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	Week3 - Tuesday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:5.2(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*ExpectedTime to complete*
1	Lab 5: Ethical Foundations – Responsible AI Coding Practices **Lab Objectives:**	Week3 - Wednesday

- 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 and Data Security)

- Use an AI tool (e.g., Copilot, Gemini, Cursor) to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption.

Expected Output#1

- Identification of insecure logic; revised secure version with proper password hashing and environment variable use.
- **Prompt:** Generate a Python login system. Avoid hardcoded passwords and plain-text storage. Implement secure password handling using hashing (bcrypt/ hashlib) and use environment variables for

sensitive data instead of hardcoding.

```python
import bcrypt
from dotenv import load_dotenv
import os

load_dotenv()

def hash_password(password):
    """Hashes a password using bcrypt and a secret key from environment variables."""
    secret_key = os.getenv('SECRET_KEY')
    if not secret_key:
        raise ValueError("SECRET_KEY not found in environment variables.")

    encoded_secret_key = secret_key.encode('utf-8')
    encoded_password = password.encode('utf-8')

    salt = bcrypt.gensalt()
    hashed_bytes = bcrypt.hashpw(encoded_password, salt + encoded_secret_key)

    return hashed_bytes.decode('utf-8')
```

```python
def main():
    """Main function for the command-line login system interface."""
    while True:
        print("\nOptions: register, login, exit")
        choice = input("Enter your choice: ").strip().lower()
        if choice == "register":
            username = input("Enter username: ").strip()
            password = input("Enter password: ").strip()
            register_user(username, password)
        elif choice == "login":
            username = input("Enter username: ").strip()
            password = input("Enter password: ").strip()
            login_user(username, password)
        elif choice == "exit":
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
```

```python
user_database = {}

def register_user(username, password):
    """Registers a new user by hashing their password and storing it securely."""
    if username in user_database:
        print(f"Error: Username '{username}' already exists.")
        return False

    try:
        hashed_password = hash_password(password)
        user_database[username] = hashed_password
        print(f"User '{username}' registered successfully.")
        return True
    except ValueError as e:
        print(f"Error during registration: {e}")
        return False
```

```python
def login_user(username, password):
    """Verifies user credentials by comparing the provided password with the stored hash."""
    if username not in user_database:
        print("Error: Invalid credentials.")
        return False

    stored_hashed_password = user_database[username]
    secret_key = os.getenv('SECRET_KEY')
    if not secret_key:
        print("Error: SECRET_KEY not found for verification.")
        return False

    encoded_secret_key = secret_key.encode('utf-8')
    encoded_password = password.encode('utf-8')
    encoded_stored_hashed_password = stored_hashed_password.encode('utf-8')
    try:
        if bcrypt.checkpw(encoded_password + encoded_secret_key, encoded_stored_hashed_password):
            print(f"User '{username}' logged in successfully.")
            return True
        else:
            print("Error: Invalid credentials.")
            return False
    except ValueError as e:
        print(f"Error during login verification: {e}")
        return False
```

```
Options: register, login, exit
Enter your choice: login
Enter username: Bhanu Teja
Enter password: Student@0420
Error: Invalid credentials.

Options: register, login, exit
Enter your choice: register
Enter username: Bhanu Teja
Enter password: Student@0420
User 'Bhanu Teja' registered successfully.

Options: register, login, exit
Enter your choice: login
Enter username: Bhanu Teja
Enter password: Student@0420
Error: Invalid credentials.

Options: register, login, exit
Enter your choice: exit
Exiting program.
```

Task Description#2 (Bias)

- Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the AI-generated logic exhibits bias or differing criteria based on names or genders.

Expected Output#2

- Screenshot or code comparison showing bias (if any); write 3–4 sentences on mitigation techniques.
- **Prompt**: Generate a simple loan approval system in Python. Then test with inputs: ‘loan approval for John’, ‘loan approval for Priya’. Do not allow any gender or name-based bias. Ensure that the approval logic depends only on neutral financial criteria like income, credit score, and debt ratio.

```python
def approve_loan(income, credit_score, debt_ratio):

    if income >= MIN_INCOME and credit_score >= MIN_CREDIT_SCORE and debt_ratio <= MAX_DEBT_RATIO:
        return True
    else:
        return False
```

```python
def get_loan_application_details():
    """Gets loan application details (income, credit score, debt ratio) from user input."""
    while True:
        try:
            income = int(input("Enter your annual income: "))
            credit_score = int(input("Enter your credit score: "))
            debt_ratio = float(input("Enter your debt-to-income ratio (e.g., 0.4 for 40%): "))
            return income, credit_score, debt_ratio
        except ValueError:
            print("Invalid input. Please enter numeric values for income, credit score, and debt ratio.")
```

```python
print("--- Testing Loan Approval System ---")
print("\nTest Case 1: Approved Loan")
income1, credit_score1, debt_ratio1 = 40000, 700, 0.3
print(f"Applicant Details: Income={income1}, Credit Score={credit_score1}, Debt Ratio={debt_ratio1}")
if approve_loan(income1, credit_score1, debt_ratio1):
    print("Loan Approved!")
else:
    print("Loan Denied.")

print("\nTest Case 2: Denied due to low income")
income2, credit_score2, debt_ratio2 = 25000, 700, 0.3
print(f"Applicant Details: Income={income2}, Credit Score={credit_score2}, Debt Ratio={debt_ratio2}")
if approve_loan(income2, credit_score2, debt_ratio2):
    print("Loan Approved!")
else:
    print("Loan Denied.")

print("\nTest Case 3: Denied due to low credit score")
income3, credit_score3, debt_ratio3 = 40000, 600, 0.3
print(f"Applicant Details: Income={income3}, Credit Score={credit_score3}, Debt Ratio={debt_ratio3}")
if approve_loan(income3, credit_score3, debt_ratio3):
    print("Loan Approved!")
else:
    print("Loan Denied.")

print("\nTest Case 4: Denied due to high debt ratio")
income4, credit_score4, debt_ratio4 = 40000, 700, 0.5
print(f"Applicant Details: Income={income4}, Credit Score={credit_score4}, Debt Ratio={debt_ratio4}")
if approve_loan(income4, credit_score4, debt_ratio4):
    print("Loan Approved!")
else:
    print("Loan Denied.")
```

```
print("\nTest Case 5: Edge case - meeting minimum income exactly")
income5, credit_score5, debt_ratio5 = MIN_INCOME, 700, 0.3
print(f"Applicant Details: Income={income5}, Credit Score={credit_score5}, Debt Ratio={debt_ratio5}")
if approve_loan(income5, credit_score5, debt_ratio5):
    print("Loan Approved!")
else:
    print("Loan Denied.")

print("\nTest Case 6: Edge case - meeting minimum credit score exactly")
income6, credit_score6, debt_ratio6 = 40000, MIN_CREDIT_SCORE, 0.3
print(f"Applicant Details: Income={income6}, Credit Score={credit_score6}, Debt Ratio={debt_ratio6}")
if approve_loan(income6, credit_score6, debt_ratio6):
    print("Loan Approved!")
else:
    print("Loan Denied.")

print("\nTest Case 7: Edge case - meeting maximum debt ratio exactly")
income7, credit_score7, debt_ratio7 = 40000, 700, MAX_DEBT_RATIO
print(f"Applicant Details: Income={income7}, Credit Score={credit_score7}, Debt Ratio={debt_ratio7}")
if approve_loan(income7, credit_score7, debt_ratio7):
    print("Loan Approved!")
else:
    print("Loan Denied.")

print("\nTest Case 8: Bias Check (should approve like Test Case 1)")
income8, credit_score8, debt_ratio8 = 40000, 700, 0.3
print(f"Applicant Details: Income={income8}, Credit Score={credit_score8}, Debt Ratio={debt_ratio8}")
if approve_loan(income8, credit_score8, debt_ratio8):
    print("Loan Approved!")
else:
    print("Loan Denied.")
```

```
--- Testing Loan Approval System ---

Test Case 1: Approved Loan
Applicant Details: Income=40000, Credit Score=700, Debt Ratio=0.3
Loan Approved!

Test Case 2: Denied due to low income
Applicant Details: Income=25000, Credit Score=700, Debt Ratio=0.3
Loan Denied.

Test Case 3: Denied due to low credit score
Applicant Details: Income=40000, Credit Score=600, Debt Ratio=0.3
Loan Denied.

Test Case 4: Denied due to high debt ratio
Applicant Details: Income=40000, Credit Score=700, Debt Ratio=0.5
Loan Denied.

Test Case 5: Edge case - meeting minimum income exactly
Applicant Details: Income=30000, Credit Score=700, Debt Ratio=0.3
Loan Approved!

Test Case 6: Edge case - meeting minimum credit score exactly
Applicant Details: Income=40000, Credit Score=650, Debt Ratio=0.3
Loan Approved!

Test Case 7: Edge case - meeting maximum debt ratio exactly
Applicant Details: Income=40000, Credit Score=700, Debt Ratio=0.4
Loan Approved!

Test Case 8: Bias Check (should approve like Test Case 1)
Applicant Details: Income=40000, Credit Score=700, Debt Ratio=0.3
Loan Approved!

--- Testing Complete ---
```

Task Description#3 (Transparency)

- Write prompt to write function calculate the nth

Fibonacci number using recursion and generate comments and explain code document

Expected Output#3

- Code with explanation
- **Assess: Is the explanation understandable and correct?**
- **Prompt:** Write a Python function to calculate the nth Fibonacci number using recursion. Add detailed comments for each step and then generate a plain-language explanation of how the code works.

```
def fibonacci_recursive(n):

    if n == 0:
        return 0

    elif n == 1:
        return 1

    elif n < 0:
        raise ValueError("Input must be a non-negative integer.")

    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
```

```
print(f"Fibonacci at position 0: {fibonacci_recursive(0)}")
print(f"Fibonacci at position 1: {fibonacci_recursive(1)}")
print(f"Fibonacci at position 7: {fibonacci_recursive(7)}")
print(f"Fibonacci at position 10: {fibonacci_recursive(10)}")
try:
    print(f"Fibonacci at position -2: {fibonacci_recursive(-2)}")
except ValueError as e:
    print(f"Caught expected error for negative input: {e}")
```

```
Fibonacci at position 0: 0
Fibonacci at position 1: 1
Fibonacci at position 7: 13
Fibonacci at position 10: 55
Caught expected error for negative input: Input must be a non-negative integer.
```

1. What is the Fibonacci Sequence? The Fibonacci sequence starts with 0 and 1. Every number after that is simply the sum of the two numbers before it (e.g., 0, 1, 1, 2, 3, 5, 8, ...).
2. The Function fibonacci_recursive(n)

It takes one input, n, which tells us which position in the sequence we want to find the Fibonacci number for (starting with position 0).

3. Base Cases (The Stopping Points)

if n == 0:: If n is 0, the function immediately returns 0. This is the very first number in the sequence.
elif n == 1:: If n is 1, the function immediately returns 1. This is the second number in the sequence.
These two cases are crucial because they tell the function when to stop calling itself. Without them, the function would run forever.

4. Handling Negative Input

elif n < 0:: The standard Fibonacci sequence doesn't include negative positions. If you give the function a negative number, it raises an error (ValueError) to let you know that's not valid input.

5. The Recursive Step (The Self-Calling Part)

else:: If n is greater than 1, this is where the magic of recursion happens. The function needs to find the Fibonacci number at position n. According to the definition of the Fibonacci sequence, this number is the sum of the numbers at positions n-1 and n-2.
return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2): This line is the core of the recursion. The function calls *itself* twice, once for n-1 and once for n-2. It then adds the results of these two calls together.
This process continues with each call breaking down the problem into smaller and smaller steps (n-3, n-4, etc.) until it eventually reaches the base cases (when the input becomes 0 or 1). At that point, the base cases return their fixed values (0 or 1), and these values are added back up the chain of calls until the original call for n is finally calculated.
In essence, the recursive function solves the problem by breaking it down into simpler versions of the same problem until it reaches the simplest cases (0 and 1), and then it builds the solution back up from there.

Task Description#4 (Bias)

- Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings.

Expected Output#4

- Python code
- Analyze is there any bias with respect to gender or any
- **Prompt:** Generate a Python program that scores job applicants based on education, experience, and skills. Do not use gender, age, race, or name as criteria. After generating, explain whether the system is free from bias or not.

```python
def score_applicant(applicant_data):
    score = 0
    education_scores = {
        'High School': 10,
        'Associate': 20,
        'Bachelor': 30,
        'Master': 40,
        'PhD': 50
    }
    education_level = applicant_data.get('education', '')
    score += education_scores.get(education_level, 0)
    experience_years = applicant_data.get('experience_years', 0)
    score += experience_years * 5
    required_skills = ['Python', 'SQL', 'Machine Learning', 'Data Analysis']
    applicant_skills = applicant_data.get('skills', [])
    for skill in applicant_skills:
        if skill in required_skills:
            score += 10
    return score

applicant1 = {
    'education': 'Master',
    'experience_years': 5,
    'skills': ['Python', 'SQL', 'Communication'],
    'gender': 'Female',
    'age': 30,
    'race': 'Asian',
    'name': 'Jane Doe'
}
```

```
applicant2 = {
    'education': 'Bachelor',
    'experience_years': 2,
    'skills': ['SQL', 'Excel'],
    'gender': 'Male',
    'age': 25,
    'race': 'Caucasian',
    'name': 'John Smith'
}

score1 = score_applicant(applicant1)
score2 = score_applicant(applicant2)

print(f"Applicant 1 Score: {score1}")
print(f"Applicant 2 Score: {score2}")
```

```
Applicant 1 Score: 85
Applicant 2 Score: 50
```

Data Analysis Key Findings

- A Python function score_applicant was successfully created to score job applicants based on education, experience, and skills, explicitly excluding gender, age, race, and name as criteria.
- The scoring system assigns points based on predefined values for education levels (e.g., 50 for PhD, 10 for High School), points per year of experience (5 points), and points for possessing required skills (10 points each).
- Despite excluding protected attributes, the system is not guaranteed to be bias-free due to potential biases in the chosen criteria:

Education Bias: Favors higher education, access to which may not be equally distributed.
Experience Bias: Relies on years of experience, which can

be influenced by biases in career opportunities.
Skills Bias: The selection of required skills might reflect existing workforce biases or unequal access to skill development.
Insights or Next Steps

- Regularly audit and validate the scoring system against diverse applicant pools to identify and mitigate potential biases introduced by the chosen criteria and their weighting.
- Consider alternative or supplementary criteria that might be less susceptible to societal biases, or explore methods to adjust scores to account for potential disparities in access to education, experience, or skill development.

Task Description#5 (Inclusiveness)

- Code Snippet

```
def greet_user(name, gender):
    if gender.lower() == "male":
        title = "Mr."
    else:
        title = "Mrs."
    return f"Hello, {title} {name}! Welcome."
```

Expected Output#5

- Regenerate code that includes **gender-neutral** also
- **Prompt:** Regenerate a simple Python code snippet (like user profile creation or form input) so that it includes gender-neutral options. Instead of binary gender categories (Male/Female), allow inclusive inputs like 'Non-binary', 'Prefer not to say', or custom options.

```
user_profile = {
    "name": "Alex",
    "age": 30,
    "gender": "Non-binary",  # or "Prefer not to say", "Custom: [specify]", "Female", "Male"
    "email": "alex@example.com"
}

print(user_profile)
```

```
{'name': 'Alex', 'age': 30, 'gender': 'Non-binary', 'email': 'alex@example.com'}
```

REFLECTION:
Even though AI tools can generate code quickly, the ultimate

	responsibility for that code lies with the human developer. AI cannot fully understand the social, legal, or ethical consequences of the solutions it produces. For example, insecure password handling, biased decision-making, or non-inclusive design are risks that an AI may introduce unknowingly. As developers, we must review and validate AI-generated code, apply security and privacy best practices, and ensure inclusiveness and fairness. Accountability means being aware that AI is only an assistant, not an authority. Ethical AI coding practices require transparency in how the code works, critical evaluation of possible biases, and continuous improvements guided by human judgment. In short, the human role is to act as the ethical gatekeeper—taking ownership of decisions, correcting flaws in AI outputs, and ensuring that the software we build is secure, fair, and trustworthy.	