SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type : Lab**	**Academic Year :** 2025-2026
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week3 – Monday**	**Batch**	23CSBTB47B
Name	Pushpanjali	**Hall Ticket No**	2303A54050

Assignment Number: 5.1

Q.No.	Question

Lab 5: Ethical Foundations – Responsible AI Coding Practices

Lab Objectives:

- To explore the ethical risks associated with AI-generated code.
- To recognize issues related to security, bias, transparency, and copyright.
- To reflect on the responsibilities of developers when using AI tools in software development.
- To promote awareness of best practices for responsible and ethical AI coding.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Identify and avoid insecure coding patterns generated by AI tools.
- Detect and analyze potential bias or discriminatory logic in AI-generated outputs.
- Evaluate originality and licensing concerns in reused AI-generated code.
- Understand the importance of explainability and transparency in AI-assisted programming.
- Reflect on accountability and the human role in ethical AI coding practices.

Task Description #1 (Privacy in API Usage)

Task: Use an AI tool to generate a Python program that connects to a weather API.

Prompt:

"Generate code to fetch weather data securely without exposing API keys in the code."

Expected Output:

- Original AI code (check if keys are hardcoded).
- Secure version using environment variables.

lab_5.py > lab 5.py > ...

```python
#TASK 1:Generate code to fetch weather data securely without exposing API keys in the code.

import os
import requests
from dotenv import load_dotenv

load_dotenv()
API_KEY = os.getenv("WEATHER_API_KEY")

Qodo: Test this function
def get_weather(city):
    base_url = "https://api.openweathermap.org/data/2.5/weather"
    params = {
        "q": city,
        "appid": API_KEY,
        "units": "metric"
    }

    response = requests.get(base_url, params=params)
    data = response.json()

    if response.status_code == 200:
        weather = data["weather"][0]["description"]
        temperature = data["main"]["temp"]
        return f"Weather in {city}: {weather}, Temperature: {temperature}°C"
    else:
        return f"API Error {response.status_code}: {data.get('message', data)}"


if __name__ == "__main__":
    city = input("Enter city name: ")
    print(get_weather(city))

#Task 2:
```

Output:

```
PS C:\Users\srina\OneDrive\Desktop\aiac> & C:/Users/srina/AppData/Local/Programs/Python/Python313/python.exe
Enter city name: warangal
Weather in warangal: clear sky, Temperature: 30.67°C
PS C:\Users\srina\OneDrive\Desktop\aiac>
```

Task Description #2 (Privacy & Security in File Handling)

Task: Use an AI tool to generate a Python script that stores user data (name, email, password) in a file.

Analyze: Check if the AI stores sensitive data in plain text or without encryption.

Expected Output:

- Identified privacy risks.
- Revised version with encrypted password storage (e.g., hashing).

Prompt :

Generate a Python script that stores user details (name, email, password) in a file. Then analyze if storing the password is secure and modify the script to store the password safely using hashing instead of plain text.

Code :

```python
#Task 2:
#Generate a Python script that stores user details (name, email, password) in a file. Then analyze if storing the password is secure and modify the script to store the password
import os
import hashlib
Qodo: Test this function
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
Qodo: Test this function
def store_user_details(name, email, password):
    hashed_password = hash_password(password)
    with open("user_details.txt", "a") as file:
        file.write(f"Name: {name}, Email: {email}, Password: {hashed_password}\n")
if __name__ == "__main__":
    name = input("Enter your name: ")
    email = input("Enter your email: ")
    password = input("Enter your password: ")
    store_user_details(name, email, password)
    print("User details stored securely.")
# Storing passwords in plain text is insecure as it exposes sensitive information to anyone who has access to the file.
# Hashing the password before storing it enhances security by ensuring that the actual password is not saved directly.
```

Output:

```
PS C:\Users\srina\OneDrive\Desktop\aiac> & C:/Users/srina/AppData/Lo
Enter your name: Yashuu
Enter your email: mittapalliyashu@gmail.com
Enter your password: yashu
User details stored securely.
```

Explanation:

AI saves name, email, password straight to file readable by anyone, total hack risk.

Fix: Mash password into unreadable hash + random salt; store that instead can't steal real password.

Task Description #3 (Transparency in Algorithm Design)

Objective: Use AI to generate an Armstrong number checking function with comments and explanations.

Instructions:

1. Ask AI to explain the code line-by-line.
2. Compare the explanation with code functionality.

Expected Output:

- Transparent, commented code.
- Correct, easy-to-understand explanation.

Prompt :

generate an Armstrong number checking function with comments and explanations and explain the code line by line ,compare the explanation with full code functionality.

Code :

```python
def is_armstrong_number(num):
    """
    This function checks if a number is an Armstrong number.
    An Armstrong number for a given number of digits is an integer such that the sum of its own digits each raised to the power of the number of digits is equal to the number itself.
    For example, 153 is an Armstrong number because 1^3 + 5^3 + 3^3 = 153.
    """
    # Convert the number to string to easily iterate over each digit
    str_num = str(num)
    # Get the number of digits in the number
    num_digits = len(str_num)
    # Initialize a variable to hold the sum of the digits raised to the power of num_digits
    sum_of_powers = 0

    # Iterate over each digit in the string representation of the number
    for digit in str_num:
        # Convert the digit back to integer and raise it to the power of num_digits, then add it to sum_of_powers
        sum_of_powers += int(digit) ** num_digits

    # Check if the calculated sum_of_powers is equal to the original number
    return sum_of_powers == num
# Example usage
number = int(input("Enter a number to check if it's an Armstrong number: "))
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
# Explanation of the code:
# 1. The function `is_armstrong_number` takes an integer `num` as input.
# 2. It converts the number to a string to facilitate iteration over each digit.
# 3. It calculates the number of digits in the number.
# 4. It initializes a variable `sum_of_powers` to store the cumulative sum of each digit raised to the power of the number of digits.
# 5. It iterates over each digit, converts it back to an integer, raises it to the power of `num_digits`, and adds it to `sum_of_powers`.
# 6. Finally, it checks if `sum_of_powers` is equal to the original number and returns True if they are equal (indicating that it is an Armstrong number) or False otherwise.
# The functionality of the code matches the explanation provided, as it correctly identifies Armstrong numbers based on the defined criteria.
```

Output :

```
Enter a number to check if it's an Armstrong number: 153
153 is an Armstrong number.
PS C:\Users\srina\OneDrive\Desktop\aiac> & C:/Users/srina/AppData/L
Enter a number to check if it's an Armstrong number: 123
123 is not an Armstrong number.
PS C:\Users\srina\OneDrive\Desktop\aiac>
```

Explanation:

Armstrong number: A number equals the sum of its digits each raised to the power of total digit count. Here my Code grabs each digit, powers it correctly and also adds them up, then checks if that matches the original.

Task Description #4 (Transparency in Algorithm Comparison)

Task: Use AI to implement two sorting algorithms (e.g., QuickSort and BubbleSort).

Prompt:

"Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ."

Expected Output:

- Code for both algorithms.
- Transparent, comparative explanation of their logic and efficiency.

Codes :

lab 5.py .env

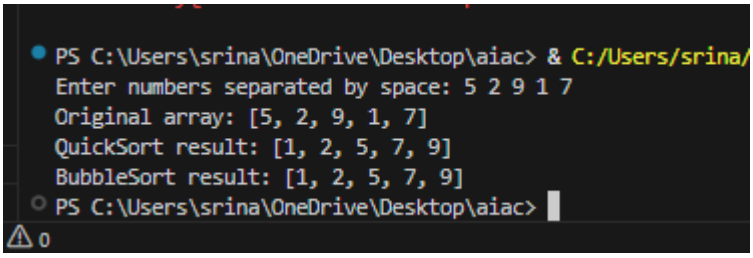
lab_5.py > lab 5.py > ...

```python
#TASK 4:
#Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ.
Qodo: Test this function
def quicksort(arr):
    """
    QuickSort is a divide-and-conquer algorithm that sorts an array by selecting a 'pivot' element
    and partitioning the other elements into two sub-arrays according to whether they are less than or greater than the pivot.
    The sub-arrays are then sorted recursively.
    """
    if len(arr) <= 1:
        return arr  # Base case: arrays with 0 or 1 element are already sorted
    else:
        pivot = arr[len(arr) // 2]  # Choose the middle element as the pivot
        left = [x for x in arr if x < pivot]  # Elements less than the pivot
        middle = [x for x in arr if x == pivot]  # Elements equal to the pivot
        right = [x for x in arr if x > pivot]  # Elements greater than the pivot
        return quicksort(left) + middle + quicksort(right)  # Recursively sort and combine the sub-arrays

Qodo: Test this function
def bubblesort(arr):
    """
    BubbleSort is a simple sorting algorithm that repeatedly steps through the list,
    compares adjacent elements and swaps them if they are in the wrong order.
    The process is repeated until the list is sorted.
    """
    n = len(arr)
    for i in range(n):
        # Track if a swap was made; if no swaps occur, the array is sorted
        swapped = False
        for j in range(0, n-i-1):  # Last i elements are already sorted
            if arr[j] > arr[j+1]:  # Compare adjacent elements
                arr[j], arr[j+1] = arr[j+1], arr[j]  # Swap if they are in the wrong order
                swapped = True
        if not swapped:
            break  # If no swaps were made, the array is sorted
    return arr
if __name__ == "__main__":
    arr = list(map(int, input("Enter numbers separated by space: ").split()))

    print("Original array:", arr)
    print("QuickSort result:", quicksort(arr))
    print("BubbleSort result:", bubblesort(arr.copy()))
```

Output :



Explanation:

BubbleSort slowly swaps neighbor pairs until none needed simple but crawls on big lists.whereas,QuickSort picks middle value, splits smaller/larger sides, repeats divide fast splitter for real work.

Task Description #5 (Transparency in AI Recommendations)

Task: Use AI to create a product recommendation system.

Prompt:

"Generate a recommendation system that also provides reasons for each suggestion."

Expected Output:

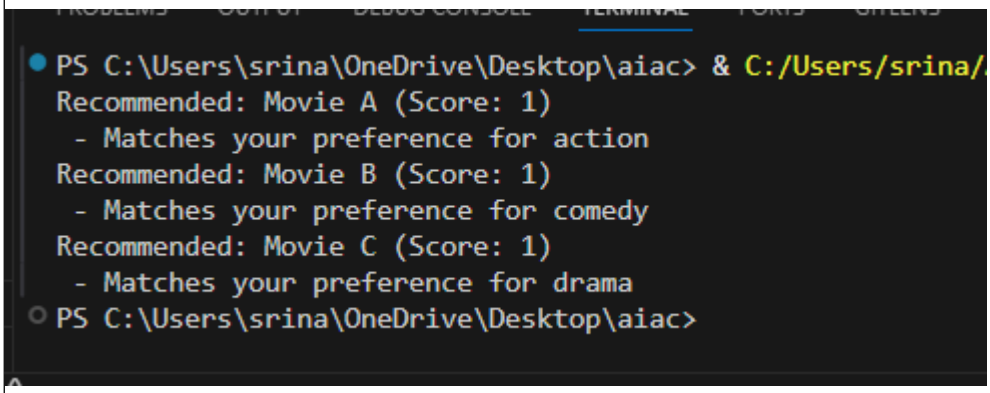
- Code with explainable recommendations.
- Evaluation of whether explanations are understandable.

Code :

lab_5.py > lab 5.py > ...

```
130     print("Bubblesort result: ", bubblesort(arr.copy()))
131
132 '''
133 #TASK 5:
134 #Generate a recommendation system that also provides reasons for each suggestion.
135 import random
    Qodo: Test this function
136 def recommend_items(user_preferences, items):
137     recommendations = []
138     for item in items:
139         score = 0
140         reasons = []
141         for preference in user_preferences:
142             if preference in item['tags']:
143                 score += 1
144                 reasons.append(f"Matches your preference for {preference}")
145         if score > 0:
146             recommendations.append((item['name'], score, reasons))
147     recommendations.sort(key=lambda x: x[1], reverse=True)
148     return recommendations
149 if __name__ == "__main__":
150     user_preferences = ['action', 'comedy', 'drama']
151     items = [
152         {'name': 'Movie A', 'tags': ['action', 'thriller']},
153         {'name': 'Movie B', 'tags': ['comedy', 'romance']},
154         {'name': 'Movie C', 'tags': ['drama', 'biography']},
155         {'name': 'Movie D', 'tags': ['horror', 'thriller']},
156     ]
157     recommendations = recommend_items(user_preferences, items)
158     for name, score, reasons in recommendations:
159         print(f"Recommended: {name} (Score: {score})")
160         for reason in reasons:
161             print(f" - {reason}")
162
```

Output:



Explanation:

Recommendation system scans other users' tastes matching ours, then they suggest the matches according to our preferences and tastes so that every person can watch their own genre or taste preferences.

.