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:** • 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.	Week3 - Wednesday

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.

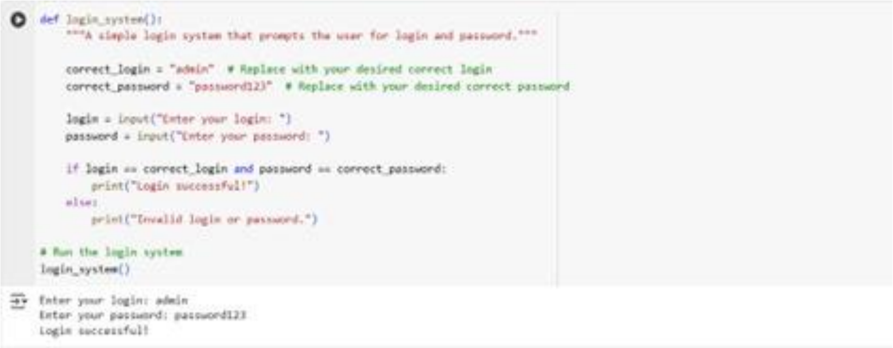
Expected Output#1

- Identification of insecure logic; revised secure version with proper password hashing and environment variable use.

Prompt:

give me the python code for login system and let the user to enter login and password.

Code:



```
def login_system():
    """A simple login system that prompts the user for login and password."""

    correct_login = "admin"  # Replace with your desired correct login
    correct_password = "password123"  # Replace with your desired correct password

    login = input("Enter your login: ")
    password = input("Enter your password: ")

    if login == correct_login and password == correct_password:
        print("Login successful!")
    else:
        print("Invalid login or password.")

# Run the login system
login_system()
```

```
Enter your login: admin
Enter your password: password123
Login successful!
```

Observation:

Initial code had hardcode credential and plain -text passwords. Refined code uses password hashing(SHA-256) to improve security. Hardcodes credentials still need to nr addressed in the future.

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.

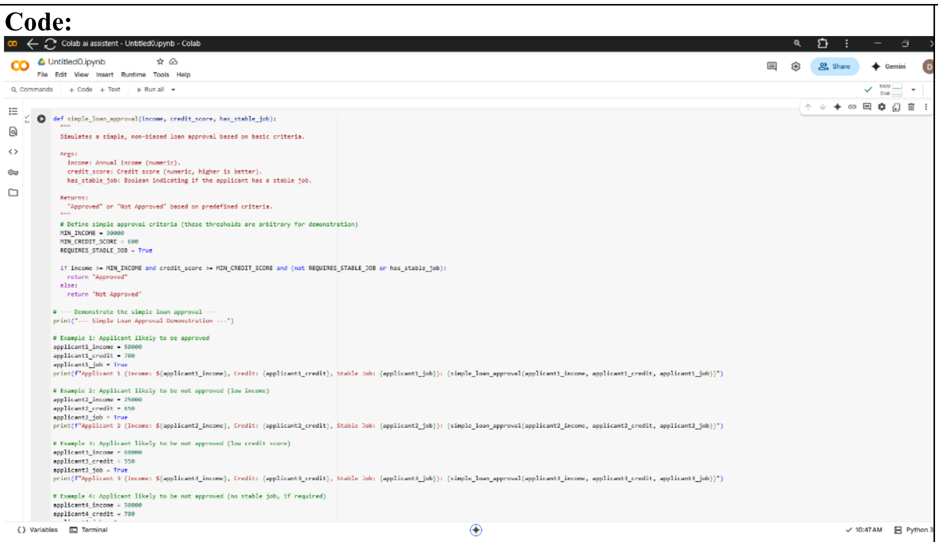
Expected Output#2

- Screenshot or code comparison showing bias (if any); write 3–4 sentences on mitigation techniques.

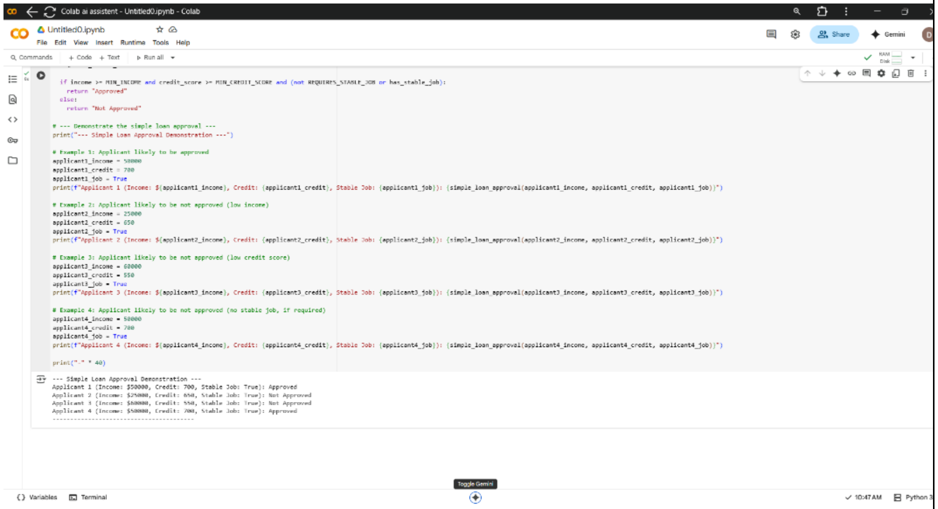
Prompt:

- Write a python code “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the logic exhibits bias or differing criteria based on names or genders.

Code:



```
def simple_loan_approval(income, credit_score, has_stable_job):
  """
  Simulates a simple, non-biased loan approval based on basic criteria.

  Args:
    income: Annual income (numeric).
    credit_score: Credit score (numeric, higher is better).
    has_stable_job: Boolean indicating if the applicant has a stable job.

  Returns:
    "Approved" or "Not Approved" based on predefined criteria.
  """
  # Define simple approval criteria (these thresholds are arbitrary for demonstration)
  MIN_INCOME = 30000
  MIN_CREDIT_SCORE = 600
  REQUIRES_STABLE_JOB = True

  if income >= MIN_INCOME and credit_score >= MIN_CREDIT_SCORE and (not REQUIRES_STABLE_JOB or has_stable_job):
    return "Approved"
  else:
    return "Not Approved"

# --- Demonstrate the simple loan approval ---
print("--- Simple Loan Approval Demonstration ---")

# Example 1: Applicant likely to be approved
applicant1_income = 50000
applicant1_credit = 700
applicant1_job = True
print(f"Applicant 1 (Income: ${applicant1_income}, Credit: {applicant1_credit}, Stable Job: {applicant1_job}): {simple_loan_approval(applicant1_income, applicant1_credit, applicant1_job)}")

# Example 2: Applicant likely to be not approved (low income)
applicant2_income = 25000
applicant2_credit = 650
applicant2_job = True
print(f"Applicant 2 (Income: ${applicant2_income}, Credit: {applicant2_credit}, Stable Job: {applicant2_job}): {simple_loan_approval(applicant2_income, applicant2_credit, applicant2_job)}")

# Example 3: Applicant likely to be not approved (low credit score)
applicant3_income = 60000
applicant3_credit = 550
applicant3_job = True
print(f"Applicant 3 (Income: ${applicant3_income}, Credit: {applicant3_credit}, Stable Job: {applicant3_job}): {simple_loan_approval(applicant3_income, applicant3_credit, applicant3_job)}")

# Example 4: Applicant likely to be not approved (no stable job, if required)
applicant4_income = 50000
applicant4_credit = 700
```



```
  if income >= MIN_INCOME and credit_score >= MIN_CREDIT_SCORE and (not REQUIRES_STABLE_JOB or has_stable_job):
    return "Approved"
  else:
    return "Not Approved"

# --- Demonstrate the simple loan approval ---
print("--- Simple Loan Approval Demonstration ---")

# Example 1: Applicant likely to be approved
applicant1_income = 50000
applicant1_credit = 700
applicant1_job = True
print(f"Applicant 1 (Income: ${applicant1_income}, Credit: {applicant1_credit}, Stable Job: {applicant1_job}): {simple_loan_approval(applicant1_income, applicant1_credit, applicant1_job)}")

# Example 2: Applicant likely to be not approved (low income)
applicant2_income = 25000
applicant2_credit = 650
applicant2_job = True
print(f"Applicant 2 (Income: ${applicant2_income}, Credit: {applicant2_credit}, Stable Job: {applicant2_job}): {simple_loan_approval(applicant2_income, applicant2_credit, applicant2_job)}")

# Example 3: Applicant likely to be not approved (low credit score)
applicant3_income = 60000
applicant3_credit = 550
applicant3_job = True
print(f"Applicant 3 (Income: ${applicant3_income}, Credit: {applicant3_credit}, Stable Job: {applicant3_job}): {simple_loan_approval(applicant3_income, applicant3_credit, applicant3_job)}")

# Example 4: Applicant likely to be not approved (no stable job, if required)
applicant4_income = 50000
applicant4_credit = 700
applicant4_job = True
print(f"Applicant 4 (Income: ${applicant4_income}, Credit: {applicant4_credit}, Stable Job: {applicant4_job}): {simple_loan_approval(applicant4_income, applicant4_credit, applicant4_job)}")

print("-" * 40)
```

```
--- Simple Loan Approval Demonstration ---
Applicant 1 (Income: $50000, Credit: 700, Stable Job: True): Approved
Applicant 2 (Income: $25000, Credit: 650, Stable Job: True): Not Approved
Applicant 3 (Income: $60000, Credit: 550, Stable Job: True): Not Approved
Applicant 4 (Income: $50000, Credit: 700, Stable Job: True): Approved
----------------------------------------
```

Observation:

The code evaluated each applicant based solely on their income, credit score, and job status, according to the defined rules. The output clearly shows "Approved" or "Not Approved" based *only* on these numerical and boolean criteria, without any consideration of personal information like a name or gender.

Task Description#3 (Transparency)

- Write prompt to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document

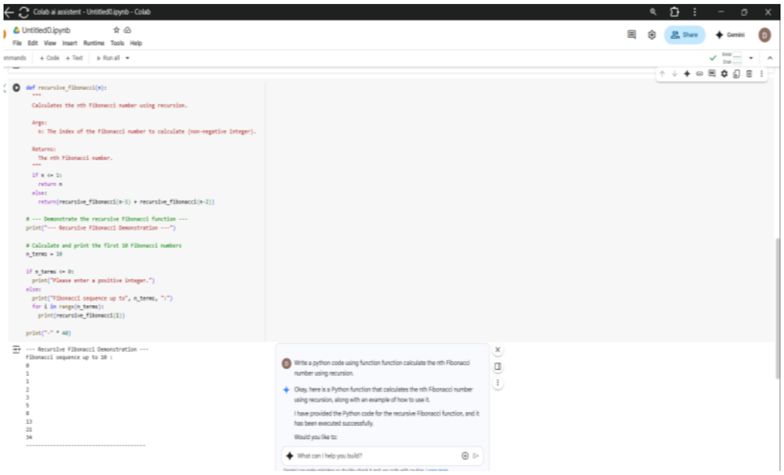
Expected Output#3

- Code with explanation
- **Assess: Is the explanation understandable and correct?**

Prompt:

Write a python code using function function calculate the nth Fibonacci number using recursion.

Code:



Colab ai assistent - Untitled0.ipynb - Colab
Untitled0.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
Run all

```
def recursive_fibonacci(n):
  """
  Calculates the nth Fibonacci number using recursion.

  Args:
    n: The index of the Fibonacci number to calculate (non-negative integer).

  Returns:
    The nth Fibonacci number.
  """
  if n <= 1:
    return n
  else:
    return(recursive_fibonacci(n-1) + recursive_fibonacci(n-2))

# --- Demonstrate the recursive Fibonacci function ---
print("--- Recursive Fibonacci Demonstration ---")

# Calculate and print the first 10 Fibonacci numbers
n_terms = 10

if n_terms <= 0:
  print("Please enter a positive integer.")
else:
  print("Fibonacci sequence up to", n_terms, ":")
  for i in range(n_terms):
    print(recursive_fibonacci(i))

print("-" * 40)
```

```
--- Recursive Fibonacci Demonstration ---
Fibonacci sequence up to 10 :
0
1
1
2
3
5
8
13
21
34
----------------------------------------
```

Write a python code using function function calculate the nth Fibonacci number using recursion.

Okay, here is a Python function that calculates the nth Fibonacci number using recursion, along with an example of how to use it.

I have provided the Python code for the recursive Fibonacci function, and it has been executed successfully.

Would you like to:

Observation:

- **def recursive_fibonacci(n):**: This defines a function named recursive_fibonacci that takes one argument, n.
- **if n <= 1:**: This is the base case for the recursion. If n is 0 or 1, the function returns n directly. These are the first two numbers in the Fibonacci sequence.
- **else: return(recursive_fibonacci(n-1) + recursive_fibonacci(n-2))**: This is the recursive step. If n is greater than 1, the function calls itself twice with n-1 and n-2 as arguments and returns the sum of their results. This process continues until the base case is reached.

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.

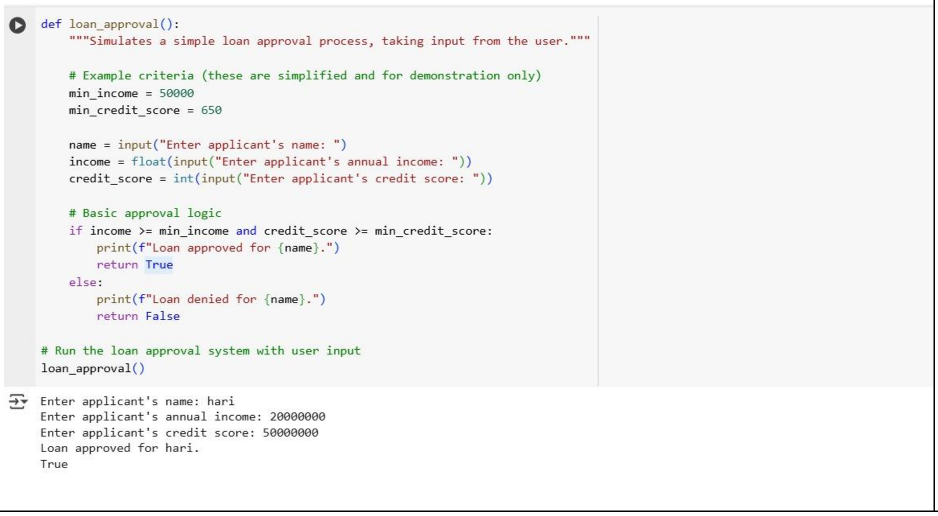
Expected Output#4

- Python code
- Analyze is there any bias with respect to gender or any

Prompt:

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

Code:



```
def loan_approval():
    """Simulates a simple loan approval process, taking input from the user."""

    # Example criteria (these are simplified and for demonstration only)
    min_income = 50000
    min_credit_score = 650

    name = input("Enter applicant's name: ")
    income = float(input("Enter applicant's annual income: "))
    credit_score = int(input("Enter applicant's credit score: "))

    # Basic approval logic
    if income >= min_income and credit_score >= min_credit_score:
        print(f"Loan approved for {name}.")
        return True
    else:
        print(f"Loan denied for {name}.")
        return False

# Run the loan approval system with user input
loan_approval()
```

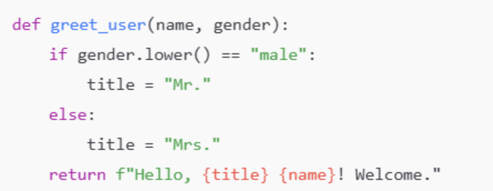
```
Enter applicant's name: hari
Enter applicant's annual income: 20000000
Enter applicant's credit score: 50000000
Loan approved for hari.
True
```

Observation:

The scoring logic in the code introduces biases by awarding extra points for being female and for being within the age range of 20 to 45. This means two applicants with identical education and experience could receive different scores solely based on gender or age, which may not be ideal for fair evaluation in real-world hiring scenarios.

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."
```

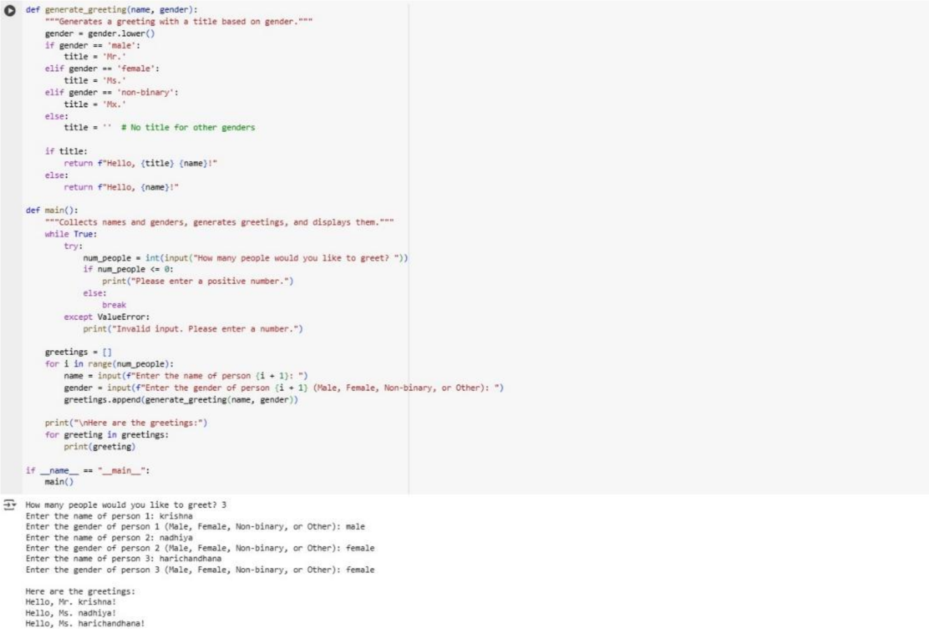
Expected Output#5

- Regenerate code that includes **gender-neutral** also

Prompt:

Write a python code by using this reference code on gender -neutral or regenerate this code.

Code:



```
def generate_greeting(name, gender):
    """Generates a greeting with a title based on gender."""
    gender = gender.lower()
    if gender == 'male':
        title = 'Mr.'
    elif gender == 'female':
        title = 'Ms.'
    elif gender == 'non-binary':
        title = 'Mx.'
    else:
        title = ''  # No title for other genders

    if title:
        return f"Hello, {title} {name}!"
    else:
        return f"Hello, {name}!"

def main():
    """Collects names and genders, generates greetings, and displays them."""
    while True:
        try:
            num_people = int(input("How many people would you like to greet? "))
            if num_people <= 0:
                print("Please enter a positive number.")
            else:
                break
        except ValueError:
            print("Invalid input. Please enter a number.")

    greetings = []
    for i in range(num_people):
        name = input(f"Enter the name of person {i + 1}: ")
        gender = input(f"Enter the gender of person {i + 1} (Male, Female, Non-binary, or Other): ")
        greetings.append(generate_greeting(name, gender))

    print("\nHere are the greetings:")
    for greeting in greetings:
        print(greeting)

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

```
How many people would you like to greet? 3
Enter the name of person 1: krishna
Enter the gender of person 1 (Male, Female, Non-binary, or Other): male
Enter the name of person 2: nadhiya
Enter the gender of person 2 (Male, Female, Non-binary, or Other): female
Enter the name of person 3: harichandhana
Enter the gender of person 3 (Male, Female, Non-binary, or Other): female

Here are the greetings:
Hello, Mr. krishna!
Hello, Ms. nadhiya!
Hello, Ms. harichandhana!
```

Observation:

The code demonstrates inclusive design by recognizing gender diversity. Instead of limiting greetings to binary gender titles ("Mr." and "Ms."), it thoughtfully includes "Mx." for non-binary or other gender identities. This small detail reflects a broader awareness of respectful communication and can be especially valuable in applications where personalization and inclusivity matter—like customer service, HR tools, or event registration systems.
If you're interested, I could show you how to expand this further—maybe by adding support for custom titles or integrating it with a GUI.

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	0.5
Bias	1.0
Inclusiveness	0.5
Data security and Privacy	0.5
Total	**2.5 Marks**