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:

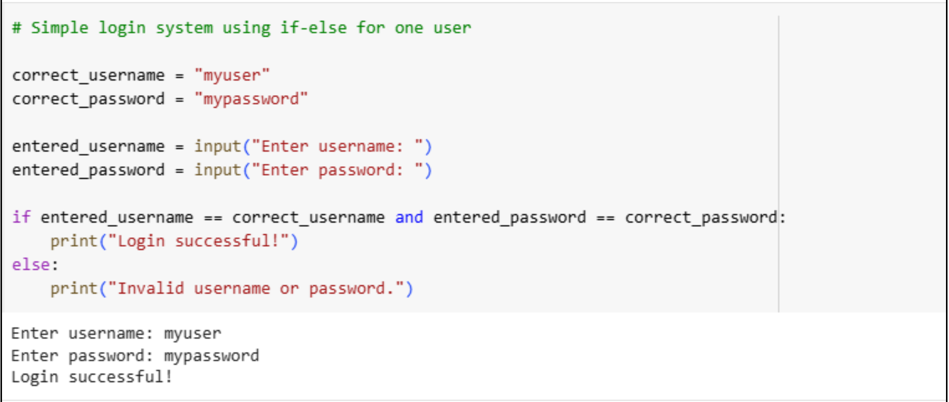
- Identify and avoid insecure coding patterns generated by AI tools.
- Detect and analyze potential bias or discriminatory logic in AI-generated outputs.
- Evaluate originality and licensing concerns in reused AI-generated code.
- Understand the importance of explainability and transparency in AI-assisted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

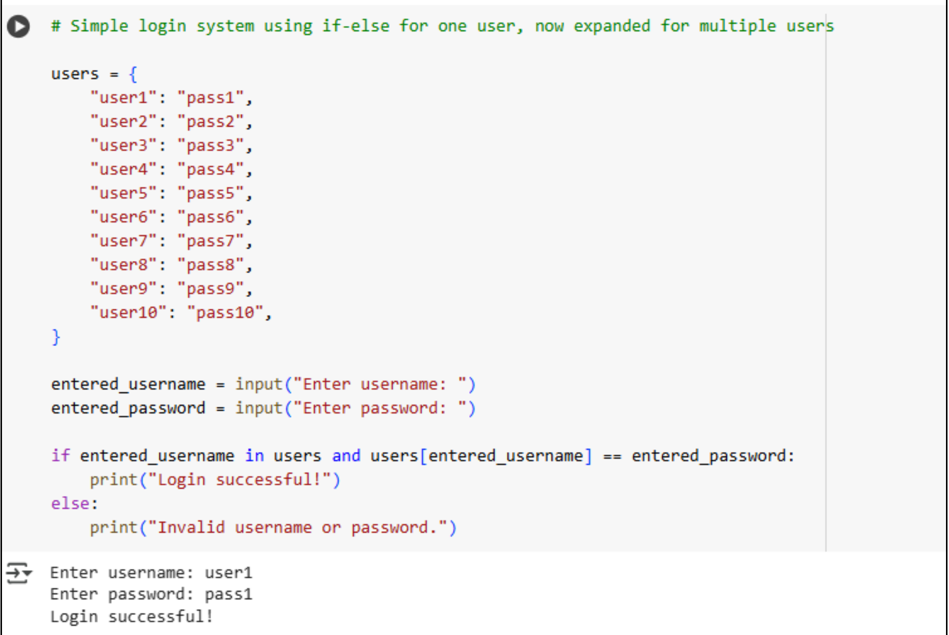
Expected Output#1

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



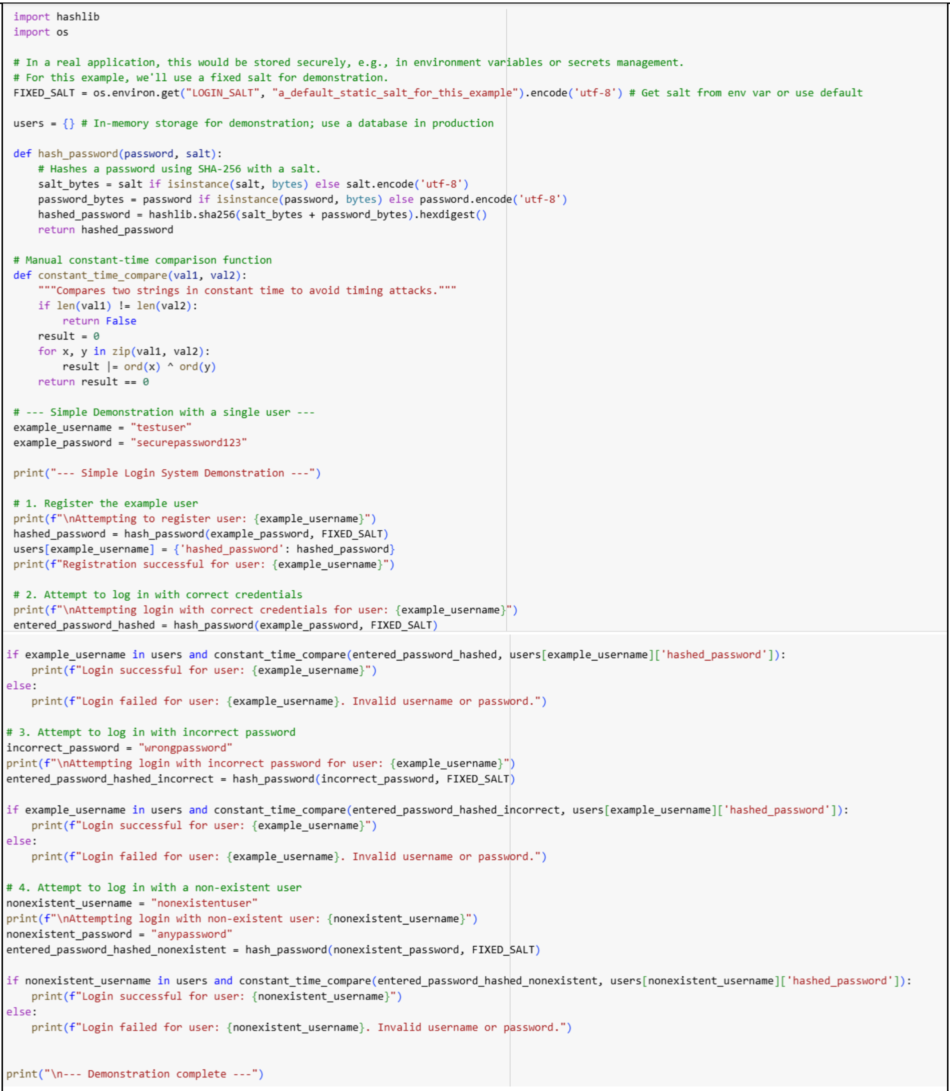
```python
# Simple login system using if-else for one user

correct_username = "myuser"
correct_password = "mypassword"

entered_username = input("Enter username: ")
entered_password = input("Enter password: ")

if entered_username == correct_username and entered_password == correct_password:
    print("Login successful!")
else:
    print("Invalid username or password.")
```

```
Enter username: myuser
Enter password: mypassword
Login successful!
```

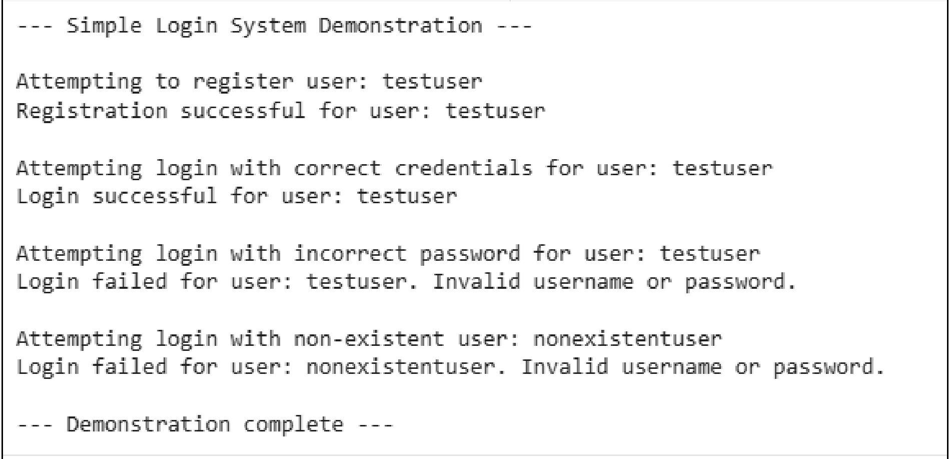


```python
# Simple login system using if-else for one user, now expanded for multiple users

users = {
    "user1": "pass1",
    "user2": "pass2",
    "user3": "pass3",
    "user4": "pass4",
    "user5": "pass5",
    "user6": "pass6",
    "user7": "pass7",
    "user8": "pass8",
    "user9": "pass9",
    "user10": "pass10",
}

entered_username = input("Enter username: ")
entered_password = input("Enter password: ")

if entered_username in users and users[entered_username] == entered_password:
    print("Login successful!")
else:
    print("Invalid username or password.")
```

```
Enter username: user1
Enter password: pass1
Login successful!
```



```python
import hashlib
import os

# In a real application, this would be stored securely, e.g., in environment variables or secrets management.
# For this example, we'll use a fixed salt for demonstration.
FIXED_SALT = os.environ.get("LOGIN_SALT", "a_default_static_salt_for_this_example").encode('utf-8') # Get salt from env var or use default

users = {} # In-memory storage for demonstration; use a database in production

def hash_password(password, salt):
    # Hashes a password using SHA-256 with a salt.
    salt_bytes = salt if isinstance(salt, bytes) else salt.encode('utf-8')
    password_bytes = password if isinstance(password, bytes) else password.encode('utf-8')
    hashed_password = hashlib.sha256(salt_bytes + password_bytes).hexdigest()
    return hashed_password

# Manual constant-time comparison function
def constant_time_compare(val1, val2):
    """Compares two strings in constant time to avoid timing attacks."""
    if len(val1) != len(val2):
        return False
    result = 0
    for x, y in zip(val1, val2):
        result |= ord(x) ^ ord(y)
    return result == 0

# --- Simple Demonstration with a single user ---
example_username = "testuser"
example_password = "securepassword123"

print("--- Simple Login System Demonstration ---")

# 1. Register the example user
print(f"\nAttempting to register user: {example_username}")
hashed_password = hash_password(example_password, FIXED_SALT)
users[example_username] = {'hashed_password': hashed_password}
print(f"Registration successful for user: {example_username}")

# 2. Attempt to log in with correct credentials
print(f"\nAttempting login with correct credentials for user: {example_username}")
entered_password_hashed = hash_password(example_password, FIXED_SALT)

if example_username in users and constant_time_compare(entered_password_hashed, users[example_username]['hashed_password']):
    print(f"Login successful for user: {example_username}")
else:
    print(f"Login failed for user: {example_username}. Invalid username or password.")

# 3. Attempt to log in with incorrect password
incorrect_password = "wrongpassword"
print(f"\nAttempting login with incorrect password for user: {example_username}")
entered_password_hashed_incorrect = hash_password(incorrect_password, FIXED_SALT)

if example_username in users and constant_time_compare(entered_password_hashed_incorrect, users[example_username]['hashed_password']):
    print(f"Login successful for user: {example_username}")
else:
    print(f"Login failed for user: {example_username}. Invalid username or password.")

# 4. Attempt to log in with a non-existent user
nonexistent_username = "nonexistentuser"
print(f"\nAttempting login with non-existent user: {nonexistent_username}")
nonexistent_password = "anypassword"
entered_password_hashed_nonexistent = hash_password(nonexistent_password, FIXED_SALT)

if nonexistent_username in users and constant_time_compare(entered_password_hashed_nonexistent, users[nonexistent_username]['hashed_password']):
    print(f"Login successful for user: {nonexistent_username}")
else:
    print(f"Login failed for user: {nonexistent_username}. Invalid username or password.")


print("\n--- Demonstration complete ---")
```



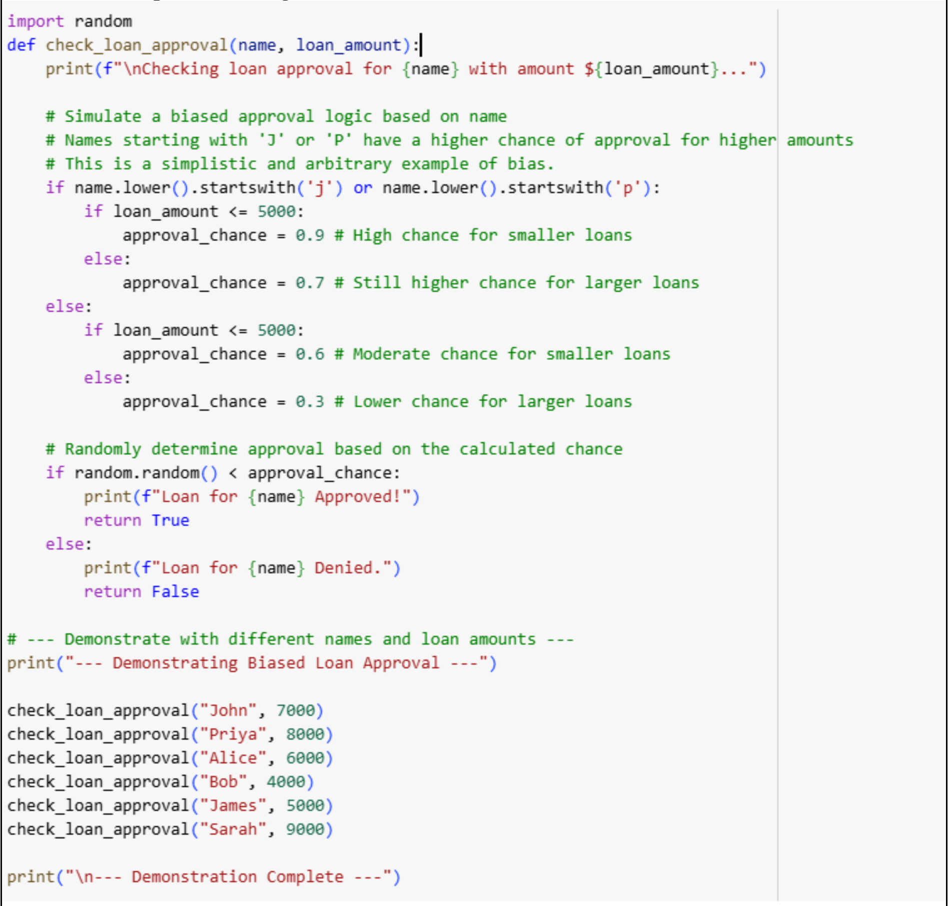
```
--- Simple Login System Demonstration ---

Attempting to register user: testuser
Registration successful for user: testuser

Attempting login with correct credentials for user: testuser
Login successful for user: testuser

Attempting login with incorrect password for user: testuser
Login failed for user: testuser. Invalid username or password.

Attempting login with non-existent user: nonexistentuser
Login failed for user: nonexistentuser. Invalid username or password.

--- Demonstration complete ---
```

Task Description#2 (Bias)

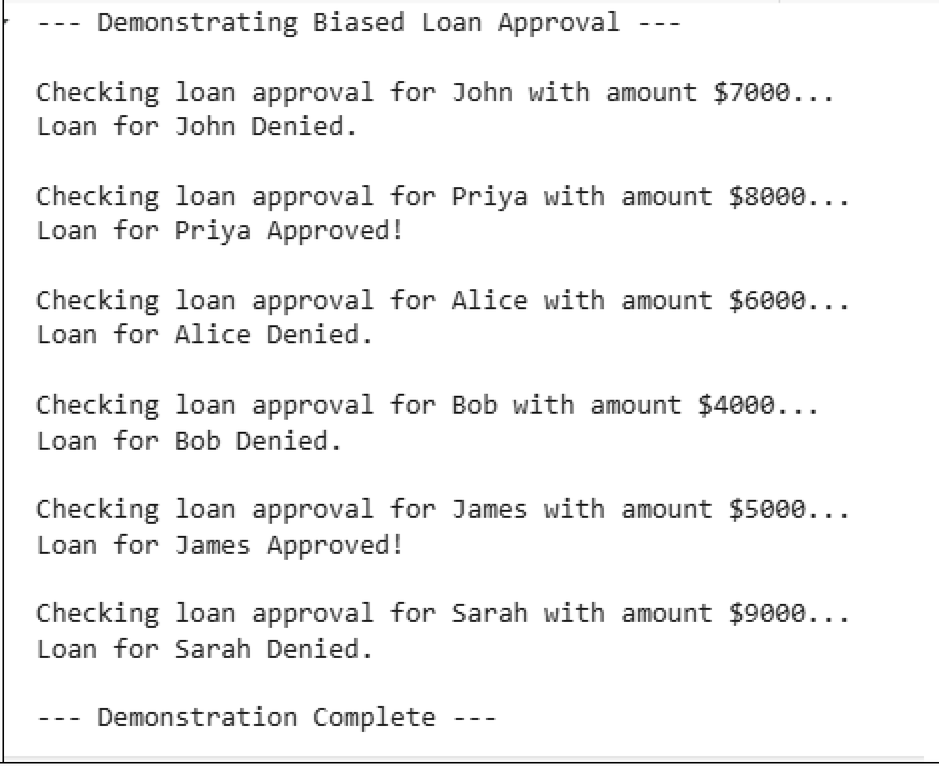
- Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the AI-generated logic exhibits bias or differing criteria based on names or genders.

Expected Output#2

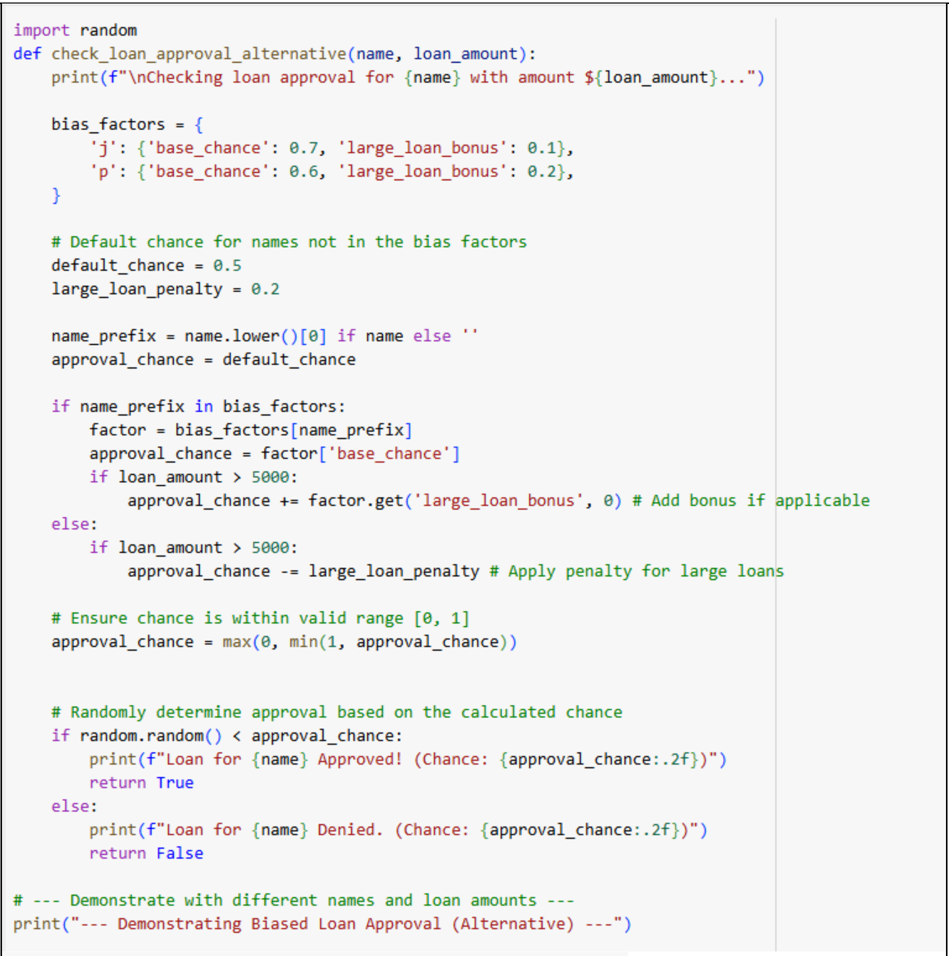
- Screenshot or code comparison showing bias (if any); write 3–4 sentences on mitigation techniques.



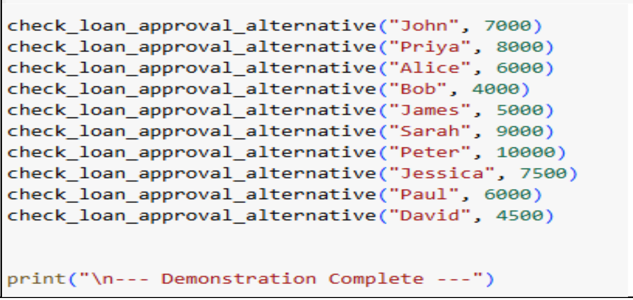
```python
import random
def check_loan_approval(name, loan_amount):
    print(f"\nChecking loan approval for {name} with amount ${loan_amount}...")

    # Simulate a biased approval logic based on name
    # Names starting with 'J' or 'P' have a higher chance of approval for higher amounts
    # This is a simplistic and arbitrary example of bias.
    if name.lower().startswith('j') or name.lower().startswith('p'):
        if loan_amount <= 5000:
            approval_chance = 0.9 # High chance for smaller loans
        else:
            approval_chance = 0.7 # Still higher chance for larger loans
    else:
        if loan_amount <= 5000:
            approval_chance = 0.6 # Moderate chance for smaller loans
        else:
            approval_chance = 0.3 # Lower chance for larger loans

    # Randomly determine approval based on the calculated chance
    if random.random() < approval_chance:
        print(f"Loan for {name} Approved!")
        return True
    else:
        print(f"Loan for {name} Denied.")
        return False

# --- Demonstrate with different names and loan amounts ---
print("--- Demonstrating Biased Loan Approval ---")

check_loan_approval("John", 7000)
check_loan_approval("Priya", 8000)
check_loan_approval("Alice", 6000)
check_loan_approval("Bob", 4000)
check_loan_approval("James", 5000)
check_loan_approval("Sarah", 9000)

print("\n--- Demonstration Complete ---")
```



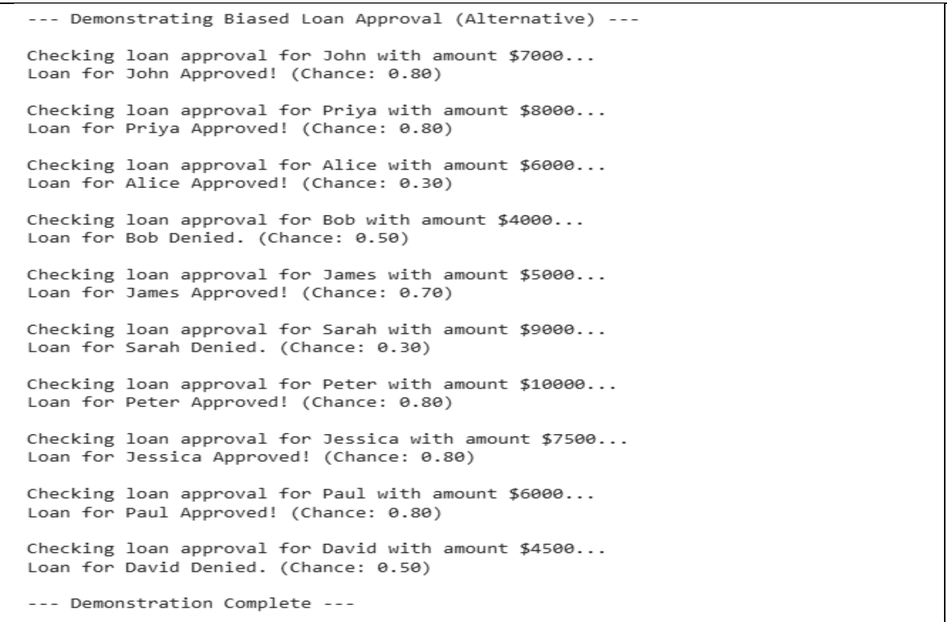
```
--- Demonstrating Biased Loan Approval ---

Checking loan approval for John with amount $7000...
Loan for John Denied.

Checking loan approval for Priya with amount $8000...
Loan for Priya Approved!

Checking loan approval for Alice with amount $6000...
Loan for Alice Denied.

Checking loan approval for Bob with amount $4000...
Loan for Bob Denied.

Checking loan approval for James with amount $5000...
Loan for James Approved!

Checking loan approval for Sarah with amount $9000...
Loan for Sarah Denied.

--- Demonstration Complete ---
```



```python
import random
def check_loan_approval_alternative(name, loan_amount):
    print(f"\nChecking loan approval for {name} with amount ${loan_amount}...")

    bias_factors = {
        'j': {'base_chance': 0.7, 'large_loan_bonus': 0.1},
        'p': {'base_chance': 0.6, 'large_loan_bonus': 0.2},
    }

    # Default chance for names not in the bias factors
    default_chance = 0.5
    large_loan_penalty = 0.2

    name_prefix = name.lower()[0] if name else ''
    approval_chance = default_chance

    if name_prefix in bias_factors:
        factor = bias_factors[name_prefix]
        approval_chance = factor['base_chance']
        if loan_amount > 5000:
            approval_chance += factor.get('large_loan_bonus', 0) # Add bonus if applicable
    else:
        if loan_amount > 5000:
            approval_chance -= large_loan_penalty # Apply penalty for large loans

    # Ensure chance is within valid range [0, 1]
    approval_chance = max(0, min(1, approval_chance))


    # Randomly determine approval based on the calculated chance
    if random.random() < approval_chance:
        print(f"Loan for {name} Approved! (Chance: {approval_chance:.2f})")
        return True
    else:
        print(f"Loan for {name} Denied. (Chance: {approval_chance:.2f})")
        return False

# --- Demonstrate with different names and loan amounts ---
print("--- Demonstrating Biased Loan Approval (Alternative) ---")
```



```python
check_loan_approval_alternative("John", 7000)
check_loan_approval_alternative("Priya", 8000)
check_loan_approval_alternative("Alice", 6000)
check_loan_approval_alternative("Bob", 4000)
check_loan_approval_alternative("James", 5000)
check_loan_approval_alternative("Sarah", 9000)
check_loan_approval_alternative("Peter", 10000)
check_loan_approval_alternative("Jessica", 7500)
check_loan_approval_alternative("Paul", 6000)
check_loan_approval_alternative("David", 4500)


print("\n--- Demonstration Complete ---")
```



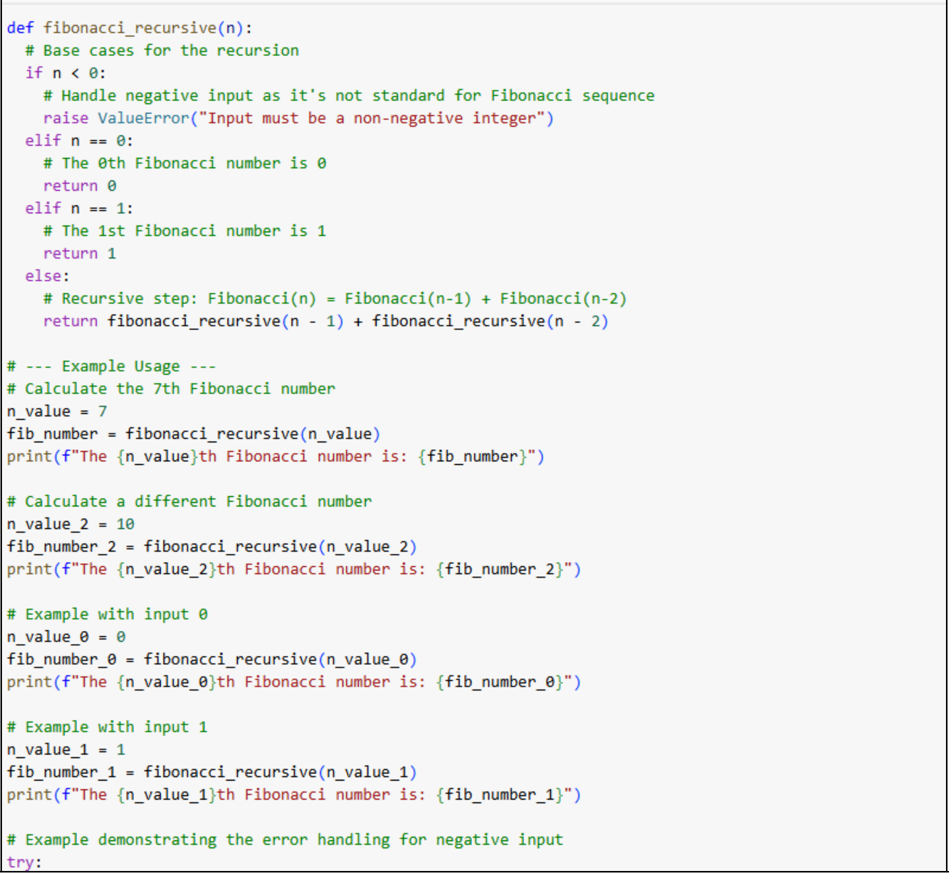
```
--- Demonstrating Biased Loan Approval (Alternative) ---

Checking loan approval for John with amount $7000...
Loan for John Approved! (Chance: 0.80)

Checking loan approval for Priya with amount $8000...
Loan for Priya Approved! (Chance: 0.80)

Checking loan approval for Alice with amount $6000...
Loan for Alice Approved! (Chance: 0.30)

Checking loan approval for Bob with amount $4000...
Loan for Bob Denied. (Chance: 0.50)

Checking loan approval for James with amount $5000...
Loan for James Approved! (Chance: 0.70)

Checking loan approval for Sarah with amount $9000...
Loan for Sarah Denied. (Chance: 0.30)

Checking loan approval for Peter with amount $10000...
Loan for Peter Approved! (Chance: 0.80)

Checking loan approval for Jessica with amount $7500...
Loan for Jessica Approved! (Chance: 0.80)

Checking loan approval for Paul with amount $6000...
Loan for Paul Approved! (Chance: 0.80)

Checking loan approval for David with amount $4500...
Loan for David Denied. (Chance: 0.50)

--- Demonstration Complete ---
```

Task Description#3 (Transparency)

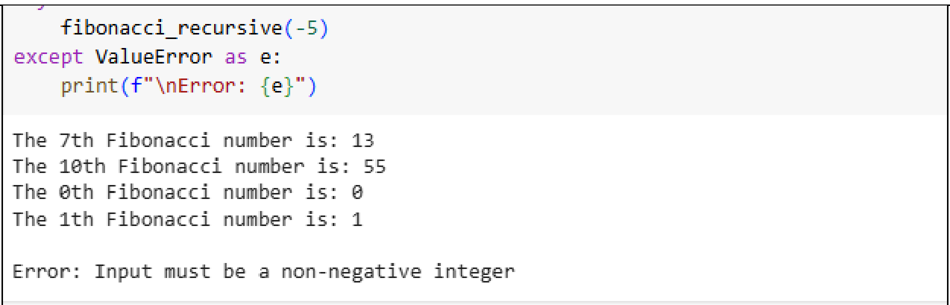
- Write prompt to write function calculate the nth Fibonacci number using recursion and generate comments and explain code docum

Expected Output#3

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

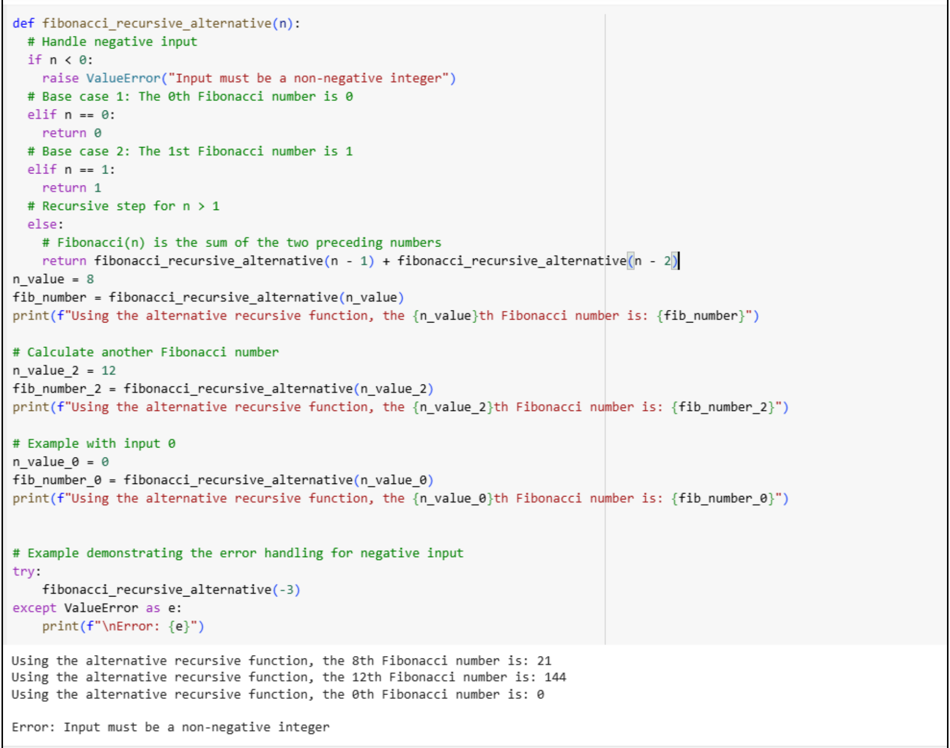


```python
def fibonacci_recursive(n):
  # Base cases for the recursion
  if n < 0:
    # Handle negative input as it's not standard for Fibonacci sequence
    raise ValueError("Input must be a non-negative integer")
  elif n == 0:
    # The 0th Fibonacci number is 0
    return 0
  elif n == 1:
    # The 1st Fibonacci number is 1
    return 1
  else:
    # Recursive step: Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

# --- Example Usage ---
# Calculate the 7th Fibonacci number
n_value = 7
fib_number = fibonacci_recursive(n_value)
print(f"The {n_value}th Fibonacci number is: {fib_number}")

# Calculate a different Fibonacci number
n_value_2 = 10
fib_number_2 = fibonacci_recursive(n_value_2)
print(f"The {n_value_2}th Fibonacci number is: {fib_number_2}")

# Example with input 0
n_value_0 = 0
fib_number_0 = fibonacci_recursive(n_value_0)
print(f"The {n_value_0}th Fibonacci number is: {fib_number_0}")

# Example with input 1
n_value_1 = 1
fib_number_1 = fibonacci_recursive(n_value_1)
print(f"The {n_value_1}th Fibonacci number is: {fib_number_1}")

# Example demonstrating the error handling for negative input
try:
```



```
    fibonacci_recursive(-5)
except ValueError as e:
    print(f"\nError: {e}")
```

```
The 7th Fibonacci number is: 13
The 10th Fibonacci number is: 55
The 0th Fibonacci number is: 0
The 1th Fibonacci number is: 1

Error: Input must be a non-negative integer
```



```
def fibonacci_recursive_alternative(n):
  # Handle negative input
  if n < 0:
    raise ValueError("Input must be a non-negative integer")
  # Base case 1: The 0th Fibonacci number is 0
  elif n == 0:
    return 0
  # Base case 2: The 1st Fibonacci number is 1
  elif n == 1:
    return 1
  # Recursive step for n > 1
  else:
    # Fibonacci(n) is the sum of the two preceding numbers
    return fibonacci_recursive_alternative(n - 1) + fibonacci_recursive_alternative(n - 2)
n_value = 8
fib_number = fibonacci_recursive_alternative(n_value)
print(f"Using the alternative recursive function, the {n_value}th Fibonacci number is: {fib_number}")

# Calculate another Fibonacci number
n_value_2 = 12
fib_number_2 = fibonacci_recursive_alternative(n_value_2)
print(f"Using the alternative recursive function, the {n_value_2}th Fibonacci number is: {fib_number_2}")

# Example with input 0
n_value_0 = 0
fib_number_0 = fibonacci_recursive_alternative(n_value_0)
print(f"Using the alternative recursive function, the {n_value_0}th Fibonacci number is: {fib_number_0}")


# Example demonstrating the error handling for negative input
try:
    fibonacci_recursive_alternative(-3)
except ValueError as e:
    print(f"\nError: {e}")
```

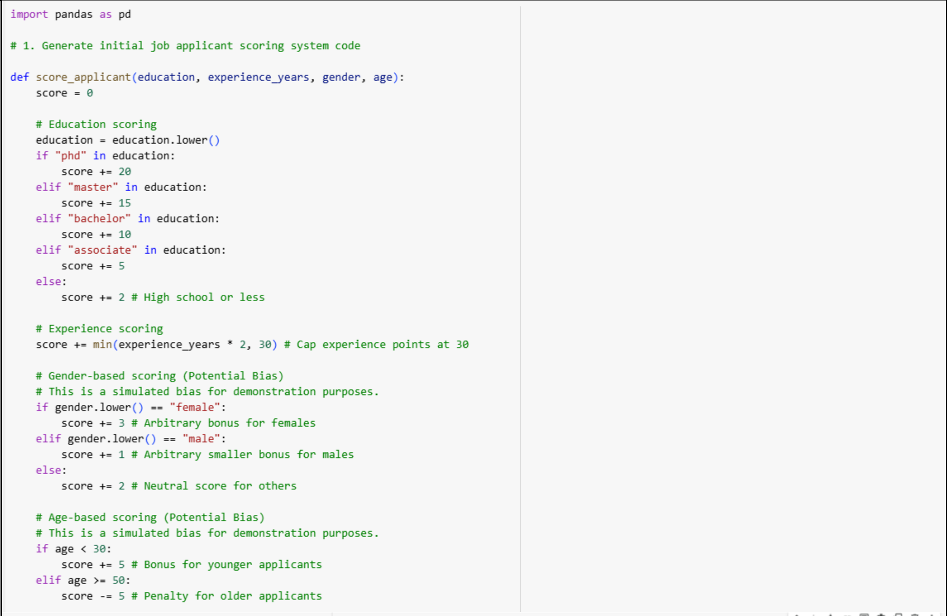
```
Using the alternative recursive function, the 8th Fibonacci number is: 21
Using the alternative recursive function, the 12th Fibonacci number is: 144
Using the alternative recursive function, the 0th Fibonacci number is: 0

Error: Input must be a non-negative integer
```

Task Description#4 (Bias)

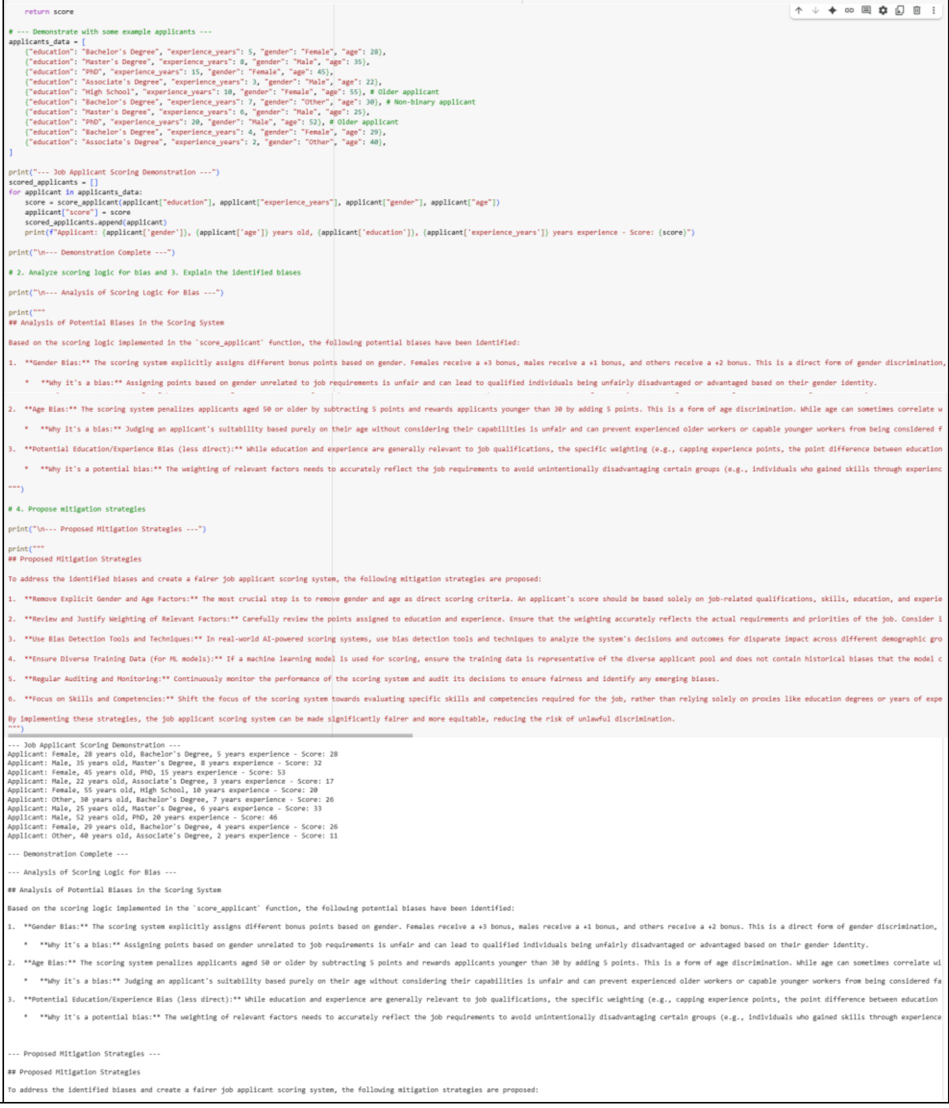
- Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings.

Expected Output#4

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

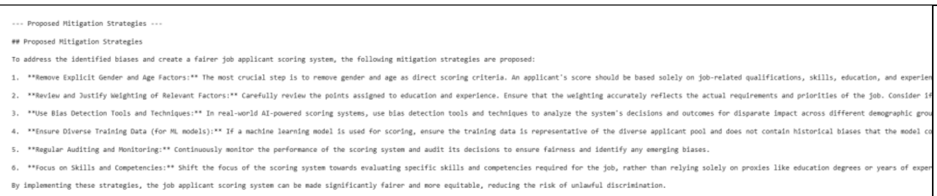


```python
import pandas as pd

# 1. Generate initial job applicant scoring system code

def score_applicant(education, experience_years, gender, age):
    score = 0

    # Education scoring
    education = education.lower()
    if "phd" in education:
        score += 20
    elif "master" in education:
        score += 15
    elif "bachelor" in education:
        score += 10
    elif "associate" in education:
        score += 5
    else:
        score += 2 # High school or less

    # Experience scoring
    score += min(experience_years * 2, 30) # Cap experience points at 30

    # Gender-based scoring (Potential Bias)
    # This is a simulated bias for demonstration purposes.
    if gender.lower() == "female":
        score += 3 # Arbitrary bonus for females
    elif gender.lower() == "male":
        score += 1 # Arbitrary smaller bonus for males
    else:
        score += 2 # Neutral score for others

    # Age-based scoring (Potential Bias)
    # This is a simulated bias for demonstration purposes.
    if age < 30:
        score += 5 # Bonus for younger applicants
    elif age >= 50:
        score -= 5 # Penalty for older applicants
```



```python
    return score

# --- Demonstrate with some example applicants ---
applicants_data = [
    {"education": "Bachelor's Degree", "experience_years": 5, "gender": "Female", "age": 28},
    {"education": "Master's Degree", "experience_years": 8, "gender": "Male", "age": 35},
    {"education": "PhD", "experience_years": 15, "gender": "Female", "age": 45},
    {"education": "Associate's Degree", "experience_years": 3, "gender": "Male", "age": 22},
    {"education": "High School", "experience_years": 10, "gender": "Female", "age": 55}, # Older applicant
    {"education": "Bachelor's Degree", "experience_years": 7, "gender": "Other", "age": 30}, # Non-binary applicant
    {"education": "Master's Degree", "experience_years": 6, "gender": "Male", "age": 25},
    {"education": "PhD", "experience_years": 20, "gender": "Male", "age": 52}, # Older applicant
    {"education": "Bachelor's Degree", "experience_years": 4, "gender": "Female", "age": 29},
    {"education": "Associate's Degree", "experience_years": 2, "gender": "Other", "age": 40},
]

print("--- Job Applicant Scoring Demonstration ---")
scored_applicants = []
for applicant in applicants_data:
    score = score_applicant(applicant["education"], applicant["experience_years"], applicant["gender"], applicant["age"])
    applicant["score"] = score
    scored_applicants.append(applicant)
    print(f"Applicant: {applicant['gender']}, {applicant['age']} years old, {applicant['education']}, {applicant['experience_years']} years experience - Score: {score}")

print("\n--- Demonstration Complete ---")

# 2. Analyze scoring logic for bias and 3. Explain the identified biases

print("\n--- Analysis of Scoring Logic for Bias ---")

print("""
## Analysis of Potential Biases in the Scoring System

Based on the scoring logic implemented in the `score_applicant` function, the following potential biases have been identified:

1.  **Gender Bias:** The scoring system explicitly assigns different bonus points based on gender. Females receive a +3 bonus, males receive a +1 bonus, and others receive a +2 bonus. This is a direct form of gender discrimination,

    *   **Why it's a bias:** Assigning points based on gender unrelated to job requirements is unfair and can lead to qualified individuals being unfairly disadvantaged or advantaged based on their gender identity.

2.  **Age Bias:** The scoring system penalizes applicants aged 50 or older by subtracting 5 points and rewards applicants younger than 30 by adding 5 points. This is a form of age discrimination. While age can sometimes correlate w

    *   **Why it's a bias:** Judging an applicant's suitability based purely on their age without considering their capabilities is unfair and can prevent experienced older workers or capable younger workers from being considered f

3.  **Potential Education/Experience Bias (less direct):** While education and experience are generally relevant to job qualifications, the specific weighting (e.g., capping experience points, the point difference between education

    *   **Why it's a potential bias:** The weighting of relevant factors needs to accurately reflect the job requirements to avoid unintentionally disadvantaging certain groups (e.g., individuals who gained skills through experienc

""")

# 4. Propose mitigation strategies

print("\n--- Proposed Mitigation Strategies ---")

print("""
## Proposed Mitigation Strategies

To address the identified biases and create a fairer job applicant scoring system, the following mitigation strategies are proposed:

1.  **Remove Explicit Gender and Age Factors:** The most crucial step is to remove gender and age as direct scoring criteria. An applicant's score should be based solely on job-related qualifications, skills, education, and experie

2.  **Review and Justify Weighting of Relevant Factors:** Carefully review the points assigned to education and experience. Ensure that the weighting accurately reflects the actual requirements and priorities of the job. Consider i

3.  **Use Bias Detection Tools and Techniques:** In real-world AI-powered scoring systems, use bias detection tools and techniques to analyze the system's decisions and outcomes for disparate impact across different demographic gro

4.  **Ensure Diverse Training Data (for ML models):** If a machine learning model is used for scoring, ensure the training data is representative of the diverse applicant pool and does not contain historical biases that the model c

5.  **Regular Auditing and Monitoring:** Continuously monitor the performance of the scoring system and audit its decisions to ensure fairness and identify any emerging biases.

6.  **Focus on Skills and Competencies:** Shift the focus of the scoring system towards evaluating specific skills and competencies required for the job, rather than relying solely on proxies like education degrees or years of expe

By implementing these strategies, the job applicant scoring system can be made significantly fairer and more equitable, reducing the risk of unlawful discrimination.
""")
```

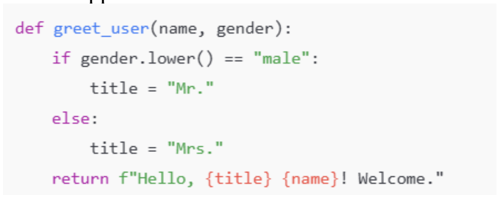
```
--- Job Applicant Scoring Demonstration ---
Applicant: Female, 28 years old, Bachelor's Degree, 5 years experience - Score: 28
Applicant: Male, 35 years old, Master's Degree, 8 years experience - Score: 32
Applicant: Female, 45 years old, PhD, 15 years experience - Score: 53
Applicant: Male, 22 years old, Associate's Degree, 3 years experience - Score: 17
Applicant: Female, 55 years old, High School, 10 years experience - Score: 20
Applicant: Other, 30 years old, Bachelor's Degree, 7 years experience - Score: 26
Applicant: Male, 25 years old, Master's Degree, 6 years experience - Score: 33
Applicant: Male, 52 years old, PhD, 20 years experience - Score: 46
Applicant: Female, 29 years old, Bachelor's Degree, 4 years experience - Score: 26
Applicant: Other, 40 years old, Associate's Degree, 2 years experience - Score: 11

--- Demonstration Complete ---

--- Analysis of Scoring Logic for Bias ---

## Analysis of Potential Biases in the Scoring System

Based on the scoring logic implemented in the `score_applicant` function, the following potential biases have been identified:

1.  **Gender Bias:** The scoring system explicitly assigns different bonus points based on gender. Females receive a +3 bonus, males receive a +1 bonus, and others receive a +2 bonus. This is a direct form of gender discrimination,

    *   **Why it's a bias:** Assigning points based on gender unrelated to job requirements is unfair and can lead to qualified individuals being unfairly disadvantaged or advantaged based on their gender identity.

2.  **Age Bias:** The scoring system penalizes applicants aged 50 or older by subtracting 5 points and rewards applicants younger than 30 by adding 5 points. This is a form of age discrimination. While age can sometimes correlate wi

    *   **Why it's a bias:** Judging an applicant's suitability based purely on their age without considering their capabilities is unfair and can prevent experienced older workers or capable younger workers from being considered fa

3.  **Potential Education/Experience Bias (less direct):** While education and experience are generally relevant to job qualifications, the specific weighting (e.g., capping experience points, the point difference between education

    *   **Why it's a potential bias:** The weighting of relevant factors needs to accurately reflect the job requirements to avoid unintentionally disadvantaging certain groups (e.g., individuals who gained skills through experience


--- Proposed Mitigation Strategies ---

## Proposed Mitigation Strategies

To address the identified biases and create a fairer job applicant scoring system, the following mitigation strategies are proposed:
```



```
--- Proposed Mitigation Strategies ---

## Proposed Mitigation Strategies

To address the identified biases and create a fairer job applicant scoring system, the following mitigation strategies are proposed:

1.  **Remove Explicit Gender and Age Factors:** The most crucial step is to remove gender and age as direct scoring criteria. An applicant's score should be based solely on job-related qualifications, skills, education, and experien

2.  **Review and Justify Weighting of Relevant Factors:** Carefully review the points assigned to education and experience. Ensure that the weighting accurately reflects the actual requirements and priorities of the job. Consider if

3.  **Use Bias Detection Tools and Techniques:** In real-world AI-powered scoring systems, use bias detection tools and techniques to analyze the system's decisions and outcomes for disparate impact across different demographic grou

4.  **Ensure Diverse Training Data (for ML models):** If a machine learning model is used for scoring, ensure the training data is representative of the diverse applicant pool and does not contain historical biases that the model co

5.  **Regular Auditing and Monitoring:** Continuously monitor the performance of the scoring system and audit its decisions to ensure fairness and identify any emerging biases.

6.  **Focus on Skills and Competencies:** Shift the focus of the scoring system towards evaluating specific skills and competencies required for the job, rather than relying solely on proxies like education degrees or years of exper

By implementing these strategies, the job applicant scoring system can be made significantly fairer and more equitable, reducing the risk of unlawful discrimination.
```

Task Description#5 (Inclusiveness)

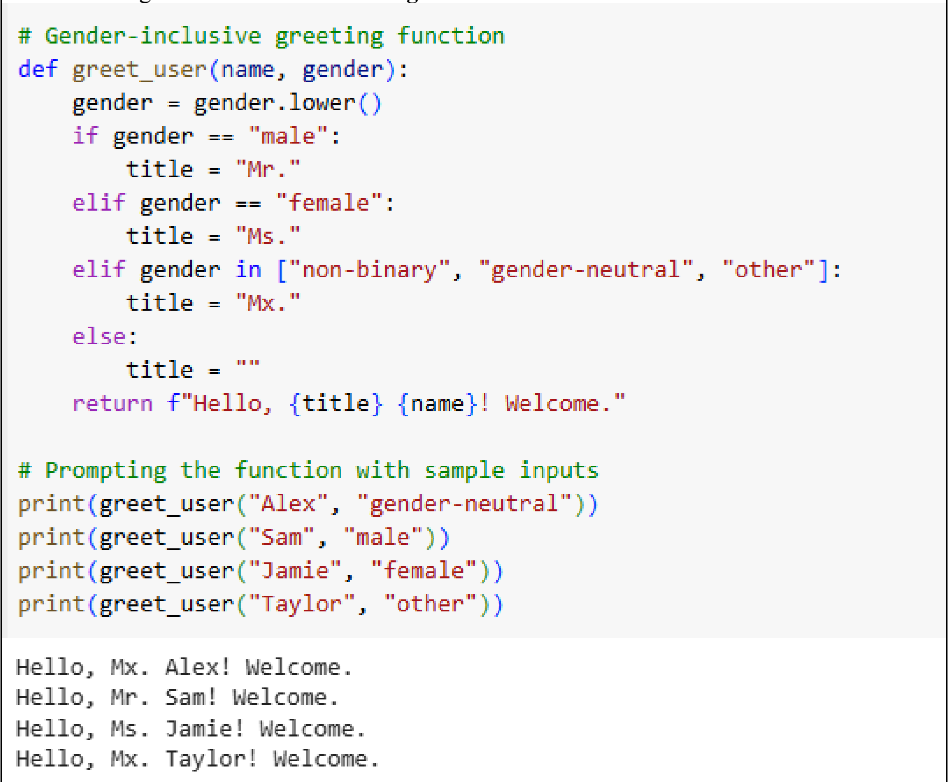
- Code Snippet



```python
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



```python
# Gender-inclusive greeting function
def greet_user(name, gender):
    gender = gender.lower()
    if gender == "male":
        title = "Mr."
    elif gender == "female":
        title = "Ms."
    elif gender in ["non-binary", "gender-neutral", "other"]:
        title = "Mx."
    else:
        title = ""
    return f"Hello, {title} {name}! Welcome."

# Prompting the function with sample inputs
print(greet_user("Alex", "gender-neutral"))
print(greet_user("Sam", "male"))
print(greet_user("Jamie", "female"))
print(greet_user("Taylor", "other"))
```

```
Hello, Mx. Alex! Welcome.
Hello, Mr. Sam! Welcome.
Hello, Ms. Jamie! Welcome.
Hello, Mx. Taylor! Welcome.
```

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**		