

NAME:KOLA SNEHA ID:2403A510F4 BATCH-05

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName: B. Tech		AssignmentType: 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	Week3 –Monday	Time(s)	
Duration	2 Hours	Applicable to Batches	
AssignmentNumber: 5.1(Present assignment number)/24(Total number of assignments)			
Q.No.	Question	Expected Time to complete	
1	Lab 5: Ethical Foundations – Responsible AI Coding Practices Lab Objectives:	Week3 - Monday	

	<ul style="list-style-type: none">• 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. <p>Lab Outcomes (LOs): After completing this lab, students will be able to:</p> <ul style="list-style-type: none">• 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.. <p>Task Description #1 (Privacy in API Usage) Task: Use an AI tool to generate a Python program that connects to a weather API. Prompt: <i>"Generate code to fetch weather data securely without exposing API keys in the code."</i> Expected Output:</p> <ul style="list-style-type: none">• Original AI code (check if keys are hardcoded).• Secure version using environment variables <p>Prompt:</p> <div>Generate a Python program that fetches current weather data from a weather API (like OpenWeatherMap). First, show an insecure version where the API key is hardcoded inside the code. Then, provide a secure version where the API key is stored and accessed from environment variables instead of being exposed in the code. Also, explain briefly why using environment variables is safer.</div> <p>Code:</p>	
--	---	--

```
# filepath: [assign_4_3.py](http://_vscodecontentref_/0)

import requests

def get_weather(city):
    api_key = "YOUR_API_KEY_HERE" # Insecure: API key is hardcoded
    url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        print(f"Weather in {city}: {data['weather'][0]['description']}, Temperature: {data['main']['temp']}°C")
    else:
        print("Failed to fetch weather data.")

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

```
# filepath: [assign_4_3.py](http://_vscodecontentref_/0)
# filepath: [assign_4_3.py](http://_vscodecontentref_/1)

import os
import requests

def get_weather(city):
    api_key = os.getenv("OPENWEATHER_API_KEY") # Secure: API key from environment variable
    if not api_key:
        print("API key not found. Please set the OPENWEATHER_API_KEY environment variable.")
        return
    url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        print(f"Weather in {city}: {data['weather'][0]['description']}, Temperature: {data['main']['temp']}°C")
    else:
        print("Failed to fetch weather data.")

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

```
set OPENWEATHER_API_KEY=your_actual_api_key
```

```
$env:OPENWEATHER_API_KEY="your_actual_api_key"
```

Output:

```
Enter city name: Hyderabad
Weather in Hyderabad: scattered clouds, Temperature: 29.5°C
```

```
Enter city name: InvalidCity
Failed to fetch weather data.
```

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 data (name, email, password) in a file	
--	--	--

Code:

```
def register_user():
    name = input("Enter your name: ")
    email = input("Enter your email: ")
    password = input("Enter your password: ")
    with open("users.txt", "a") as file:
        file.write(f"{name},{email},{password}\n")
    print("User registered successfully!")

if __name__ == "__main__":
    register_user()
```

Output:

```
Enter your name: sri
Enter your email: sri123@gmail.com
Enter your password: sri@123
User registered successfully!
```

IN the file:

- **Plain-text password storage:** Passwords are stored exactly as entered.
- **No hashing or encryption:** A data breach would expose user passwords.
- **Vulnerable to insider threats:** Anyone with access to the file can read sensitive data.

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 python code for Armstrong number checking function with
comments and explanations
```

Code:

```
def is_armstrong(number):
    """
    Checks if a number is an Armstrong number.
    An Armstrong number is an n-digit number that is equal to the sum of its own digits
    each raised to the power of n.
    Example: 153 = 1^3 + 5^3 + 3^3 = 153
    """
    # Convert the number to string to easily iterate over digits
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0

    # Calculate the sum of each digit raised to the power of number of digits
    for digit in num_str:
        sum_of_powers += int(digit) ** num_digits

    # Check if the sum equals the original number
    return sum_of_powers == number

if __name__ == "__main__":
    num = int(input("Enter a number to check if it is an Armstrong number: "))
    if is_armstrong(num):
        print(f"{num} is an Armstrong number.")
    else:
        print(f"{num} is not an Armstrong number.")
```

Output:

```
Enter a number to check if it is an Armstrong number: 469
469 is not an Armstrong number.
```

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. **Prompt:**

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

Code:

```

def quicksort(arr):
    """
    QuickSort is a divide-and-conquer algorithm.
    It picks a 'pivot' element and partitions the array into two sub-arrays:
    elements less than the pivot and elements greater than the pivot.
    It then recursively sorts the sub-arrays.
    """
    if len(arr) <= 1:
        return arr # Base case: arrays with 0 or 1 element are already sorted
    else:
        pivot = arr[0] # Choose the first element as pivot
        less = [x for x in arr[1:] if x <= pivot] # Elements less than or equal to pivot
        greater = [x for x in arr[1:] if x > pivot] # Elements greater than pivot
        # Recursively sort sub-arrays and combine
        return quicksort(less) + [pivot] + quicksort(greater)

def bubblesort(arr):
    """
    BubbleSort repeatedly steps through the list, compares adjacent elements,
    and swaps them if they are in the wrong order.
    This process is repeated until the list is sorted.
    """
    n = len(arr)
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap if the element found is greater than the next element
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

if __name__ == "__main__":
    # Example usage and demonstration of differences
    data = [64, 34, 25, 12, 22, 11, 90]
    print("Original list:", data)

    # QuickSort (returns a new sorted list)
    sorted_quick = quicksort(data.copy())
    print("Sorted with QuickSort:", sorted_quick)

    # BubbleSort (sorts in place)
    sorted_bubble = bubblesort(data.copy())
    print("Sorted with BubbleSort:", sorted_bubble)

    # Key difference:
    # - QuickSort is generally faster (O(n log n) average) and uses recursion.
    # - BubbleSort is simpler but slower (O(n^2)),

```

Output:

```

Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted with QuickSort: [11, 12, 22, 25, 34, 64, 90]
Sorted with BubbleSort: [11, 12, 22, 25, 34, 64, 90]

```

Task Description #5 (Transparency in AI Recommendations) Task:
Use AI to create a product recommendation system.

Prompt:

	<p><i>"Generate a recommendation system that also provides reasons for each suggestion."</i></p>	
--	--	--

Expected Output:

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

Prompt:

Generate a python code create a product recommendation system. recommendation system that also provides reasons for each suggestion with explanations in comments

Code:

```
def recommend_products(user_preferences, products):
    """
    Recommends products based on user preferences and provides reasons for each suggestion.
    :param user_preferences: dict with user preferences (e.g., {'category': 'electronics', 'budget': 1000})
    :param products: list of product dicts (each with 'name', 'category', 'price', etc.)
    :return: list of tuples (product, reason)
    """
    recommendations = []
    for product in products:
        # Check if product matches preferred category
        if product['category'] == user_preferences.get('category'):
            # Check if product is within budget
            if product['price'] <= user_preferences.get('budget', float('inf')):
                reason = f"Matches your preferred category '{product['category']}' and is within your budget."
                recommendations.append((product, reason))
            else:
                reason = f"Matches your preferred category '{product['category']}', but is above your budget."
                recommendations.append((product, reason))
        else:
            # If not matching category, but within budget, still suggest with a reason
            if product['price'] <= user_preferences.get('budget', float('inf')):
                reason = f"Is within your budget, though not your preferred category."
                recommendations.append((product, reason))
    return recommendations

if __name__ == "__main__":
    # Example product list
    products = [
        {'name': 'Smartphone', 'category': 'electronics', 'price': 900},
        {'name': 'Laptop', 'category': 'electronics', 'price': 1200},
        {'name': 'Book', 'category': 'books', 'price': 20},
        {'name': 'Headphones', 'category': 'electronics', 'price': 150},
        {'name': 'Coffee Mug', 'category': 'kitchen', 'price': 10}
    ]

    # Example user preferences
    user_preferences = {
        'category': 'electronics',
        'budget': 1000
    }

    # Get recommendations
    recommendations = recommend_products(user_preferences, products)

    # Display recommendations with reasons
    print("Product Recommendations:")
    for product, reason in recommendations:
        print(f"- {product['name']} (Rs.{product['price']}): {reason}")

    # Explanations:
    # - The system checks each product for category and budget match.
    # - If both match, it's a strong recommendation with a clear reason.
    # - If only budget matches, it's a weaker suggestion, and the reason explains this.
    # - This approach helps users understand why
```


Output:

```
Product Recommendations:
- Smartphone (Rs.900): Matches your preferred category 'electronics' and is within your budget.
- Laptop (Rs.1200): Matches your preferred category 'electronics', but is above your budget.
- Book (Rs.20): Is within your budget, though not your preferred category.
- Headphones (Rs.150): Matches your preferred category 'electronics' and is within your budget.
- Coffee Mug (Rs.10): Is within your budget, though not your preferred category.
```

Task Description #6 (Transparent Code Generation)

Task: Ask AI to generate a Python function for calculating factorial using recursion.

Prompt:

"Generate a recursive factorial function with comments that explain each line and a final summary of the algorithm's flow." Expected

Output:

- Fully commented code.
- Clear documentation of how recursion works. **Prompt:**

```
generate a Python function for calculating factorial using recursion.
```

Code:

```
def factorial(n):
    """
    Calculates the factorial of a number using recursion.
    :param n: integer value to calculate factorial for
    :return: factorial of n
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

# Example usage:
if __name__ == "__main__":
    num = int(input("Enter a number to find its factorial: "))
    try:
        result = factorial(num)
        print(f"The factorial of {num} is {result}")
    except ValueError as e:
        print(f"Error: {e}")
```

Output:

```
Enter a number to find its factorial: 5
The factorial of 5 is 120
```

Task Description #7 (Inclusiveness in Customer Support) Code

Snippet:


```
def support_reply(name, gender):
    if gender.lower() == "male":
        prefix = "Mr."
    else:
        prefix = "Mrs."
    return f"Dear {prefix} {name}, we have resolved your i
```

Task:

Regenerate the code so that support messages use neutral language (e.g., “Dear {name}”) and optionally accept preferred titles. Expected

Output:

- Neutral, user-friendly support responses.

Prompt:

Regenerate the code so that support messages use neutral language (e.g., “Dear {name}”) and optionally accept preferred titles.

Code:

```
def support_reply(name, gender=None, title=None):
    """
    Generates a support reply message using neutral language.
    Optionally accepts a preferred title.
    :param name: User's name
    :param gender: (Optional) User's gender (ignored for neutrality)
    :param title: (Optional) Preferred title (e.g., Dr., Prof.)
    :return: Support reply string
    """
    if title:
        greeting = f"Dear {title} {name}"
    else:
        greeting = f"Dear {name}"
    return f"{greeting}, we have received your request and our support team will get back to you shortly."

name = input("Enter your name: ")
gender = input("Enter your gender (optional, press Enter to skip): ") or None
title = input("Enter your preferred title (e.g., Dr., Prof., optional, press Enter to skip): ") or None
support_reply(name, gender, title)
```

Output:

```
Enter your name: sai
Enter your gender (optional, press Enter to skip):
Enter your preferred title (e.g., Dr., Prof., optional, press Enter to skip): student
Dear student sai, we have received your request and our support team will get back to you shortly.
```

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		
		Transparency	1		
		Inclusiveness	0.5		
		Data security and Privacy	1		
		Total	2.5 Marks		