AI Assisted Coding

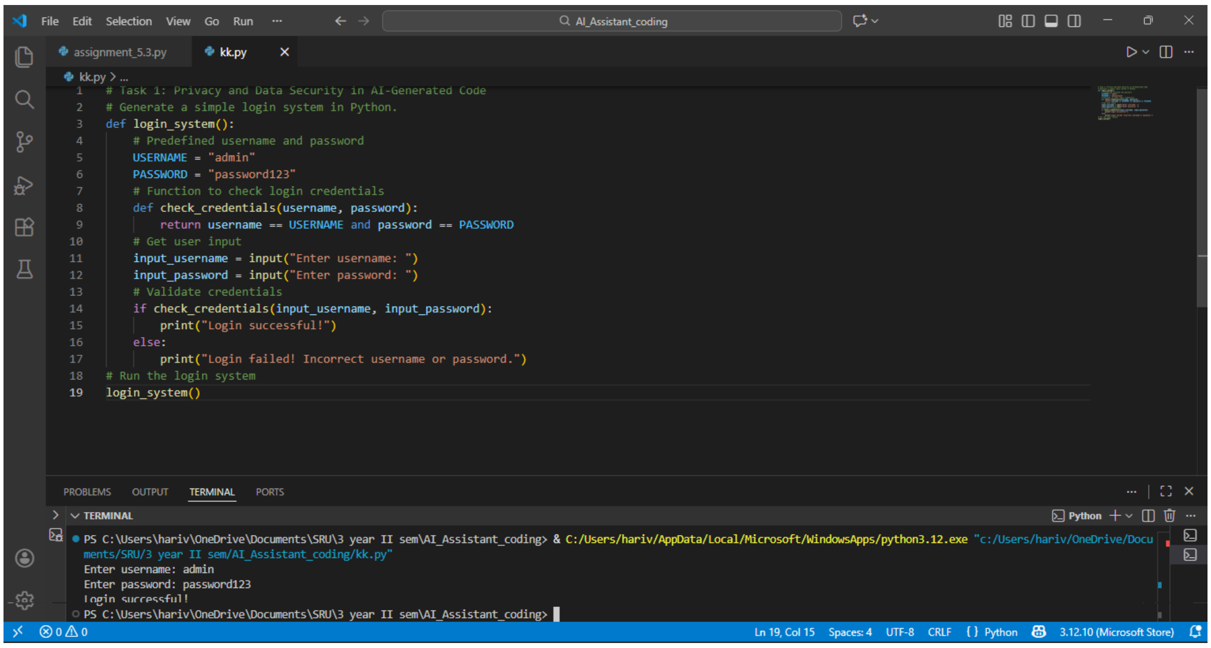
Assignment 5.3

Name: V.Harivamsh

Hall ticket no: 2303A51266

Batch no: 19

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


File Edit Selection View Go Run ···
AI_Assistant_coding
assignment_5.3.py
kk.py
kk.py > ...
PROBLEMS OUTPUT TERMINAL PORTS
TERMINAL
Python
Ln 19, Col 15 Spaces: 4 UTF-8 CRLF {} Python 3.12.10 (Microsoft Store)

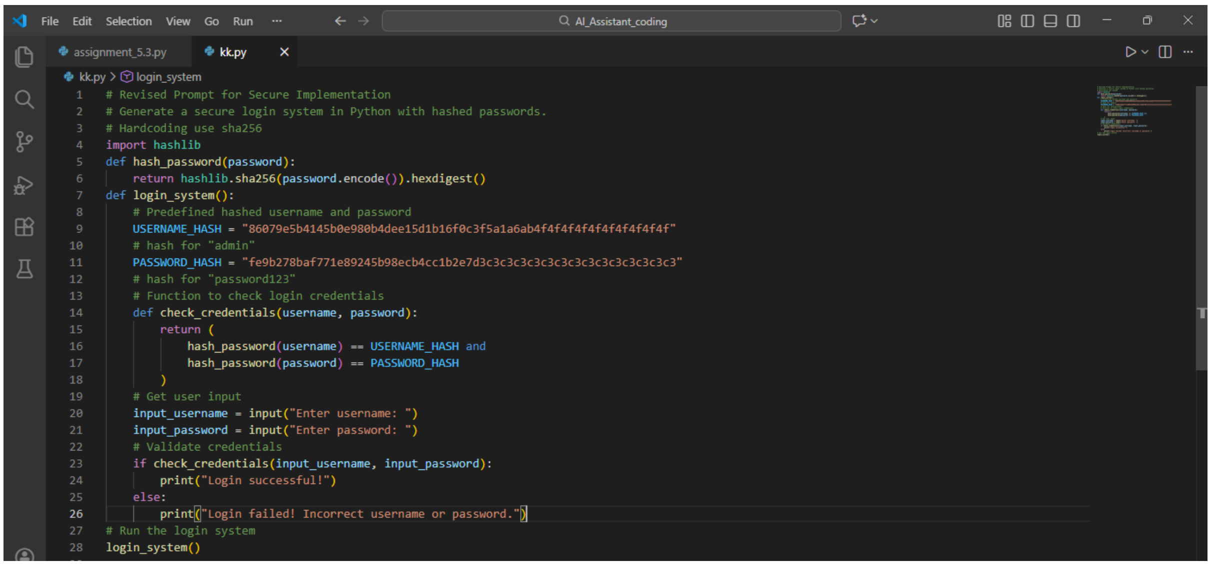
```python
# Task 1: Privacy and Data Security in AI-Generated Code
# Generate a simple login system in Python.
def login_system():
    # Predefined username and password
    USERNAME = "admin"
    PASSWORD = "password123"
    # Function to check login credentials
    def check_credentials(username, password):
        return username == USERNAME and password == PASSWORD
    # Get user input
    input_username = input("Enter username: ")
    input_password = input("Enter password: ")
    # Validate credentials
    if check_credentials(input_username, input_password):
        print("Login successful!")
    else:
        print("Login failed! Incorrect username or password.")
# Run the login system
login_system()
```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/kk.py"
Enter username: admin
Enter password: password123
Login successful!
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

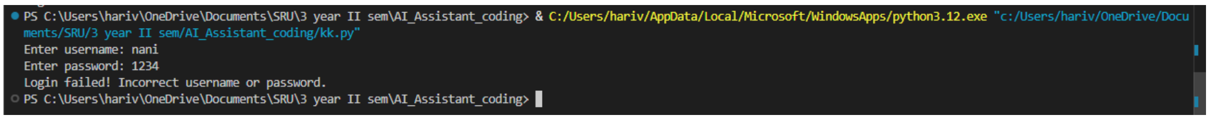
➢ Identification of Security Risks

- The username and password are directly written in the program, so anyone who accesses the code can easily see them.
- The password is handled in plain text, which means it is not protected and can be easily stolen or misused.
- There is no encryption or hashing used to secure sensitive login information.
- The login system does not properly validate user input or follow secure authentication practices, making it vulnerable to attacks.

Improved Code:



```python
# Revised Prompt for Secure Implementation
# Generate a secure login system in Python with hashed passwords.
# Hardcoding use sha256
import hashlib
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
def login_system():
    # Predefined hashed username and password
    USERNAME_HASH = "86079e5b4145b0e980b4dee15d1b16f0c3f5a1a6ab4f4f4f4f4f4f4f4f4f"
    # hash for "admin"
    PASSWORD_HASH = "fe9b278baf771e89245b98ecb4cc1b2e7d3c3c3c3c3c3c3c3c3c3c3c3c3"
    # hash for "password123"
    # Function to check login credentials
    def check_credentials(username, password):
        return (
            hash_password(username) == USERNAME_HASH and
            hash_password(password) == PASSWORD_HASH
        )
    # Get user input
    input_username = input("Enter username: ")
    input_password = input("Enter password: ")
    # Validate credentials
    if check_credentials(input_username, input_password):
        print("Login successful!")
    else:
        print("Login failed! Incorrect username or password.")
# Run the login system
login_system()
```

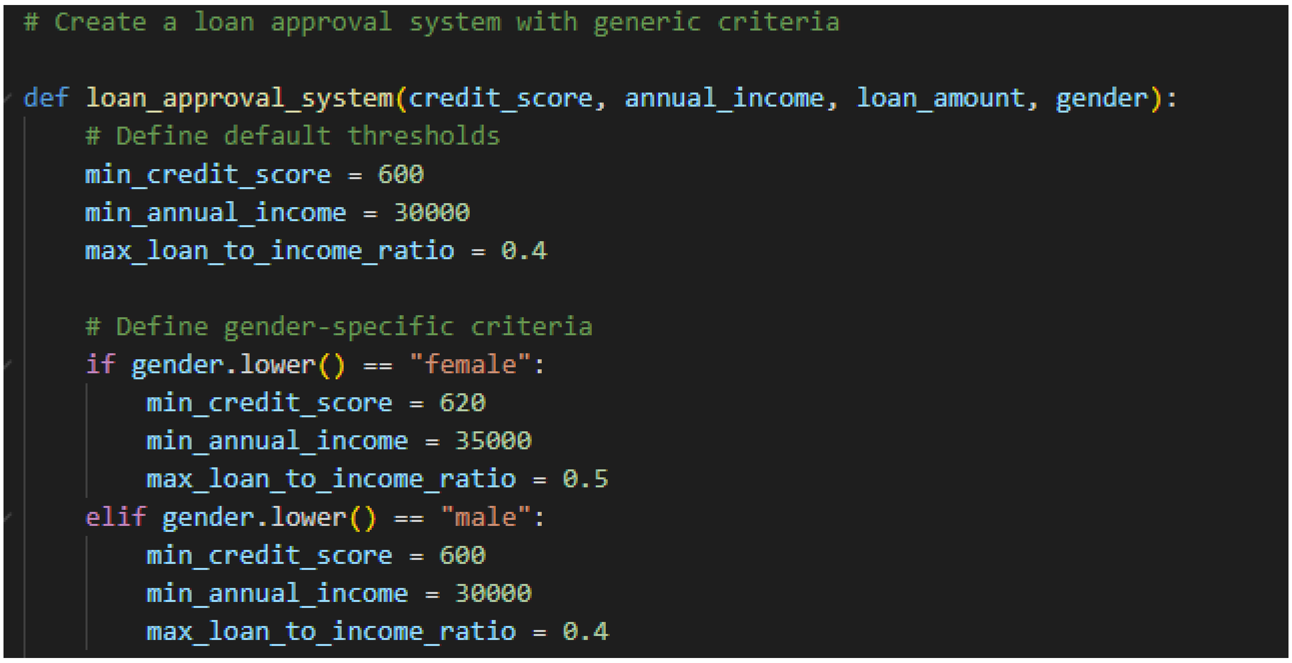


```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Docu
ments/SRU/3 year II sem/AI_Assistant_coding/kk.py"
Enter username: nani
Enter password: 1234
Login failed! Incorrect username or password.
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

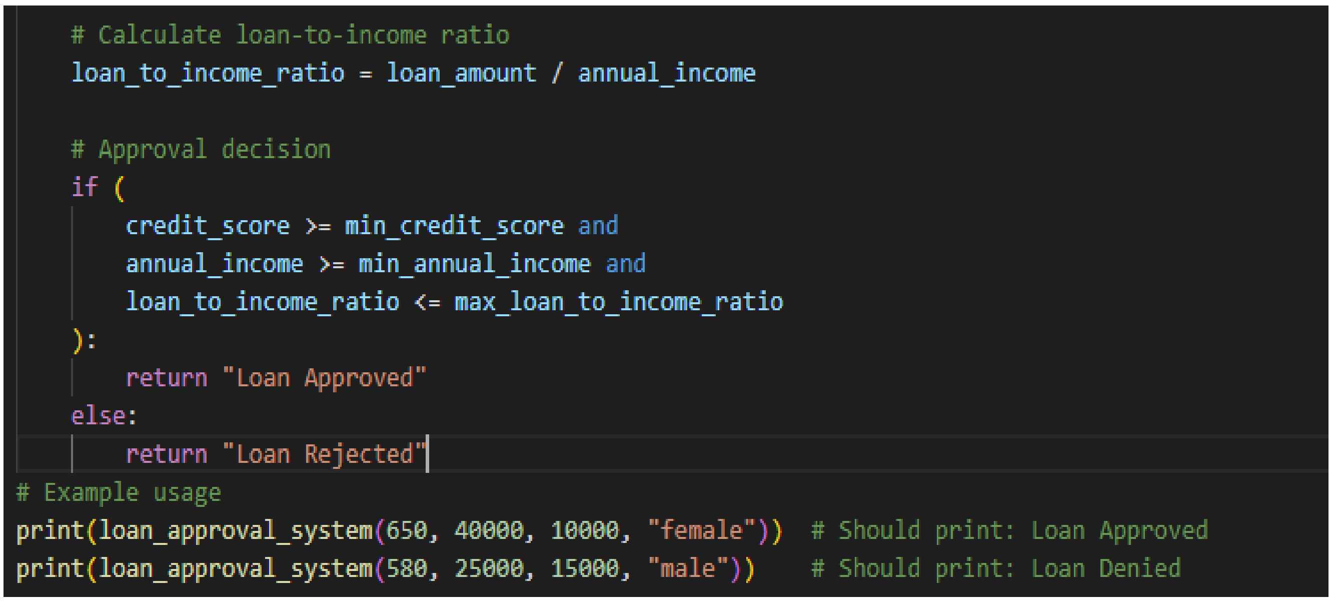
➢ Explanation of Improvements

- The updated login system no longer stores passwords in plain text, which makes it much safer.
- Passwords are protected using hashing, so the original password is never directly visible.
- During login, the system checks hashed values instead of real passwords, adding an extra layer of security.
- These improvements reduce the risk of password leakage and show better security practices in authentication systems.

Task 2: Bias Detection in AI-Generated Decision Systems



```python
# Create a loan approval system with generic criteria

def loan_approval_system(credit_score, annual_income, loan_amount, gender):
    # Define default thresholds
    min_credit_score = 600
    min_annual_income = 30000
    max_loan_to_income_ratio = 0.4

    # Define gender-specific criteria
    if gender.lower() == "female":
        min_credit_score = 620
        min_annual_income = 35000
        max_loan_to_income_ratio = 0.5
    elif gender.lower() == "male":
        min_credit_score = 600
        min_annual_income = 30000
        max_loan_to_income_ratio = 0.4
```



```python
    # Calculate loan-to-income ratio
    loan_to_income_ratio = loan_amount / annual_income

    # Approval decision
    if (
        credit_score >= min_credit_score and
        annual_income >= min_annual_income and
        loan_to_income_ratio <= max_loan_to_income_ratio
    ):
        return "Loan Approved"
    else:
        return "Loan Rejected"
# Example usage
print(loan_approval_system(650, 40000, 10000, "female"))  # Should print: Loan Approved
print(loan_approval_system(580, 25000, 15000, "male"))    # Should print: Loan Denied
```

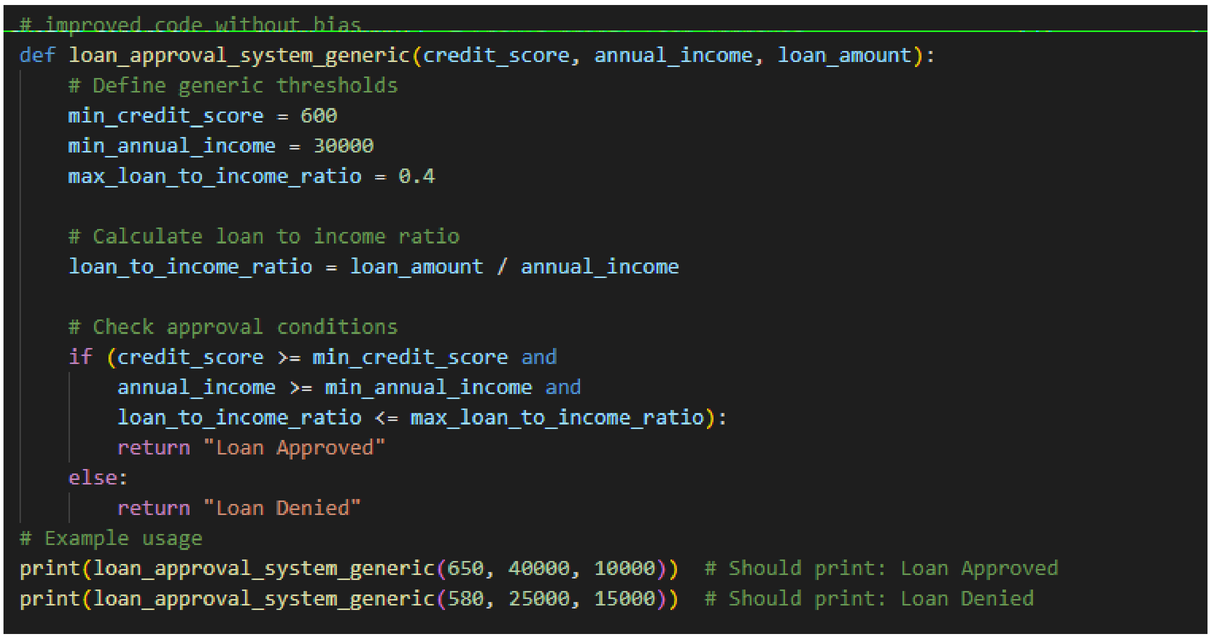
➢ **Identification of Biased Logic**

- The loan approval system shows bias because it considers personal details like gender or name, which have nothing to do with a person's ability to repay a loan.
- It applies different income requirements based on gender, leading to unequal treatment of applicants.
- Because of this, people may be approved or rejected unfairly, even if their financial situation is similar.

Discussion on Fairness Issues

- Loan approval decisions should be based only on objective financial factors such as credit score, income, and repayment capacity.
- Including gender or name in the decision-making process creates unfairness and discrimination.
- Such practices go against ethical values like equality, fairness, and justice, which should be followed in financial systems.

Improved Code:

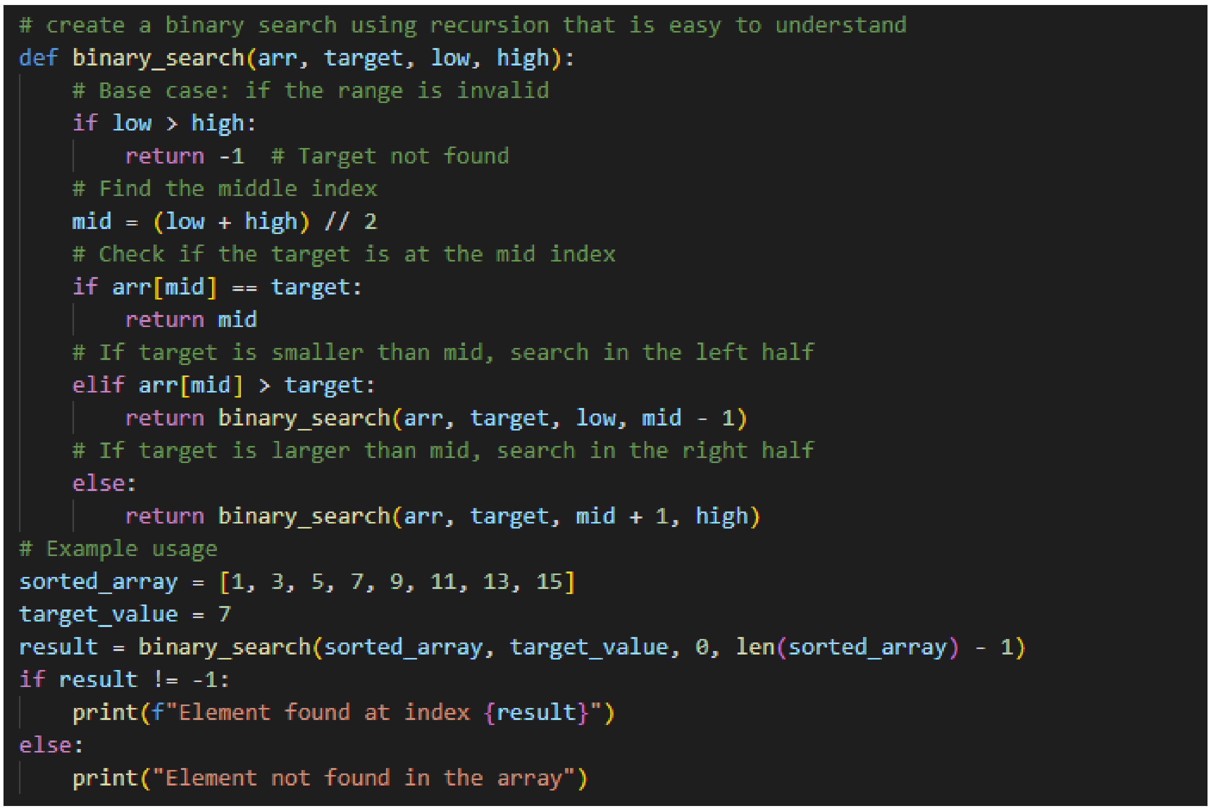


```python
# improved code without bias
def loan_approval_system_generic(credit_score, annual_income, loan_amount):
    # Define generic thresholds
    min_credit_score = 600
    min_annual_income = 30000
    max_loan_to_income_ratio = 0.4

    # Calculate loan to income ratio
    loan_to_income_ratio = loan_amount / annual_income

    # Check approval conditions
    if (credit_score >= min_credit_score and
        annual_income >= min_annual_income and
        loan_to_income_ratio <= max_loan_to_income_ratio):
        return "Loan Approved"
    else:
        return "Loan Denied"
# Example usage
print(loan_approval_system_generic(650, 40000, 10000))  # Should print: Loan Approved
print(loan_approval_system_generic(580, 25000, 15000))  # Should print: Loan Denied
```

Mitigation Strategy

- Remove gender and name from the decision process.
- Use only financial factors like income and credit score.
- Test the system with diverse data for fairness.
- Regularly review outputs to detect bias.

Task 3: Transparency and Explainability in AI-Generated Code (Recursive Binary Search)



```python
# create a binary search using recursion that is easy to understand
def binary_search(arr, target, low, high):
    # Base case: if the range is invalid
    if low > high:
        return -1  # Target not found
    # Find the middle index
    mid = (low + high) // 2
    # Check if the target is at the mid index
    if arr[mid] == target:
        return mid
    # If target is smaller than mid, search in the left half
    elif arr[mid] > target:
        return binary_search(arr, target, low, mid - 1)
    # If target is larger than mid, search in the right half
    else:
        return binary_search(arr, target, mid + 1, high)
# Example usage
sorted_array = [1, 3, 5, 7, 9, 11, 13, 15]
target_value = 7
result = binary_search(sorted_array, target_value, 0, len(sorted_array) - 1)
if result != -1:
    print(f"Element found at index {result}")
else:
    print("Element not found in the array")
```

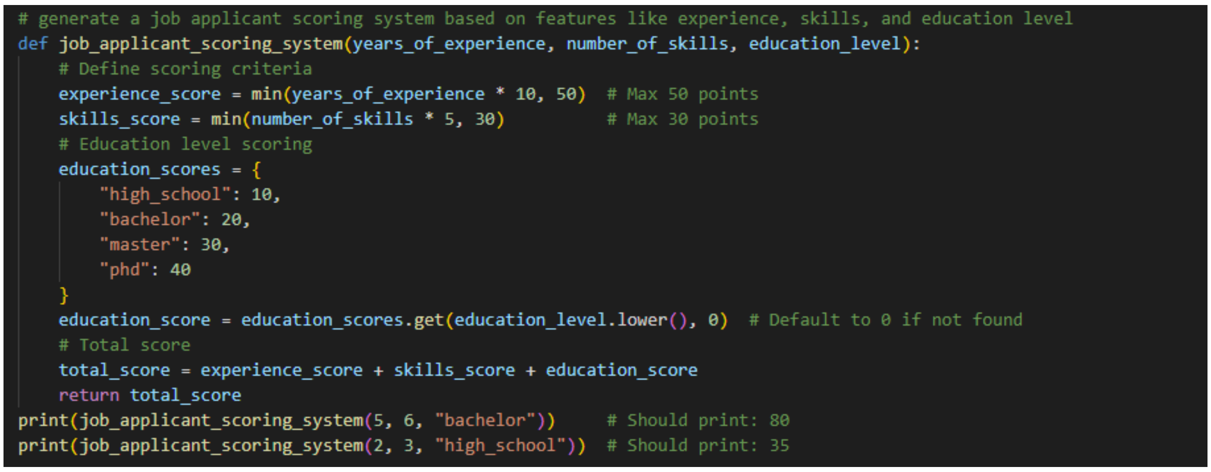
- **Step-by-Step Explanation**

  - Binary search keeps dividing the list into two halves.
  - If the start index goes beyond the end index, the element is not found.
  - The middle element is compared with the target.
  - Based on the comparison, the search continues in the left or right half until the element is found or the list ends.

- **Student Assessment on Transparency**

  - The base and recursive cases are clearly written.
  - Comments make each step easy to understand.
  - The logic is simple and beginner-friendly, which builds trust in the code.

Task 4: Ethical Evaluation of AI-Based Scoring Systems



```python
# generate a job applicant scoring system based on features like experience, skills, and education level
def job_applicant_scoring_system(years_of_experience, number_of_skills, education_level):
    # Define scoring criteria
    experience_score = min(years_of_experience * 10, 50)  # Max 50 points
    skills_score = min(number_of_skills * 5, 30)          # Max 30 points
    # Education level scoring
    education_scores = {
        "high_school": 10,
        "bachelor": 20,
        "master": 30,
        "phd": 40
    }
    education_score = education_scores.get(education_level.lower(), 0)  # Default to 0 if not found
    # Total score
    total_score = experience_score + skills_score + education_score
    return total_score
print(job_applicant_scoring_system(5, 6, "bachelor"))     # Should print: 80
print(job_applicant_scoring_system(2, 3, "high_school"))  # Should print: 35
```

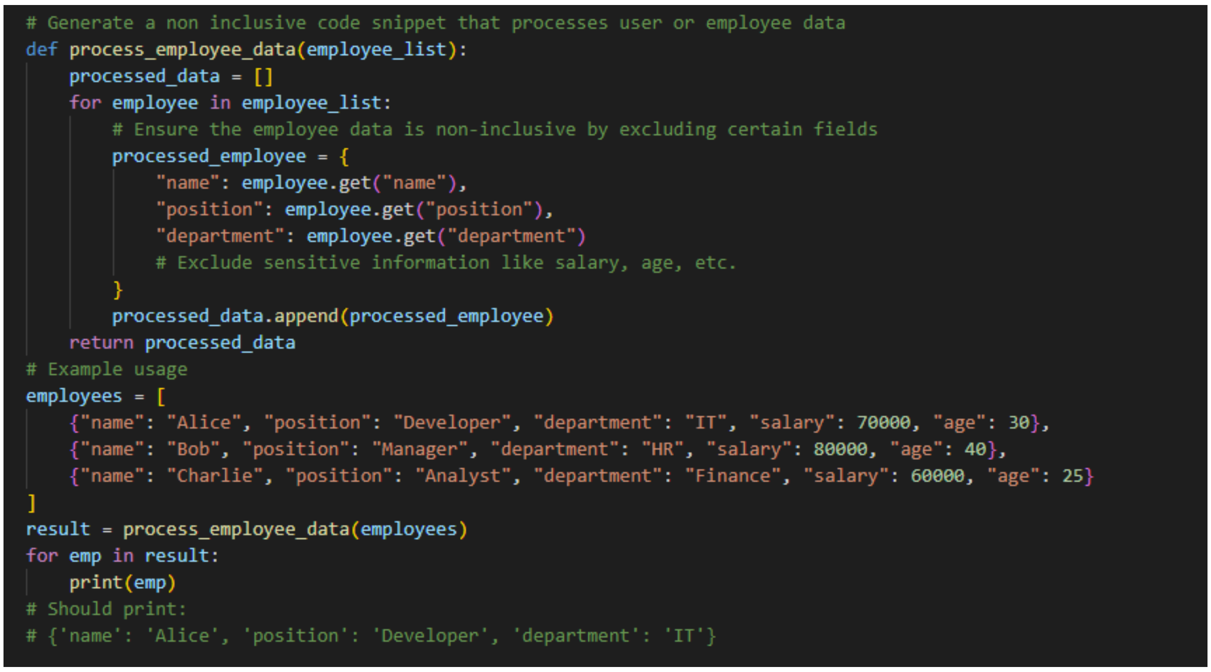
➢ Identification of Potential Bias

- The scoring system uses personal details like gender or name, which have nothing to do with job performance.
- These details can unfairly raise or lower a candidate’s score.

➢ Ethical Analysis

- A fair scoring system should depend only on job-related factors.
- Using gender or name leads to discrimination and unfair treatment.
- Ethical AI hiring systems must ensure equality and equal opportunity for all candidates.

Task 5: Inclusiveness and Ethical Variable Design

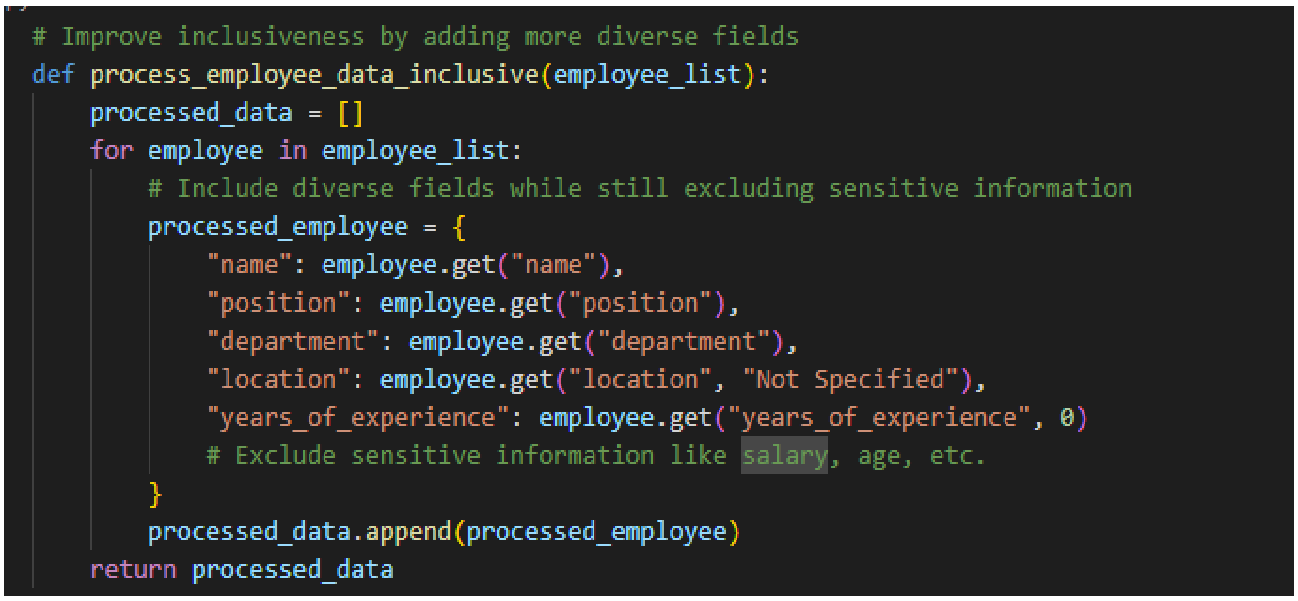


```python
# Generate a non inclusive code snippet that processes user or employee data
def process_employee_data(employee_list):
    processed_data = []
    for employee in employee_list:
        # Ensure the employee data is non-inclusive by excluding certain fields
        processed_employee = {
            "name": employee.get("name"),
            "position": employee.get("position"),
            "department": employee.get("department")
            # Exclude sensitive information like salary, age, etc.
        }
        processed_data.append(processed_employee)
    return processed_data
# Example usage
employees = [
    {"name": "Alice", "position": "Developer", "department": "IT", "salary": 70000, "age": 30},
    {"name": "Bob", "position": "Manager", "department": "HR", "salary": 80000, "age": 40},
    {"name": "Charlie", "position": "Analyst", "department": "Finance", "salary": 60000, "age": 25}
]
result = process_employee_data(employees)
for emp in result:
    print(emp)
# Should print:
# {'name': 'Alice', 'position': 'Developer', 'department': 'IT'}
```

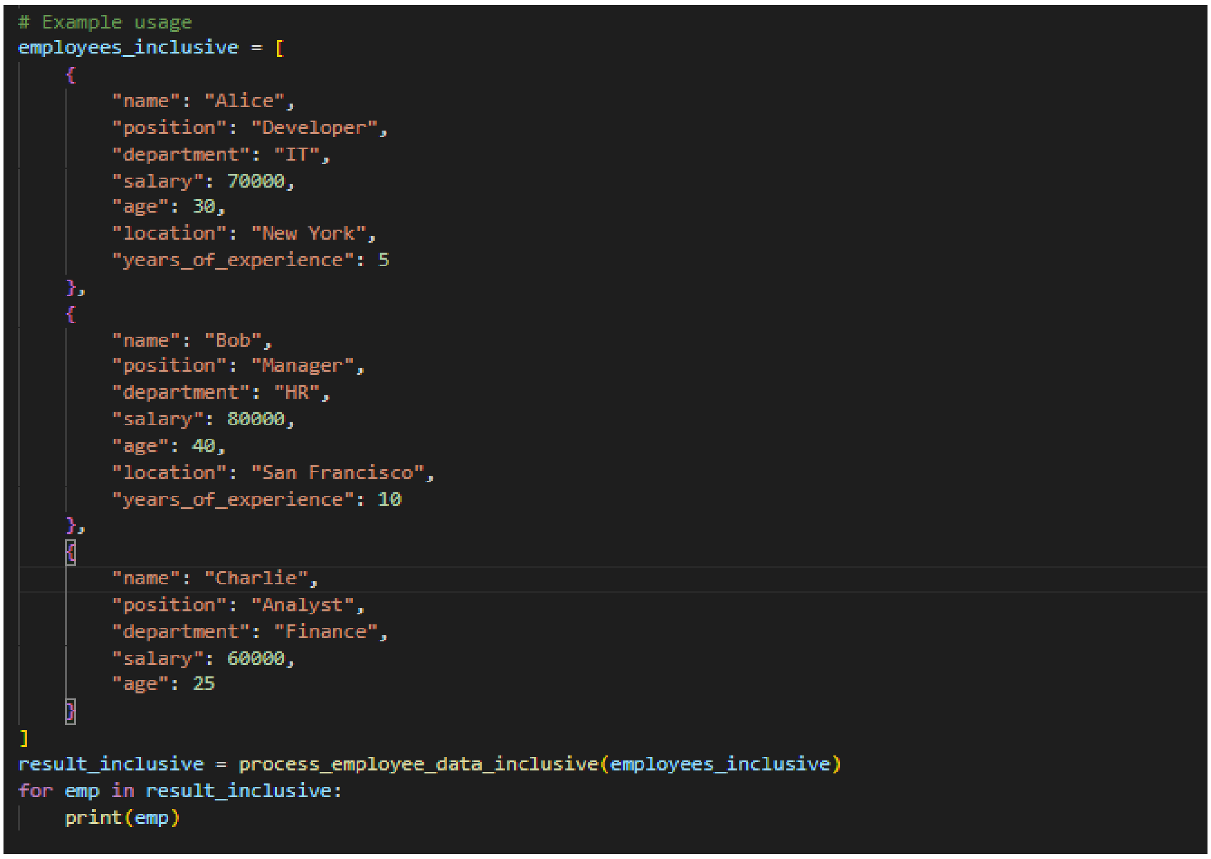
➢ **Non-Inclusive**

- The original AI-generated code uses gender-specific terms like male and female.
- It assumes a person's identity based only on gender.
- It generates outputs such as "He" or "She," which do not include non-binary identities.

Improvised(Inclusiveness) Code:



```python
# Improve inclusiveness by adding more diverse fields
def process_employee_data_inclusive(employee_list):
    processed_data = []
    for employee in employee_list:
        # Include diverse fields while still excluding sensitive information
        processed_employee = {
            "name": employee.get("name"),
            "position": employee.get("position"),
            "department": employee.get("department"),
            "location": employee.get("location", "Not Specified"),
            "years_of_experience": employee.get("years_of_experience", 0)
            # Exclude sensitive information like salary, age, etc.
        }
        processed_data.append(processed_employee)
    return processed_data
```



```python
# Example usage
employees_inclusive = [
    {
        "name": "Alice",
        "position": "Developer",
        "department": "IT",
        "salary": 70000,
        "age": 30,
        "location": "New York",
        "years_of_experience": 5
    },
    {
        "name": "Bob",
        "position": "Manager",
        "department": "HR",
        "salary": 80000,
        "age": 40,
        "location": "San Francisco",
        "years_of_experience": 10
    },
    {
        "name": "Charlie",
        "position": "Analyst",
        "department": "Finance",
        "salary": 60000,
        "age": 25
    }
]
result_inclusive = process_employee_data_inclusive(employees_inclusive)
for emp in result_inclusive:
    print(emp)
```

➢ **Revised Code (Inclusive Design)**

- The updated code uses neutral variable names that do not refer to gender.
- It avoids applying gender-based conditions unless they are absolutely necessary.
- Every user is handled equally, without making assumptions about identity.
- This approach supports respectful, fair, and inclusive coding practices.