
        self.assertTrue(is_sentence_palindrome("madam"))
        print("test_simple_palindrome")

    def test_sentence_palindrome(self):
        self.assertTrue(is_sentence_palindrome("A man a plan a canal Panama"))
        print("test_sentence_palindrome")

    def test_sentence_with_punctuation(self):
        self.assertTrue(is_sentence_palindrome("Was it a car or a cat I saw?"))
        print("test_sentence_with_punctuation")

    def test_sentence_with_mixed_case(self):
        self.assertTrue(is_sentence_palindrome("No lemon, no melon"))
        print("test_sentence_with_mixed_case")

    def test_not_palindrome(self):
        self.assertFalse(is_sentence_palindrome("This is not a palindrome"))
        print("test_not_palindrome")

    def test_empty_string(self):
        self.assertTrue(is_sentence_palindrome(""))
        print("test_empty_string")

    def test_single_character(self):
        self.assertTrue(is_sentence_palindrome("x"))
```


```python
        print("test_empty_string")

    def test_single_character(self):
        self.assertTrue(is_sentence_palindrome("x"))
        print("test_single_character")
```


```python
if __name__ == '__main__':
    unittest.main()
```

OUTPUT:

```
Enter a sentence: "A man a paln a canal panama"
The sentence is not a palindrome.
test_empty_string
.test_not_palindrome
.test_sentence_palindrome
.test_sentence_with_mixed_case
.test_sentence_with_punctuation
.test_simple_palindrome
.test_single_character
.
```

```
.
----------------------------------
Ran 7 tests in 0.002s

OK
```

Task Description#4

- Let AI fix it Prompt AI to generate test cases for a ShoppingCart class (add_item, remove_item, total_cost).

  Methods:
  Add_item(name,orice)
  Remove_item(name)
  Total_cost()

Expected Output#4

- Full class with tested functionalities

PROMPT: write the python function ShoppingCart class (add_item, remove_item, total_cost)
Add_item(name,orice)
Remove_item(name)
Total_cost().

CODE:

```
class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, name, price):
        if name in self.items:
            self.items[name]['quantity'] += 1
        else:
            self.items[name] = {'price': price, 'quantity': 1}

    def remove_item(self, name):
        if name in self.items:
            if self.items[name]['quantity'] > 1:
                self.items[name]['quantity'] -= 1
            else:
                del self.items[name]

    def total_cost(self):
        return sum(item['price'] * item['quantity'] for item in self.items.values())
# Example usage:
cart = ShoppingCart()
#test case1:
cart.add_item("apple", 1.0)
print(cart.total_cost())  # Output: 1.0
#test case2:
cart.add_item("banana", 0.5)
print(cart.total_cost())  # Output: 1.5
```

OUTPUT:

```
eDrive/Desktop/AIAC/Lab-8/Task4.py
1.0
1.5
```

TEST CASES FROM VSC:

```python
from Task4 import ShoppingCart

def test_cart_initially_empty():
    cart = ShoppingCart()
    assert cart.total_cost() == 0

def test_add_single_item():
    cart = ShoppingCart()
    cart.add_item("apple", 1.0)
    assert cart.total_cost() == 1.0
    assert cart.items["apple"]["quantity"] == 1

def test_add_multiple_items():
    cart = ShoppingCart()
    cart.add_item("apple", 1.0)
    cart.add_item("banana", 0.5)
    assert cart.total_cost() == 1.5
    assert cart.items["banana"]["quantity"] == 1

def test_add_same_item_multiple_times():
    cart = ShoppingCart()
    cart.add_item("apple", 1.0)
    cart.add_item("apple", 1.0)
    assert cart.items["apple"]["quantity"] == 2
    assert cart.total_cost() == 2.0

def test_remove_item_decreases_quantity():
    cart = ShoppingCart()
    cart.add_item("apple", 1.0)
    cart.add_item("apple", 1.0)
    cart.remove_item("apple")
    assert cart.items["apple"]["quantity"] == 1
    assert cart.total_cost() == 1.0

def test_remove_item_removes_when_quantity_one():
    cart = ShoppingCart()
```

```
    cart.add_item("apple", 1.0)
    cart.remove_item("apple")
    assert "apple" not in cart.items
    assert cart.total_cost() == 0

def test_remove_item_not_in_cart():
    cart = ShoppingCart()
    cart.remove_item("banana")  # Should not raise
    assert cart.total_cost() == 0
```

Task Description#5

- Use AI to write test cases for convert_date_format(date_str) to switch from "YYYY-MM-DD" to "DD-MM-YYYY".
  Example: "2023-10-15" → "15-10-2023"

Expected Output#5

- Function converts input format correctly for all test cases

PROMPT: Write the python function for convert_date_format(date_str) to switch from "YYYY-MM-DD" to "DD-MM-YYYY".

CODE:

Task5.py > ...

```
1   def convert_date_format(date_str):
2       parts = date_str.split('-')
3       if len(parts) != 3:
4           raise ValueError("Input date must be in 'YYYY-MM-DD' format.")
5       year, month, day = parts
6       return f"{day}-{month}-{year}"
7   # Test cases1:
8   print(convert_date_format("2023-10-05"))  # Output: "05-10-2023"
9   # Test cases2:
0   print(convert_date_format("1999-01-15"))  # Output: "15-01-1999"
```

OUTPUT:

```
05-10-2023
15-01-1999
```

TEST CASES FROM VSC:

```python
from Task5 import convert_date_format
import pytest

def test_valid_date():
    result1 = convert_date_format("1999-01-24")
    result2 = convert_date_format("2000-12-31")
    print(result1)
    print(result2)
    assert result1 == "24-01-1999"
    assert result2 == "31-12-2000"

def test_invalid_format_missing_parts():
    with pytest.raises(ValueError):
        convert_date_format("2000-12")
    with pytest.raises(ValueError):
        convert_date_format("2000")

def test_invalid_format_extra_parts():
    with pytest.raises(ValueError):
        convert_date_format("2000-12-31-01")

def test_invalid_format_wrong_separator():
    with pytest.raises(ValueError):
        convert_date_format("2000/12/05")
```

OUTPUT:
1999-01-24
2000-12-31

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

Evaluation Criteria:

Criteria	Max Marks
Task #1	0.5
Task #2	0.5
Task #3	0.5
Task #4	0.5
Task #5	0.5
Total	**2.5 Marks**