AI ASSISSTANT CODING LAB 5

ROLL NO :2303A52291 BATCH :42

NAME: M. SHASHIDHAR

Task 1: Privacy and Data Security in AI-Generated Code

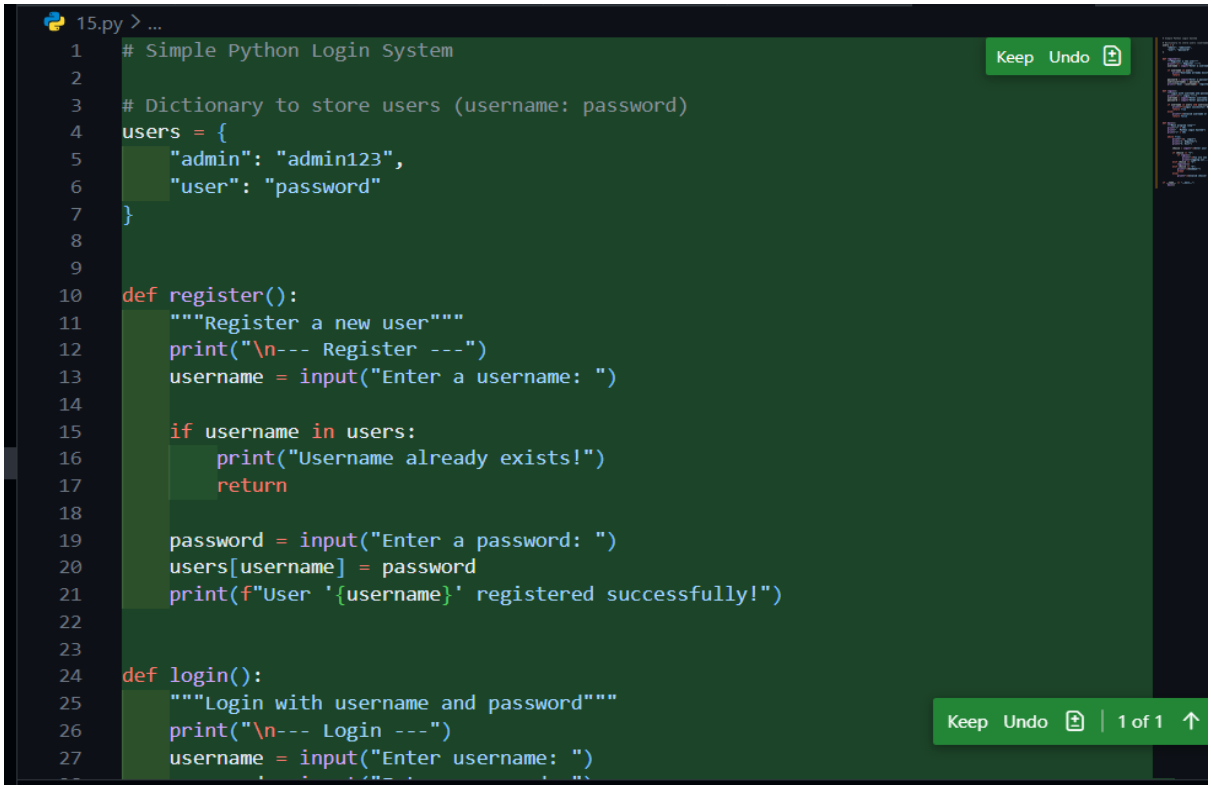
Objective:

To analyze AI-generated login code for security flaws and revise it to follow responsible and secure coding practices.

Prompt Used for AI Tool:

Generate a simple Python login system for username and password authentication.

Generated code:



15.py > ...

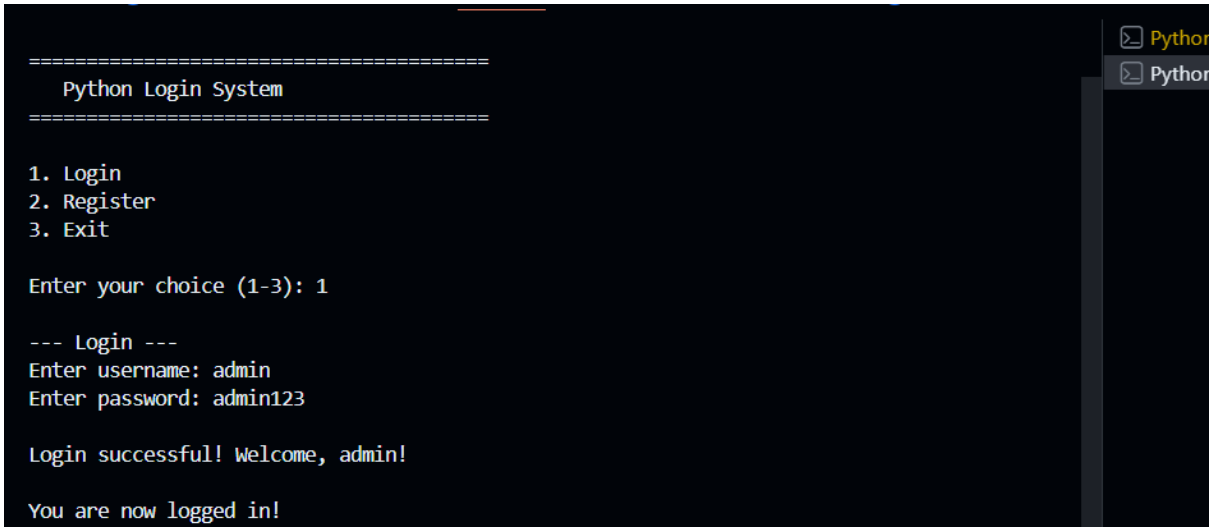
```python
def login():
    """Login with username and password"""
    print("\n--- Login ---")
    username = input("Enter username: ")
    password = input("Enter password: ")

    if username in users and users[username] == password:
        print(f"\nLogin successful! Welcome, {username}!")
        return True
    else:
        print("\nInvalid username or password!")
        return False


def main():
    """Main program loop"""
    print("=" * 40)
    print("   Python Login System")
    print("=" * 40)

    while True:
        print("\n1. Login")
        print("2. Register")
        print("3. Exit")

        choice = input("\nEnter your choice (1-3): ")
```

Keep Undo | 1 of

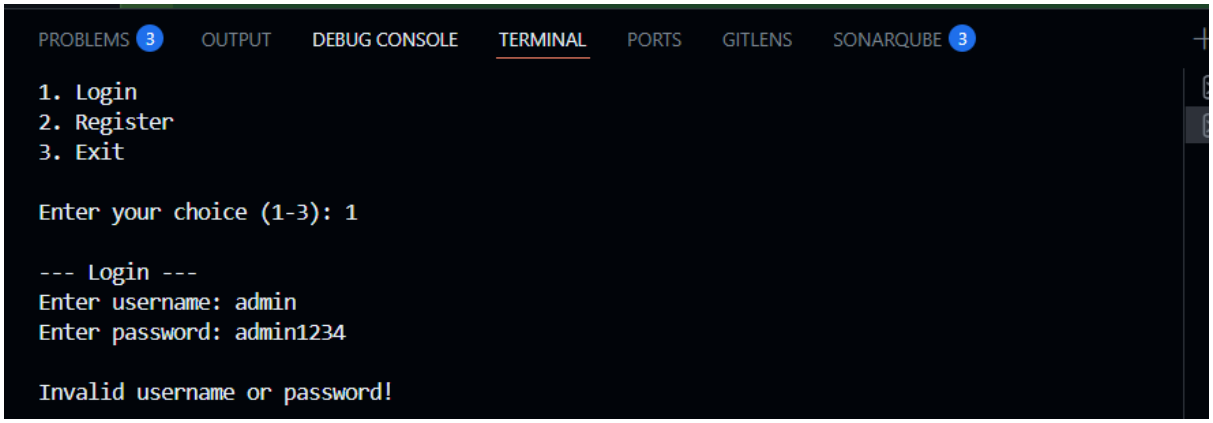
```python
        choice = input("\nEnter your choice (1-3): ")

        if choice == "1":
            if login():
                print("\nYou are now logged in!")
                print("Logging out...")
        elif choice == "2":
            register()
        elif choice == "3":
            print("\nGoodbye!")
            break
        else:
            print("\nInvalid choice! Please try again.")


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

Keep Undo | 1

Output:



For wrong credentials



Security Issues Identified

1. Hardcoded Credentials
    - Username and password are directly written in the source code.
    - Anyone with access to the code can see the credentials.
2. Plain-Text Password Comparison
    - Password is stored and compared in plain text.
    - Vulnerable to password leakage and attacks.
3. No Input Validation
    - No checks for empty or invalid input.
4. No Password Hashing
    - Violates standard authentication security practices.

Improved code with security

15.py > ...

```python
import hashlib

# Storing hashed password instead of plain text
stored_username = "admin"
stored_password_hash = hashlib.sha256("admin123".encode()).hexdigest()

input_username = input("Enter username: ").strip()
input_password = input("Enter password: ").strip()

# Hashing user-entered password
input_password_hash = hashlib.sha256(input_password.encode()).hexdigest()

if input_username == stored_username and input_password_hash == stored_password_hash:
    print("Login successful")
else:
    print("Invalid credentials")

```

Output for correct credentials

```
PROBLEMS 3  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SONARQUBE 3  Python

PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python
exe c:/Users/s9409/Downloads/aiassitantcoding/15.py
Enter username: admin
Enter password: admin123
Login successful
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

For wrong credentials

```
PROBLEMS 3  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SONARQUBE 3  Python

PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.
exe c:/Users/s9409/Downloads/aiassitantcoding/15.py
Enter username: admin
Enter password: test
Invalid credentials
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Justification for improved code

1. **Password Hashing**
    - SHA-256 hashing is used to prevent storing passwords in plain text.
    - Even if data is exposed, actual passwords are protected.
2. **Reduced Risk of Credential Theft**
    - Password comparison is done using hashes, not raw strings.
3. **Basic Input Sanitization**

- .strip() removes accidental spaces and avoids simple input errors.

4. **Ethical Responsibility**
   - Developers must not blindly trust AI-generated code.
   - Human review is essential to ensure privacy, security, and responsible AI usage.

Task 2: Bias Detection in AI-Generated Decision Systems

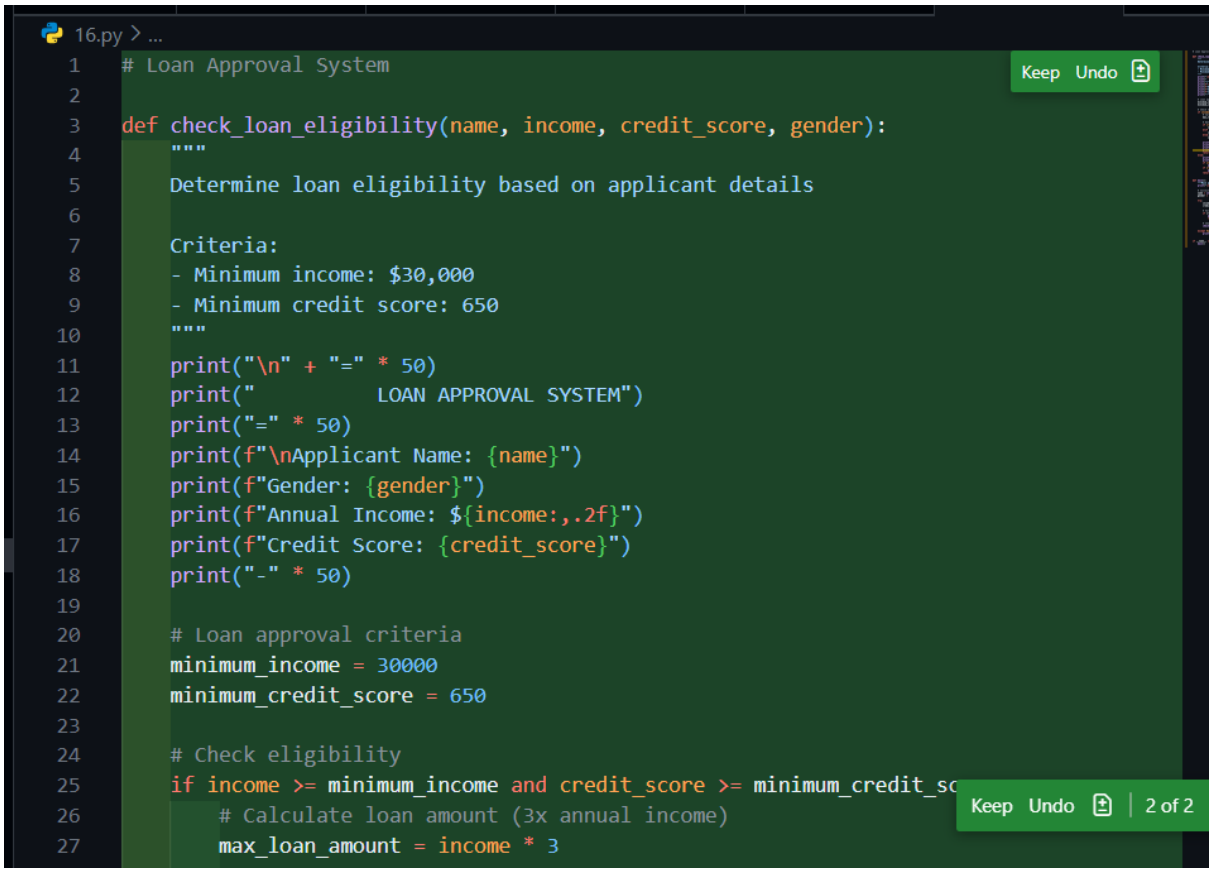
Objective

To evaluate AI-generated decision logic for bias and identify unfair treatment based on irrelevant personal attributes such as name or gender.

Prompt Used for AI Tool

Create a Python loan approval system based on applicant details like income, credit score, gender, and name.

Code generated



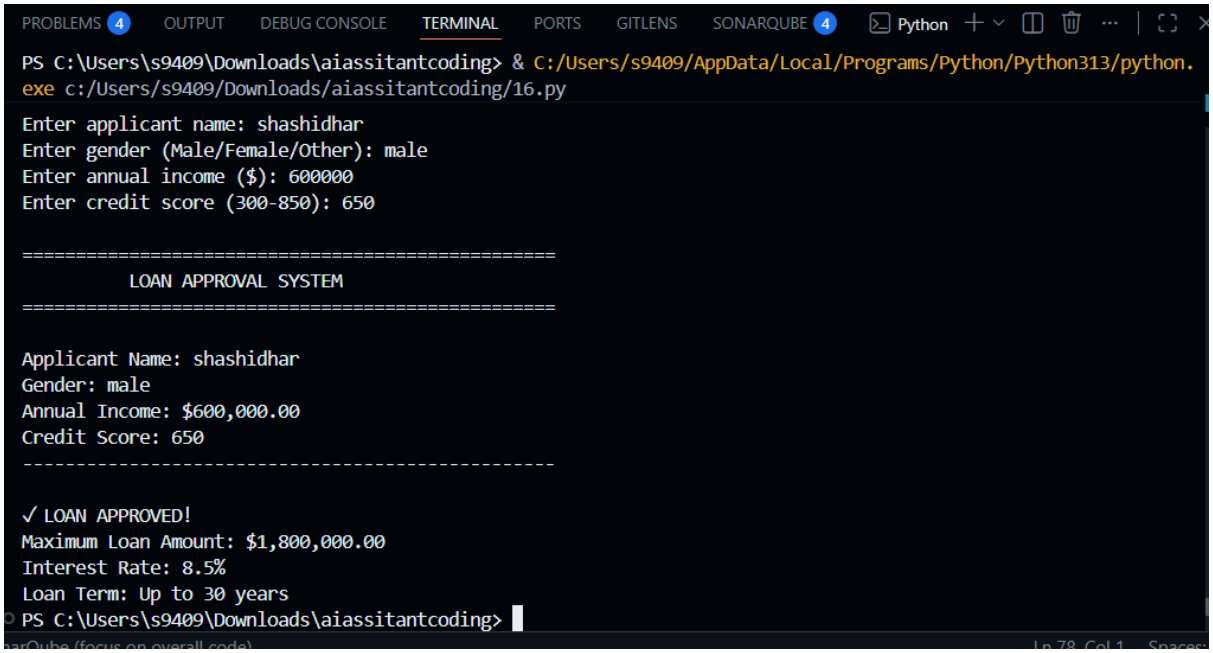
```python
def check_loan_eligibility(name, income, credit_score, gender):

        # Determine interest rate based on credit score
        if credit_score >= 750:
            interest_rate = 4.5
        elif credit_score >= 700:
            interest_rate = 6.0
        else:
            interest_rate = 8.5

        print("\n✓ LOAN APPROVED!")
        print(f"Maximum Loan Amount: ${max_loan_amount:,.2f}")
        print(f"Interest Rate: {interest_rate}%")
        print(f"Loan Term: Up to 30 years")    Add replacement fields or use a normal
        return True
    else:
        print("\nX LOAN DENIED")
        print("\nReason(s):")
        if income < minimum_income:
            print(f"  - Income below minimum requirement (${minimum_income:,})")
        if credit_score < minimum_credit_score:
            print(f"  - Credit score below minimum requirement ({minimum_credit_score}
        return False


def main():
    """Main program to collect applicant information"""
    print("\n*** Welcome to the Loan Approval System ***\n")
```

Keep Undo | 2 o

```python
def main():
    """Main program to collect applicant information"""
    print("\n*** Welcome to the Loan Approval System ***\n")

    # Collect applicant details
    name = input("Enter applicant name: ").strip()
    gender = input("Enter gender (Male/Female/Other): ").strip()

    try:
        income = float(input("Enter annual income ($): "))
        credit_score = int(input("Enter credit score (300-850): "))

        # Validate credit score range
        if credit_score < 300 or credit_score > 850:
            print("\nError: Credit score must be between 300 and 850")
            return

        # Check loan eligibility
        check_loan_eligibility(name, income, credit_score, gender)

    except ValueError:
        print("\nError: Please enter valid numeric values for income and credit score")


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

Keep Undo | 2 of 2

Output:



Bias Identification

1. Gender-Based Discrimination
    - Different approval thresholds for males and females.
    - Female applicants require higher income and credit score for approval.
2. Irrelevant Personal Attributes
    - Gender is used even though it has no financial relevance.
3. Unfair Decision Logic
    - Two applicants with identical financial profiles receive different outcomes.
4. Ethical Concern
    - Violates fairness, equality, and responsible AI principles.

Discussion on Fairness Issues

- Loan approval should depend only on financial capability.
- Using gender or name introduces systemic bias.
- Such AI systems can reinforce social discrimination if deployed in real-world banking systems.

Revised code

```python
def loan_approval(income, credit_score):
    if income > 30000 and credit_score > 650:
        return "Loan Approved"
    else:
        return "Loan Rejected"

print(loan_approval(35000, 680))
print(loan_approval(35000, 680))
```

Output:

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/16.py
Loan Approved
Loan Approved
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Justification

Bias Mitigation Strategies

1. Remove Irrelevant Attributes
    - Exclude gender, name, religion, or identity from decision logic.
2. Use Objective Criteria Only
    - Income, credit score, and repayment history.
3. Fairness Audits
    - Regularly test outputs for demographic parity.
4. Human Oversight
    - Developers must review AI logic before deployment.
5. Explainable Decisions
    - Ensure approval rules are transparent and justifiable.

Task 3: Transparency and Explainability in AI-Generated Code

(Recursive Binary Search)

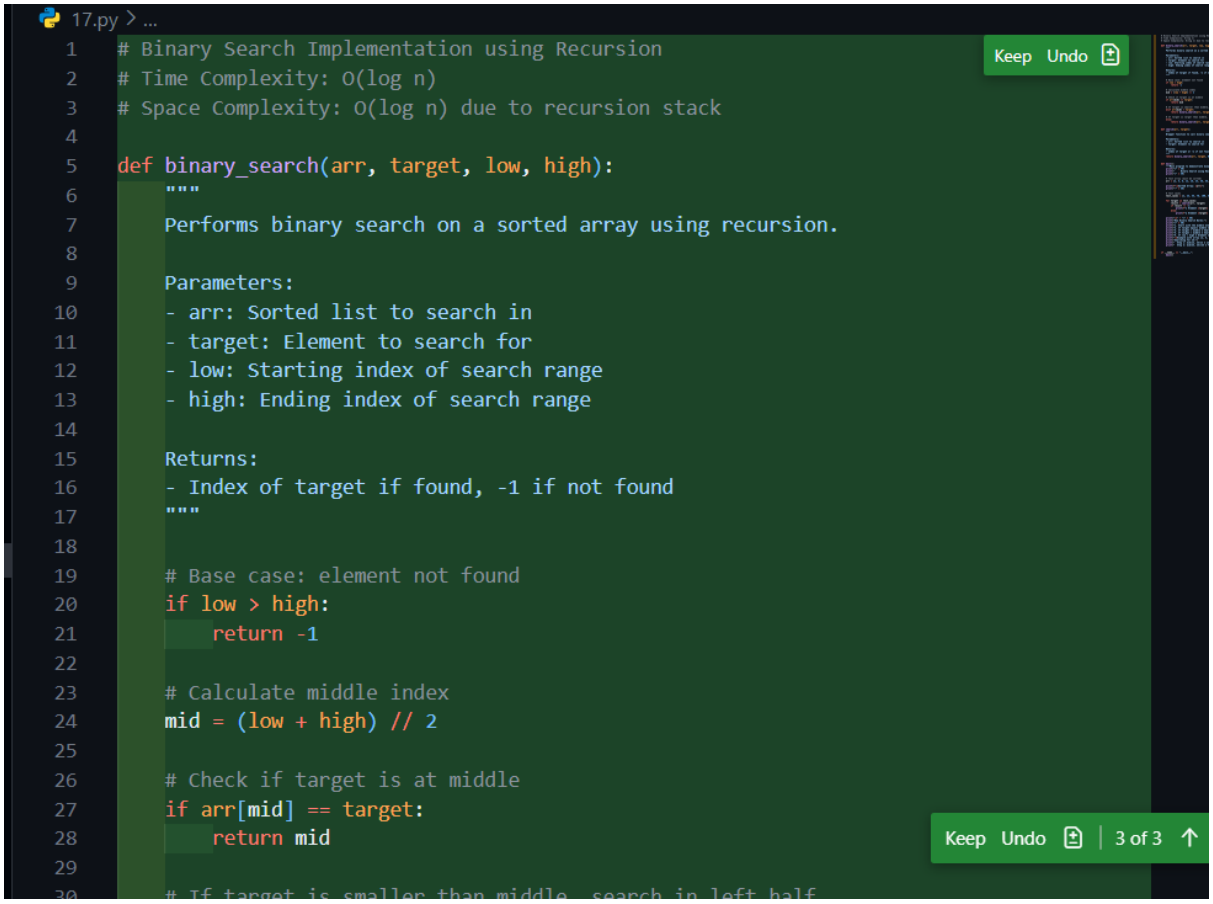
Objective

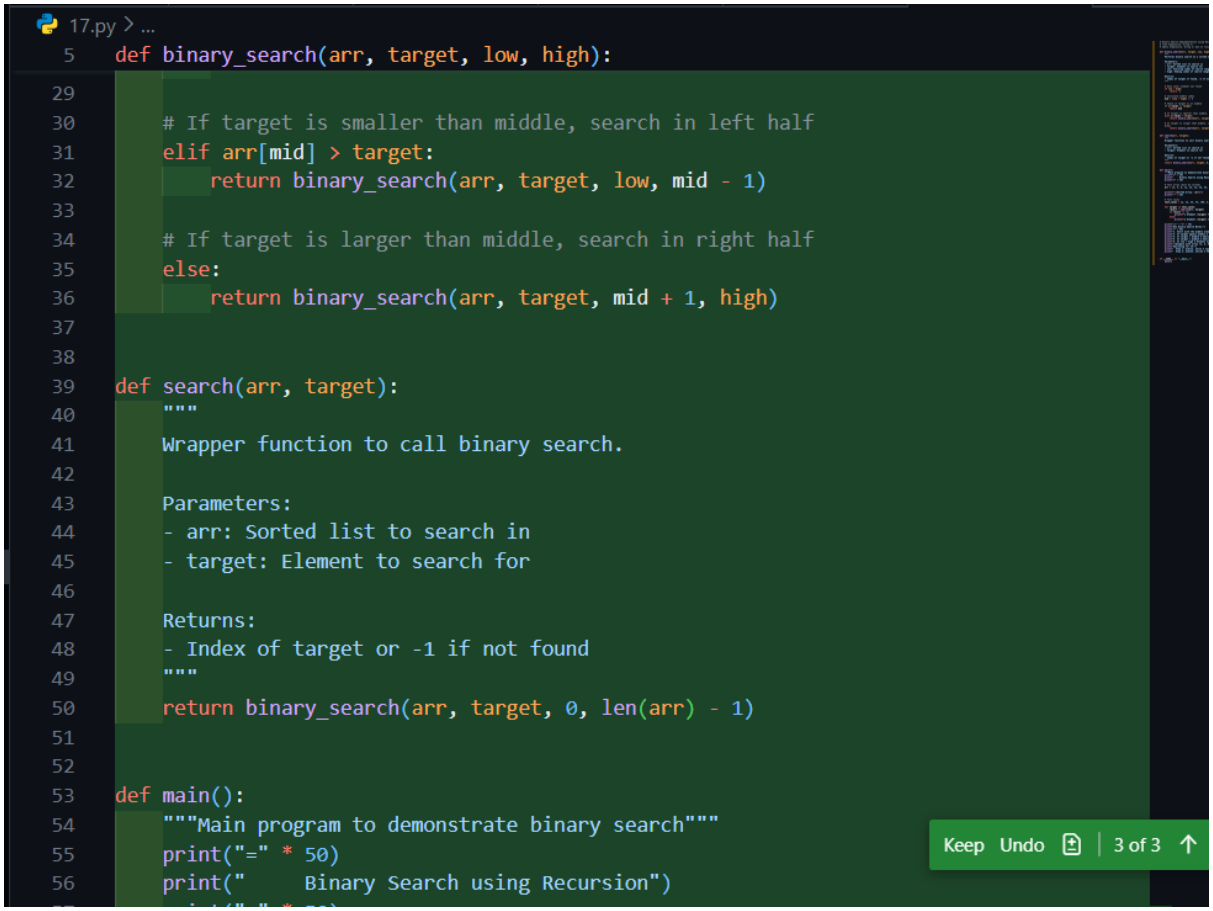
To evaluate whether AI-generated code is transparent, well-commented, and understandable for beginner-level learners, with a focus on explaining recursive logic clearly.

Prompt Used for AI Tool

Generate a Python program that implements Binary Search using recursion with clear comments and explanation.

Code generated




17.py > ...
def binary_search(arr, target, low, high):
# If target is smaller than middle, search in left half
elif arr[mid] > target:
return binary_search(arr, target, low, mid - 1)
# If target is larger than middle, search in right half
else:
return binary_search(arr, target, mid + 1, high)
def search(arr, target):
"""
Wrapper function to call binary search.
Parameters:
- arr: Sorted list to search in
- target: Element to search for
Returns:
- Index of target or -1 if not found
"""
return binary_search(arr, target, 0, len(arr) - 1)
def main():
"""Main program to demonstrate binary search"""
print("=" * 50)
print("   Binary Search using Recursion")
Keep Undo
3 of 3

```python
def main():
    print("=" * 50)
    print("     Binary Search using Recursion")
    print("=" * 50)

    # Test array (must be sorted)
    arr = [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]

    print(f"\nSorted Array: {arr}")
    print("-" * 50)

    # Test cases
    test_cases = [2, 23, 45, 78, 100, 5, 1]

    for target in test_cases:
        result = search(arr, target)
        if result != -1:
            print(f"✓ Element {target} found at index: {result}")
        else:
            print(f"✗ Element {target} not found in array")

    print("\n" + "=" * 50)
    print("How Binary Search Works:")
    print("=" * 50)
    print("1. Start with the middle element of sorted array")
    print("2. If target equals middle → Found! Return index")
    print("3. If target < middle → Search left half recursively")
    print("4. If target > middle → Search right half recursively")
    print("5. If low > high → Element not found, return -1")
    print("\nExample with array [2, 5, 8, 12, 16, 23, 38]")
```

Keep Undo | 3 of

```python
    print("\nExample with array [2, 5, 8, 12, 16, 23, 38]")
    print("Searching for 23:")
    print("  Step 1: mid=12, 23>12 → search right")
    print("  Step 2: mid=23, 23==23 → FOUND at index 5")


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

Output:

```
exe c:/Users/s9409/Downloads/aiassitantcoding/17.py
================================================
     Binary Search using Recursion
================================================

Sorted Array: [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]
------------------------------------------------
✓ Element 2 found at index: 0
✓ Element 23 found at index: 5
✓ Element 45 found at index: 7
✓ Element 78 found at index: 10
✗ Element 100 not found in array
✓ Element 5 found at index: 1
✗ Element 1 not found in array

================================================
How Binary Search Works:
================================================
1. Start with the middle element of sorted array
2. If target equals middle → Found! Return index
3. If target < middle → Search left half recursively
4. If target > middle → Search right half recursively
5. If low > high → Element not found, return -1

Example with array [2, 5, 8, 12, 16, 23, 38]
Searching for 23:
  Step 1: mid=12, 23>12 → search right
  Step 2: mid=23, 23==23 → FOUND at index 5
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

AI-Generated Explanation

- *Binary search works by repeatedly dividing the search interval in half.*
- *If the target value is less than the middle element, the search continues on the left side.*
- *If the target value is greater, the search continues on the right side.*
- *Recursion is used to call the function again with a smaller search range until the element is found or the range becomes invalid.*

Student Assessment

1. Base Case Explanation

Clearly defined:

if low > high:

return -1

- *Correctly stops recursion when element is not present.*

2. Recursive Case Explanation

Correctly handled:

- *Left recursion when target < middle element*

- *Right recursion when target > middle element*

3. Comment Accuracy

Comments correctly match the logic in each part of the function.

4. Beginner-Level Clarity

Code is:

- *Readable*
- *Modular*
- *Logically structured*
  Easy to trace recursion step-by-step.

Transparency Evaluation

Aspect	***Evaluation***
Code readability	*High*
Comment relevance	*Accurate*
Logic correctness	*Correct*
Beginner understanding	*Suitable*

Ethical Analysis

- *The code is **transparent and explainable**.*
- *Each decision point is understandable and verifiable.*
- *Encourages **trustworthy AI-assisted coding**.*
- *Demonstrates that AI output must still be **reviewed by humans**.*

Justification

The AI-generated recursive binary search code demonstrates good transparency and explainability. Clear comments, logical structure, and accurate explanations make the algorithm understandable even for beginner students, fulfilling responsible AI coding principles.

Task 4: Ethical Evaluation of AI-Based Scoring Systems

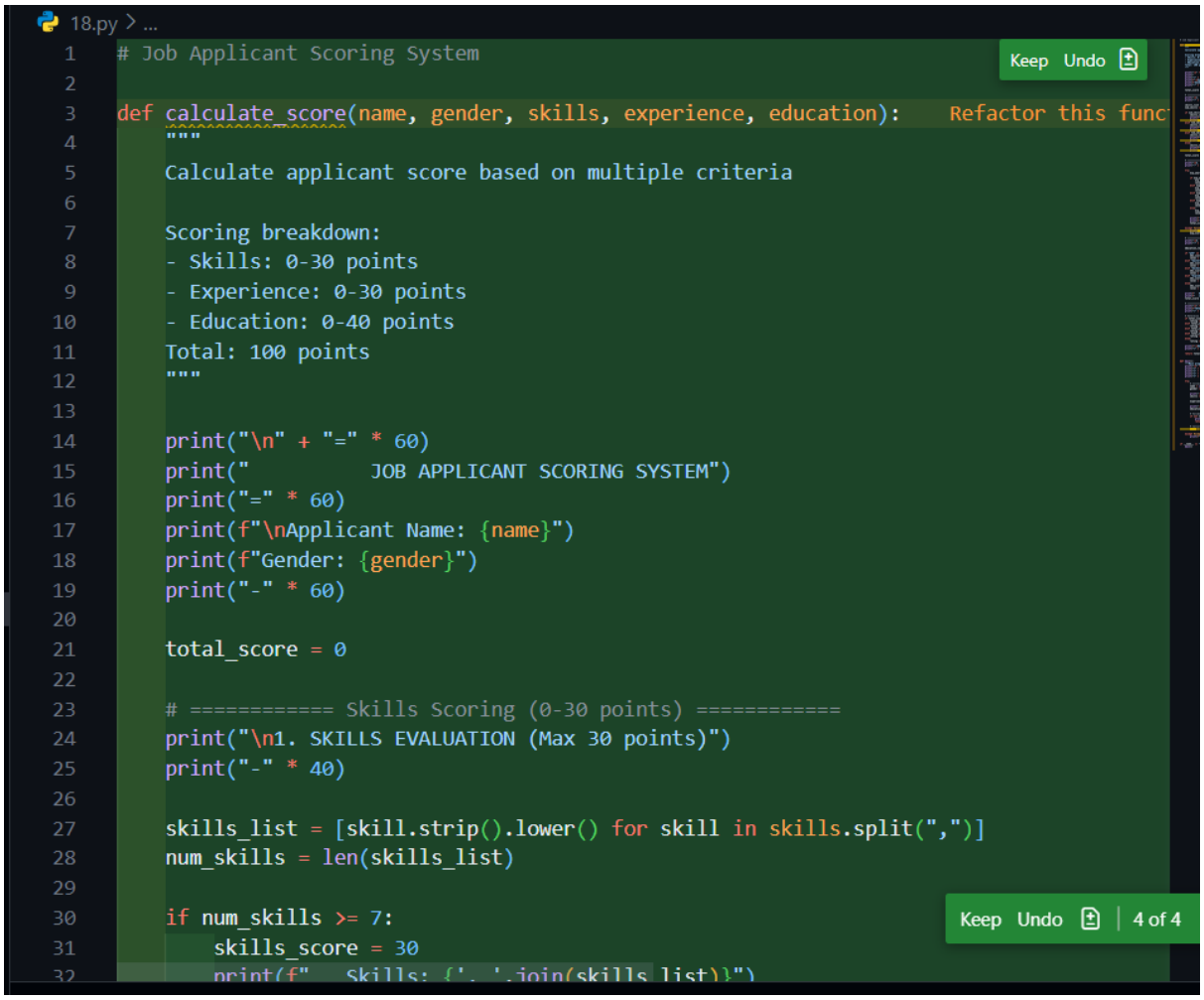
Objective

To examine AI-generated scoring logic for job applicants and evaluate whether the system is fair, objective, and free from bias related to gender, name, or other irrelevant attributes.

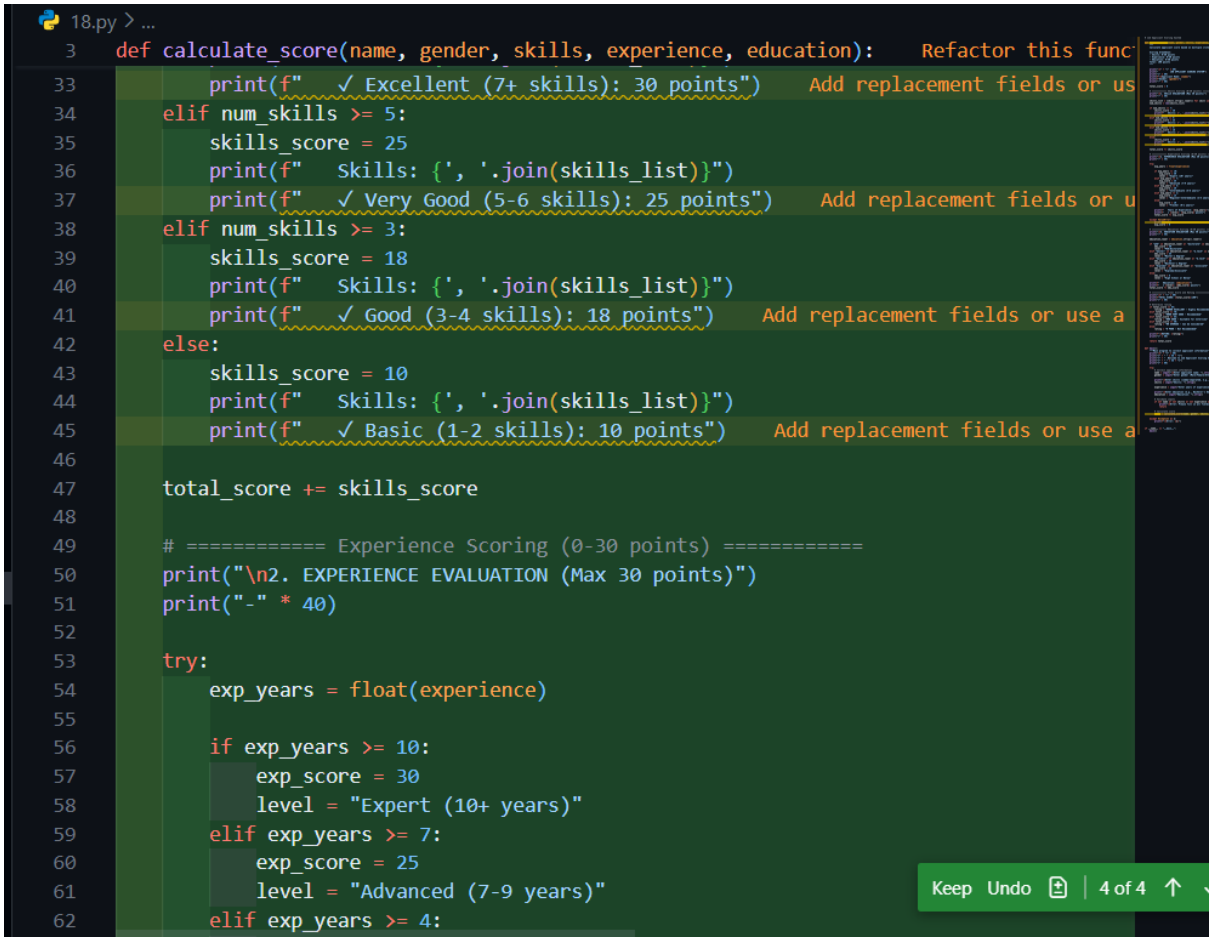
Prompt Used for AI Tool

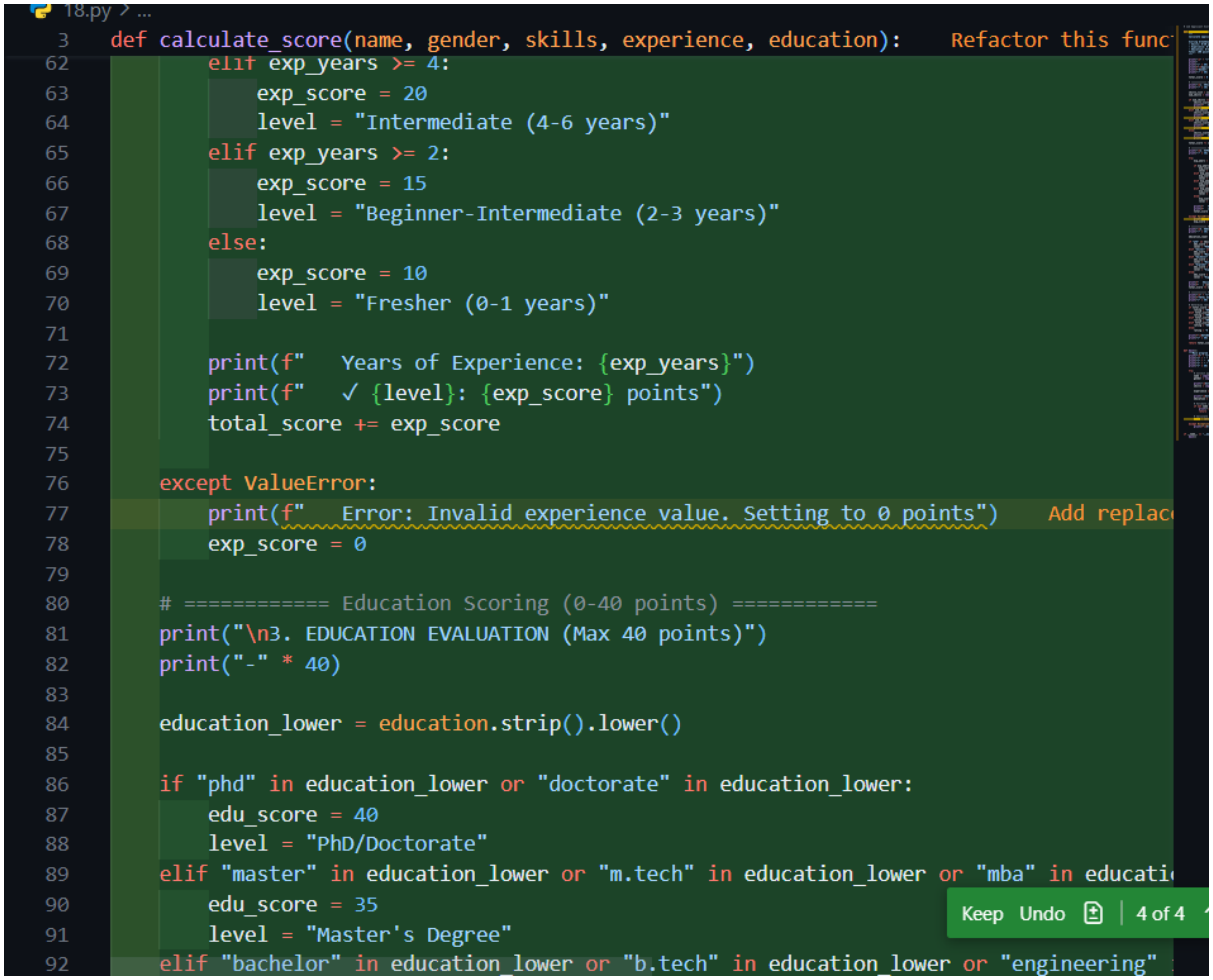
Create a Python job applicant scoring system based on skills, experience, education, and gender.

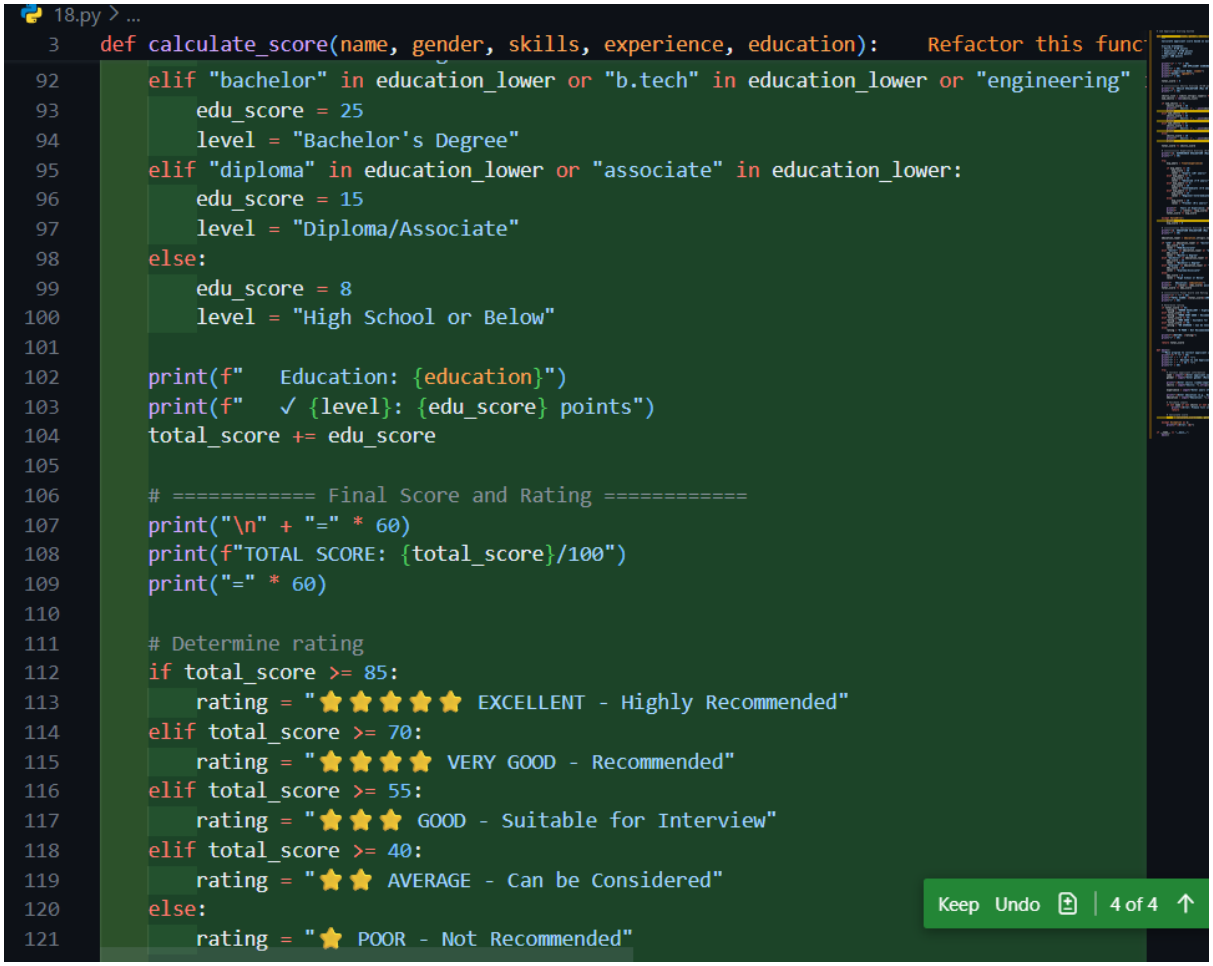
Generated code




```python
# Job Applicant Scoring System

def calculate_score(name, gender, skills, experience, education):
    """
    Calculate applicant score based on multiple criteria

    Scoring breakdown:
    - Skills: 0-30 points
    - Experience: 0-30 points
    - Education: 0-40 points
    Total: 100 points
    """

    print("\n" + "=" * 60)
    print("          JOB APPLICANT SCORING SYSTEM")
    print("=" * 60)
    print(f"\nApplicant Name: {name}")
    print(f"Gender: {gender}")
    print("-" * 60)

    total_score = 0

    # ============ Skills Scoring (0-30 points) ============
    print("\n1. SKILLS EVALUATION (Max 30 points)")
    print("-" * 40)

    skills_list = [skill.strip().lower() for skill in skills.split(",")]
    num_skills = len(skills_list)

    if num_skills >= 7:
        skills_score = 30
        print(f"   Skills: {', '.join(skills_list)}")
```


18.py > ...
3 def calculate_score(name, gender, skills, experience, education): Refactor this func
33 print(f" ✓ Excellent (7+ skills): 30 points") Add replacement fields or us
34 elif num_skills >= 5:
35 skills_score = 25
36 print(f" Skills: {', '.join(skills_list)}")
37 print(f" ✓ Very Good (5-6 skills): 25 points") Add replacement fields or u
38 elif num_skills >= 3:
39 skills_score = 18
40 print(f" Skills: {', '.join(skills_list)}")
41 print(f" ✓ Good (3-4 skills): 18 points") Add replacement fields or use a
42 else:
43 skills_score = 10
44 print(f" Skills: {', '.join(skills_list)}")
45 print(f" ✓ Basic (1-2 skills): 10 points") Add replacement fields or use a
46
47 total_score += skills_score
48
49 # ============ Experience Scoring (0-30 points) ============
50 print("\n2. EXPERIENCE EVALUATION (Max 30 points)")
51 print("-" * 40)
52
53 try:
54 exp_years = float(experience)
55
56 if exp_years >= 10:
57 exp_score = 30
58 level = "Expert (10+ years)"
59 elif exp_years >= 7:
60 exp_score = 25
61 level = "Advanced (7-9 years)"
62 elif exp_years >= 4:
Keep Undo 4 of 4

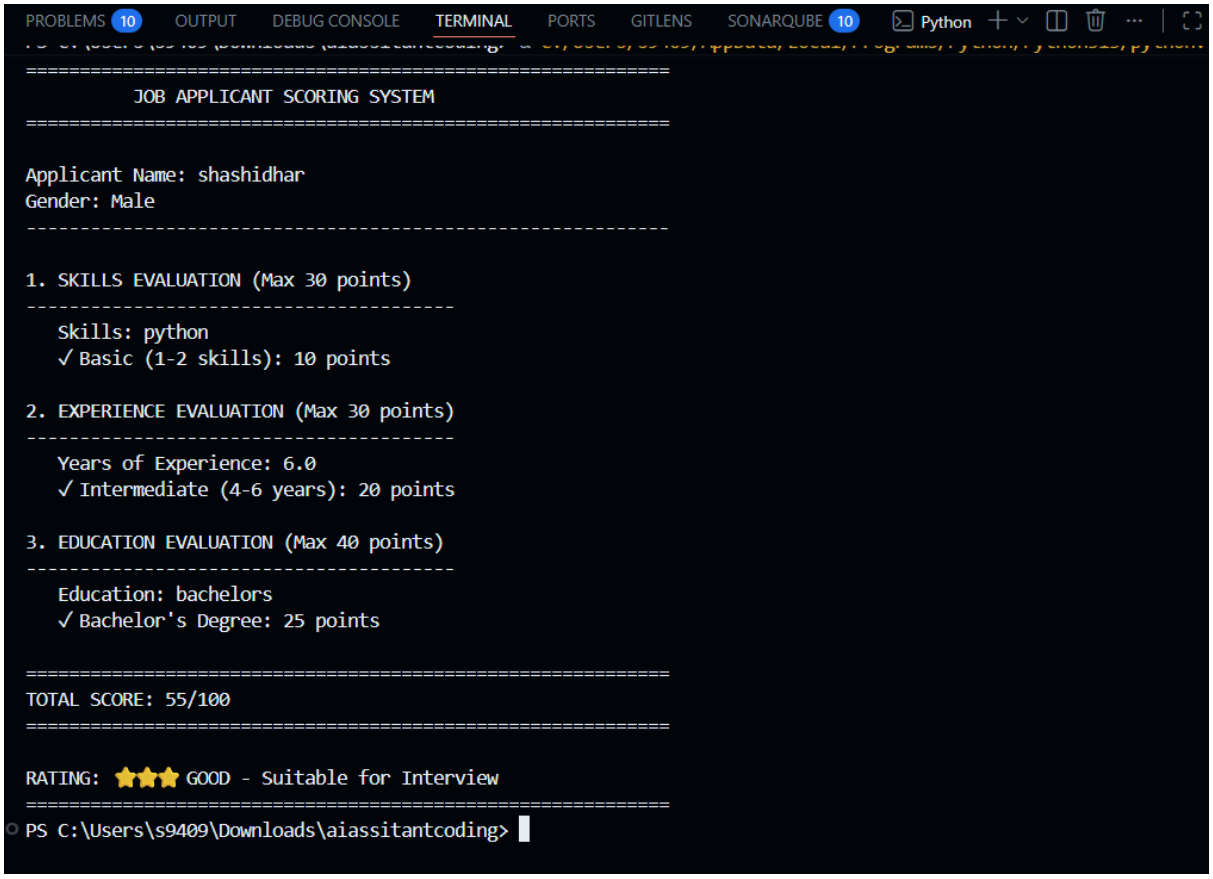

18.py > ...
def calculate_score(name, gender, skills, experience, education):
Refactor this func
elif exp_years >= 4:
exp_score = 20
level = "Intermediate (4-6 years)"
elif exp_years >= 2:
exp_score = 15
level = "Beginner-Intermediate (2-3 years)"
else:
exp_score = 10
level = "Fresher (0-1 years)"
print(f" Years of Experience: {exp_years}")
print(f" ✓ {level}: {exp_score} points")
total_score += exp_score
except ValueError:
print(f" Error: Invalid experience value. Setting to 0 points")
Add replac
exp_score = 0
# ============ Education Scoring (0-40 points) ============
print("\n3. EDUCATION EVALUATION (Max 40 points)")
print("-" * 40)
education_lower = education.strip().lower()
if "phd" in education_lower or "doctorate" in education_lower:
edu_score = 40
level = "PhD/Doctorate"
elif "master" in education_lower or "m.tech" in education_lower or "mba" in educati
edu_score = 35
level = "Master's Degree"
elif "bachelor" in education_lower or "b.tech" in education_lower or "engineering"
Keep Undo 4 of 4


18.py > ...
def calculate_score(name, gender, skills, experience, education): Refactor this func
elif "bachelor" in education_lower or "b.tech" in education_lower or "engineering"
edu_score = 25
level = "Bachelor's Degree"
elif "diploma" in education_lower or "associate" in education_lower:
edu_score = 15
level = "Diploma/Associate"
else:
edu_score = 8
level = "High School or Below"
print(f" Education: {education}")
print(f" ✓ {level}: {edu_score} points")
total_score += edu_score
# ============ Final Score and Rating ============
print("\n" + "=" * 60)
print(f"TOTAL SCORE: {total_score}/100")
print("=" * 60)
# Determine rating
if total_score >= 85:
rating = "⭐⭐⭐⭐⭐ EXCELLENT - Highly Recommended"
elif total_score >= 70:
rating = "⭐⭐⭐⭐ VERY GOOD - Recommended"
elif total_score >= 55:
rating = "⭐⭐⭐ GOOD - Suitable for Interview"
elif total_score >= 40:
rating = "⭐⭐ AVERAGE - Can be Considered"
else:
rating = "⭐ POOR - Not Recommended"
Keep Undo 4 of 4

```
def calculate_score(name, gender, skills, experience, education):    Refactor this func
    print(f"\nRATING: {rating}")
    print("=" * 60)

    return total_score


def main():
    """Main program to collect applicant information"""
    print("\n" + "*" * 60)
    print("*" + " " * 58 + "*")
    print("*" + "  Welcome to Job Applicant Scoring System".center(58) + "*")
    print("*" + " " * 58 + "*")
    print("*" * 60)

    try:
        # Collect applicant information
        name = input("\nEnter applicant name: ").strip()
        gender = input("Enter gender (Male/Female/Other): ").strip()

        print("\nEnter skills (comma-separated, e.g., Python, Java, SQL):")
        skills = input("Skills: ").strip()

        experience = input("Enter years of experience: ").strip()

        print("\nEnter education (e.g., Bachelor's Degree, MBA, B.Tech):")
        education = input("Education: ").strip()

        # Validate inputs
        if not name or not skills or not experience or not education:
            print("\nError: Please fill in all fields")
            return

        # Calculate score
        score = calculate_score(name, gender, skills, experience, edu
```

Keep Undo | 4 of 4

Output:



Identification of Ethical Issues / Bias

1. *Gender-Based Bias*
    - *Male candidates receive additional score points solely due to gender.*
2. *Unrelated Attribute Influence*
    - *Gender has no relevance to job performance or qualification.*
3. *Unfair Ranking*
    - *Two identical candidates receive different scores.*
4. *Ethical Risk*
    - *Such logic can lead to discriminatory hiring practices.*

Ethical Analysis

- *Job scoring systems directly impact employment opportunities.*
- *Any bias can lead to systemic discrimination.*
- *AI-generated code must adhere to:*
    - *Fairness*
    - *Objectivity*
    - *Equal opportunity*

Revised code

Output:

Justification for Improvements

1. ***Removed Gender Dependency***
   - *Gender is excluded from scoring logic.*
2. ***Objective Criteria Only***
   - *Skills, experience, and education are job-relevant attributes.*
3. ***Ethical Compliance***
   - *Aligns with responsible AI and equal employment standards.*
4. ***Transparency***
   - *Scoring rules are clear and explainable.*

Task 5: Inclusiveness and Ethical Variable Design

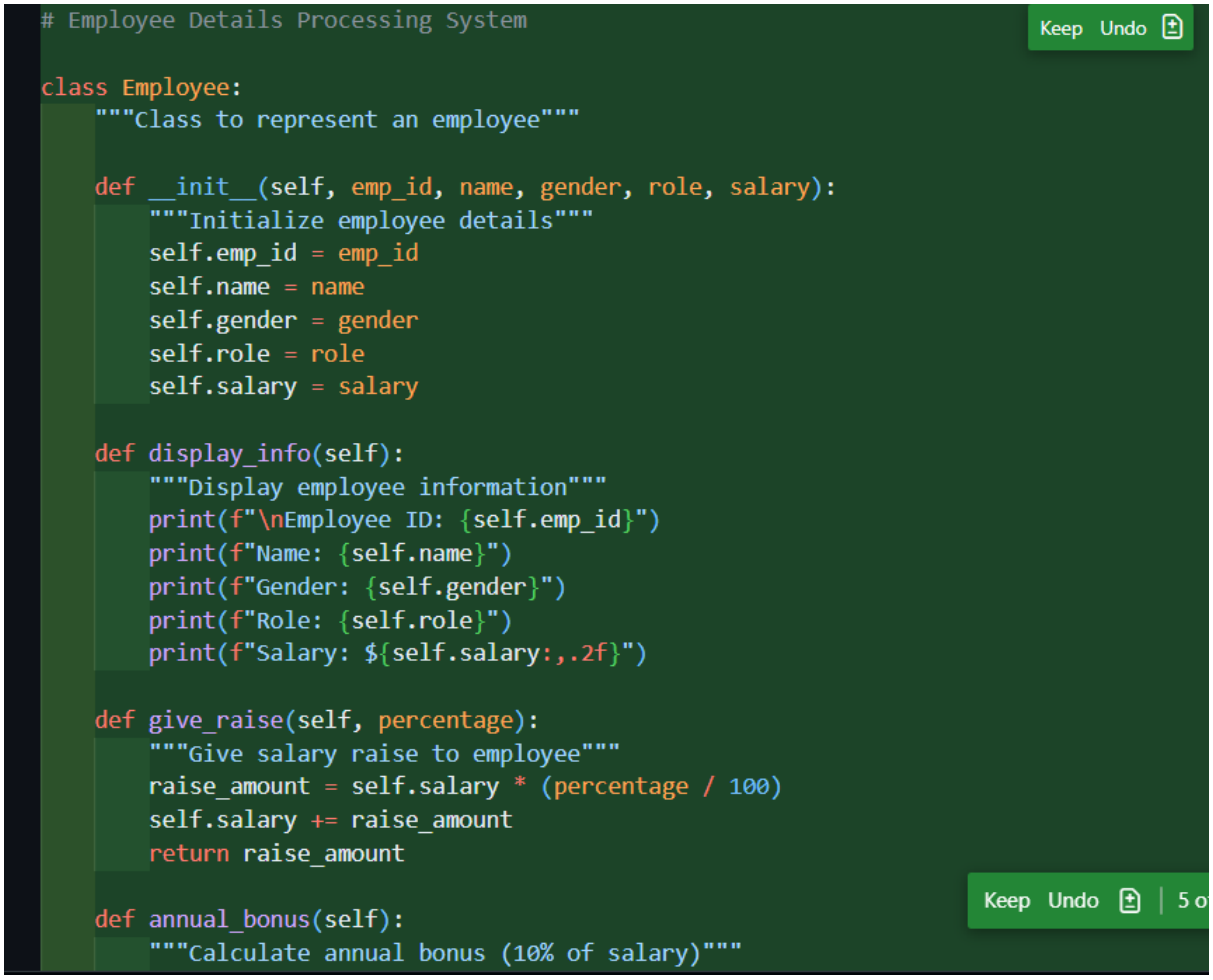
Objective

To analyze AI-generated code for non-inclusive practices and redesign it using gender-neutral variables and inclusive logic.

Prompt Used for AI Tool

Generate a Python program to process employee details including name, gender, role, and salary.

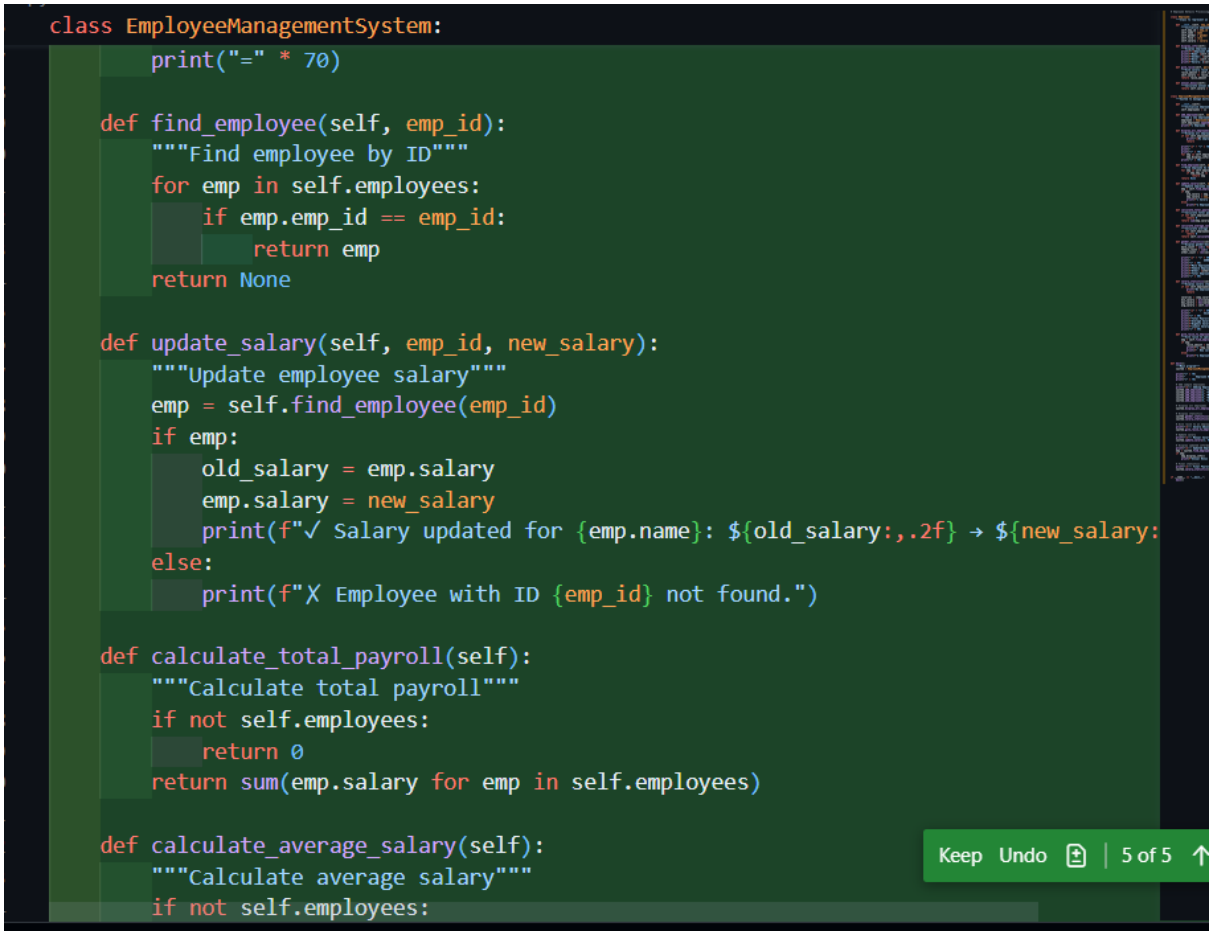
Generated code



```python
# Employee Details Processing System

class Employee:
    """Class to represent an employee"""

    def __init__(self, emp_id, name, gender, role, salary):
        """Initialize employee details"""
        self.emp_id = emp_id
        self.name = name
        self.gender = gender
        self.role = role
        self.salary = salary

    def display_info(self):
        """Display employee information"""
        print(f"\nEmployee ID: {self.emp_id}")
        print(f"Name: {self.name}")
        print(f"Gender: {self.gender}")
        print(f"Role: {self.role}")
        print(f"Salary: ${self.salary:,.2f}")

    def give_raise(self, percentage):
        """Give salary raise to employee"""
        raise_amount = self.salary * (percentage / 100)
        self.salary += raise_amount
        return raise_amount

    def annual_bonus(self):
        """Calculate annual bonus (10% of salary)"""
```

```
        return raise_amount

    def annual_bonus(self):
        """Calculate annual bonus (10% of salary)"""
        return self.salary * 0.10


class EmployeeManagementSystem:
    """System to manage multiple employees"""

    def __init__(self):
        """Initialize employee list"""
        self.employees = []

    def add_employee(self, emp_id, name, gender, role, salary):
        """Add a new employee"""
        employee = Employee(emp_id, name, gender, role, salary)
        self.employees.append(employee)
        print(f"✓ Employee '{name}' added successfully!")

    def display_all_employees(self):
        """Display all employees"""
        if not self.employees:
            print("\nNo employees in the system.")
            return

        print("\n" + "=" * 70)
```

Keep Undo | 5 of 5



```
class EmployeeManagementSystem:
        print("=" * 70)

    def find_employee(self, emp_id):
        """Find employee by ID"""
        for emp in self.employees:
            if emp.emp_id == emp_id:
                return emp
        return None

    def update_salary(self, emp_id, new_salary):
        """Update employee salary"""
        emp = self.find_employee(emp_id)
        if emp:
            old_salary = emp.salary
            emp.salary = new_salary
            print(f"✓ Salary updated for {emp.name}: ${old_salary:,.2f} → ${new_salary:
        else:
            print(f"✗ Employee with ID {emp_id} not found.")

    def calculate_total_payroll(self):
        """Calculate total payroll"""
        if not self.employees:
            return 0
        return sum(emp.salary for emp in self.employees)

    def calculate_average_salary(self):
        """Calculate average salary"""
        if not self.employees:
```

```python
    def calculate_average_salary(self):
        """Calculate average salary"""
        if not self.employees:
            return 0
        return self.calculate_total_payroll() / len(self.employees)

    def gender_statistics(self):
        """Display gender-based statistics"""
        male_count = sum(1 for emp in self.employees if emp.gender.lower() == "male")
        female_count = sum(1 for emp in self.employees if emp.gender.lower() == "female"
        other_count = len(self.employees) - male_count - female_count

        print("\n" + "=" * 50)
        print("          GENDER STATISTICS")
        print("=" * 50)
        print(f"Male Employees: {male_count}")
        print(f"Female Employees: {female_count}")
        print(f"Other: {other_count}")
        print(f"Total Employees: {len(self.employees)}")
        print("=" * 50)

    def salary_statistics(self):
        """Display salary statistics"""
        if not self.employees:
            print("No employees to calculate statistics.")
            return

        salaries = [emp.salary for emp in self.employees]
        min_salary = min(salaries)
        max_salary = max(salaries)
        avg_salary = self.calculate_average_salary()

        print("\n" + "=" * 50)
        print("          SALARY STATISTICS")
        print("=" * 50)
```

```python
    def give_raise_to_employee(self, emp_id, percentage):
        """Give raise to specific employee"""
        emp = self.find_employee(emp_id)
        if emp:
            raise_amount = emp.give_raise(percentage)
            print(f"✓ {emp.name} received a {percentage}% raise: ${raise_amount:,.2f}")
            print(f"  New salary: ${emp.salary:,.2f}")
        else:
            print(f"X Employee with ID {emp_id} not found.")


def main():
    """Main program"""
    system = EmployeeManagementSystem()

    print("=" * 70)
    print("        Employee Details Processing System")
    print("=" * 70)

    # Add sample employees
    print("\n--- Adding Employees ---")
    system.add_employee(1, "John Smith", "Male", "Software Engineer", 75000)
    system.add_employee(2, "Sarah Johnson", "Female", "Product Manager", 85000)
    system.add_employee(3, "Michael Brown", "Male", "Data Analyst", 65000)
    system.add_employee(4, "Emily Davis", "Female", "UX Designer", 70000)
    system.add_employee(5, "James Wilson", "Male", "DevOps Engineer", 80000)

    # Display all employees
    system.display_all_employees()

    # Display statistics
    system.gender_statistics()
    system.salary_statistics()

```

```python
    system.add_employee(4, "Emily Davis", "Female", "UX Designer", 70000)
    system.add_employee(5, "James Wilson", "Male", "DevOps Engineer", 80000)

    # Display all employees
    system.display_all_employees()

    # Display statistics
    system.gender_statistics()
    system.salary_statistics()

    # Give raise to an employee
    print("\n--- Salary Adjustment ---")
    system.give_raise_to_employee(1, 10)

    # Update salary
    print("\n--- Manual Salary Update ---")
    system.update_salary(2, 90000)

    # Display updated information
    print("\n--- Updated Employee Information ---")
    emp = system.find_employee(1)
    if emp:
        emp.display_info()
        print(f"Annual Bonus (10%): ${emp.annual_bonus():,.2f}")

    # Final statistics
    print("\n--- Final Payroll Summary ---")
    system.salary_statistics()


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

Output:

```
======================================================================
        Employee Details Processing System
======================================================================

--- Adding Employees ---
✓ Employee 'John Smith' added successfully!
✓ Employee 'Sarah Johnson' added successfully!
✓ Employee 'Michael Brown' added successfully!
✓ Employee 'Emily Davis' added successfully!
✓ Employee 'James Wilson' added successfully!

======================================================================
                    ALL EMPLOYEES
======================================================================

Employee ID: 1
Name: John Smith
Gender: Male
Role: Software Engineer
Salary: $75,000.00

Employee ID: 2
Name: Sarah Johnson
Gender: Female
Role: Product Manager
Salary: $85,000.00

Employee ID: 3
Name: Michael Brown
Gender: Male
Role: Data Analyst
Salary: $65,000.00

Employee ID: 4
Name: Emily Davis
Gender: Female
Role: UX Designer
Salary: $70,000.00

Employee ID: 5
Name: James Wilson
```

```
Employee ID: 5
Name: James Wilson
Gender: Male
Role: DevOps Engineer
Salary: $80,000.00
======================================================================

================================================
         GENDER STATISTICS
================================================
Male Employees: 3
Female Employees: 2
Other: 0
Total Employees: 5
================================================

================================================
         SALARY STATISTICS
================================================
Total Payroll: $375,000.00
Average Salary: $75,000.00
Highest Salary: $85,000.00
Lowest Salary: $65,000.00
================================================

--- Salary Adjustment ---
✓ John Smith received a 10% raise: $7,500.00
  New salary: $82,500.00

--- Manual Salary Update ---
✓ Salary updated for Sarah Johnson: $85,000.00 → $90,000.00

--- Updated Employee Information ---

Employee ID: 1
Name: John Smith
Gender: Male
Role: Software Engineer
Salary: $82,500.00
Annual Bonus (10%): $8,250.00
```



```
Employee ID: 1
Name: John Smith
Gender: Male
Role: Software Engineer
Salary: $82,500.00
Annual Bonus (10%): $8,250.00

--- Final Payroll Summary ---

================================================
         SALARY STATISTICS
================================================
Total Payroll: $387,500.00
Average Salary: $77,500.00
Highest Salary: $90,000.00
Lowest Salary: $65,000.00
================================================
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Revised code

```python
def process_employee(name, salary, performance_rating):
    bonus_rate = 0

    if performance_rating >= 4:
        bonus_rate = 0.10
    elif performance_rating >= 3:
        bonus_rate = 0.08

    bonus = salary * bonus_rate
    print(name, "gets a bonus of", bonus)


process_employee("Ramesh", 50000, 4)
process_employee("Anita", 50000, 4)
```

Output:

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Use
exe c:/Users/s9409/Downloads/aiassitantcoding/19.py
Ramesh gets a bonus of 5000.0
Anita gets a bonus of 5000.0
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Explanation of Improvements

What Was Non-Inclusive

- *Gender-specific variables influenced salary benefits.*
- *Binary gender assumptions limited fairness.*

How Inclusiveness Was Improved

- *Removed gender from logic entirely.*
- *Introduced performance-based evaluation.*
- *Used neutral variables aligned with job relevance.*

Justification

This task shows that AI-generated code can unintentionally embed non-inclusive assumptions. Ethical and inclusive coding requires gender-neutral design, fairness-driven logic, and careful human review of AI-assisted outputs.