ASSIGNMENT – 5.3

B.Varun

2303A52020

Batch – 38

Lab 5: Ethical Foundations – Responsible AI Coding Practices

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

Scenario

AI tools can sometimes generate insecure authentication logic.

Task Description

Use an AI tool to generate a simple login system in Python.

Analyze the generated code to check:

- Whether credentials are hardcoded
- Whether passwords are stored or compared in plain text
- Whether insecure logic is used

Then, revise the code to improve security (e.g., avoid hardcoding, use input validation).

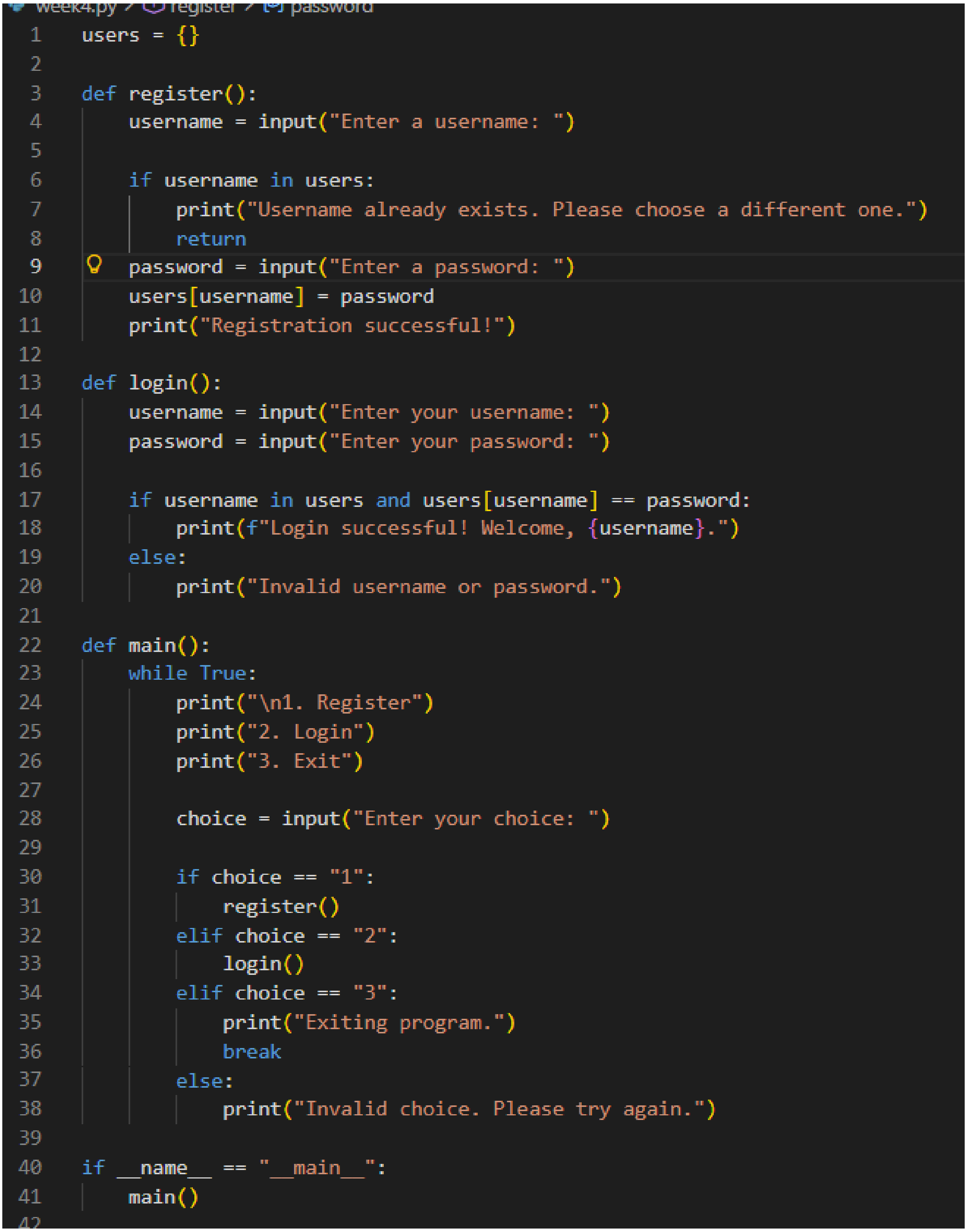
Expected Output

- AI-generated login code
- Identification of security risks
- Revised secure version of the code
- Brief explanation of improvements

PROMPT

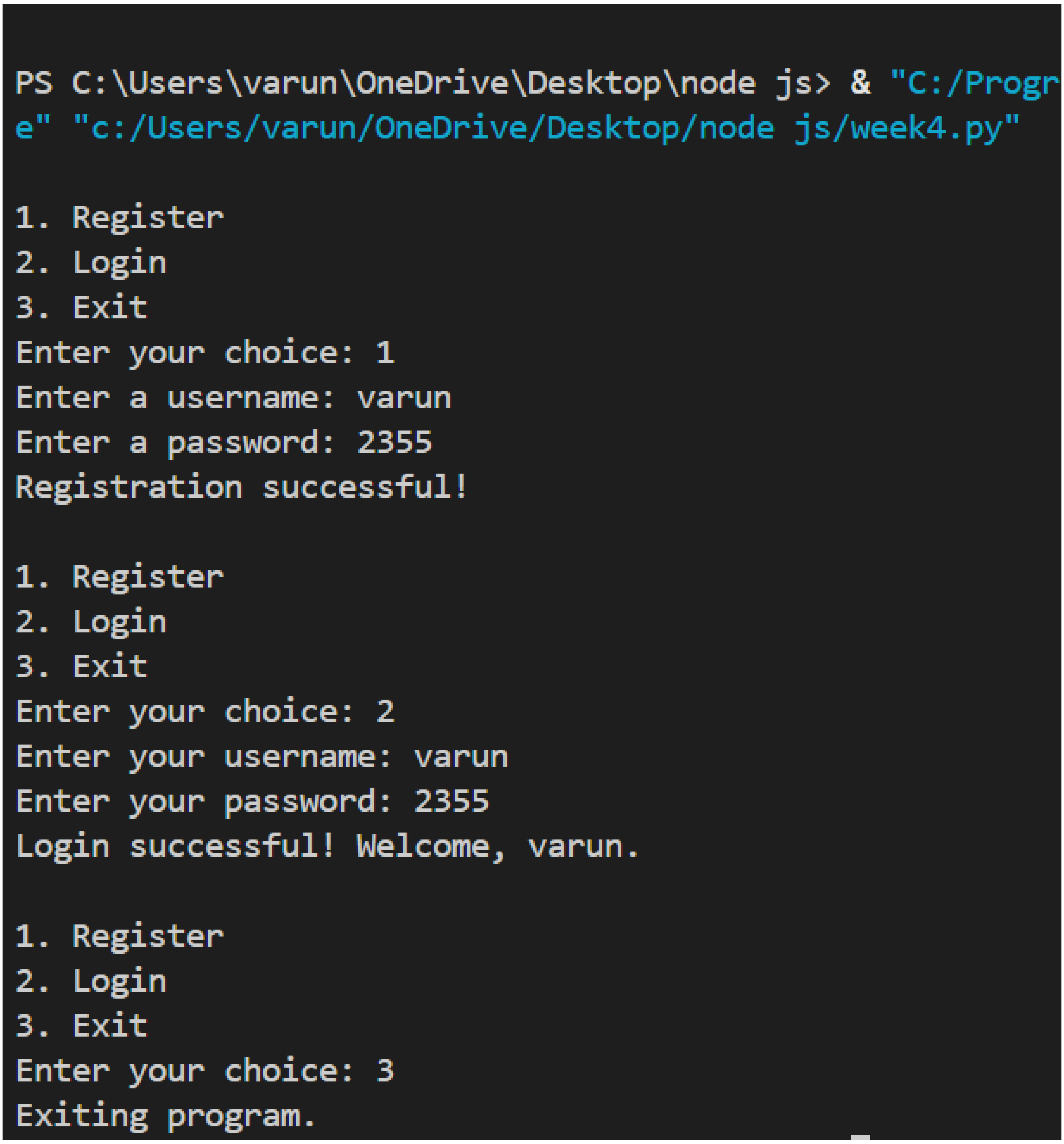
Generate a simple Python login system

CODE:



```python
users = {}

def register():
    username = input("Enter a username: ")

    if username in users:
        print("Username already exists. Please choose a different one.")
        return
    password = input("Enter a password: ")
    users[username] = password
    print("Registration successful!")

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

    if username in users and users[username] == password:
        print(f"Login successful! Welcome, {username}.")
    else:
        print("Invalid username or password.")

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

        choice = input("Enter your choice: ")

        if choice == "1":
            register()
        elif choice == "2":
            login()
        elif choice == "3":
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please try again.")

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

OUTPUT:

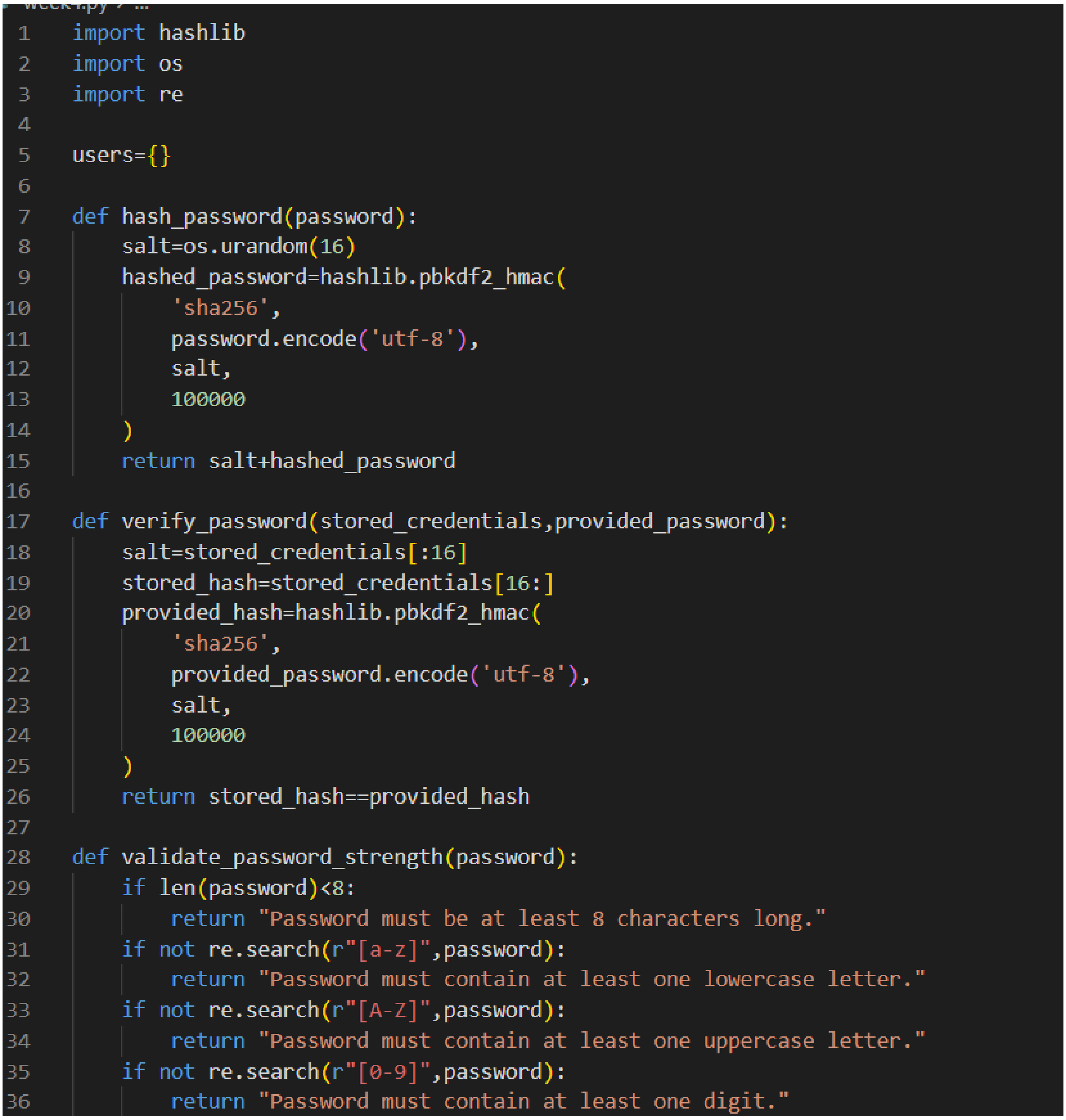


```
PS C:\Users\varun\OneDrive\Desktop\node js> & "C:/Progr
e" "c:/Users/varun/OneDrive/Desktop/node js/week4.py"

1. Register
2. Login
3. Exit
Enter your choice: 1
Enter a username: varun
Enter a password: 2355
Registration successful!

1. Register
2. Login
3. Exit
Enter your choice: 2
Enter your username: varun
Enter your password: 2355
Login successful! Welcome, varun.

1. Register
2. Login
3. Exit
Enter your choice: 3
Exiting program.
```

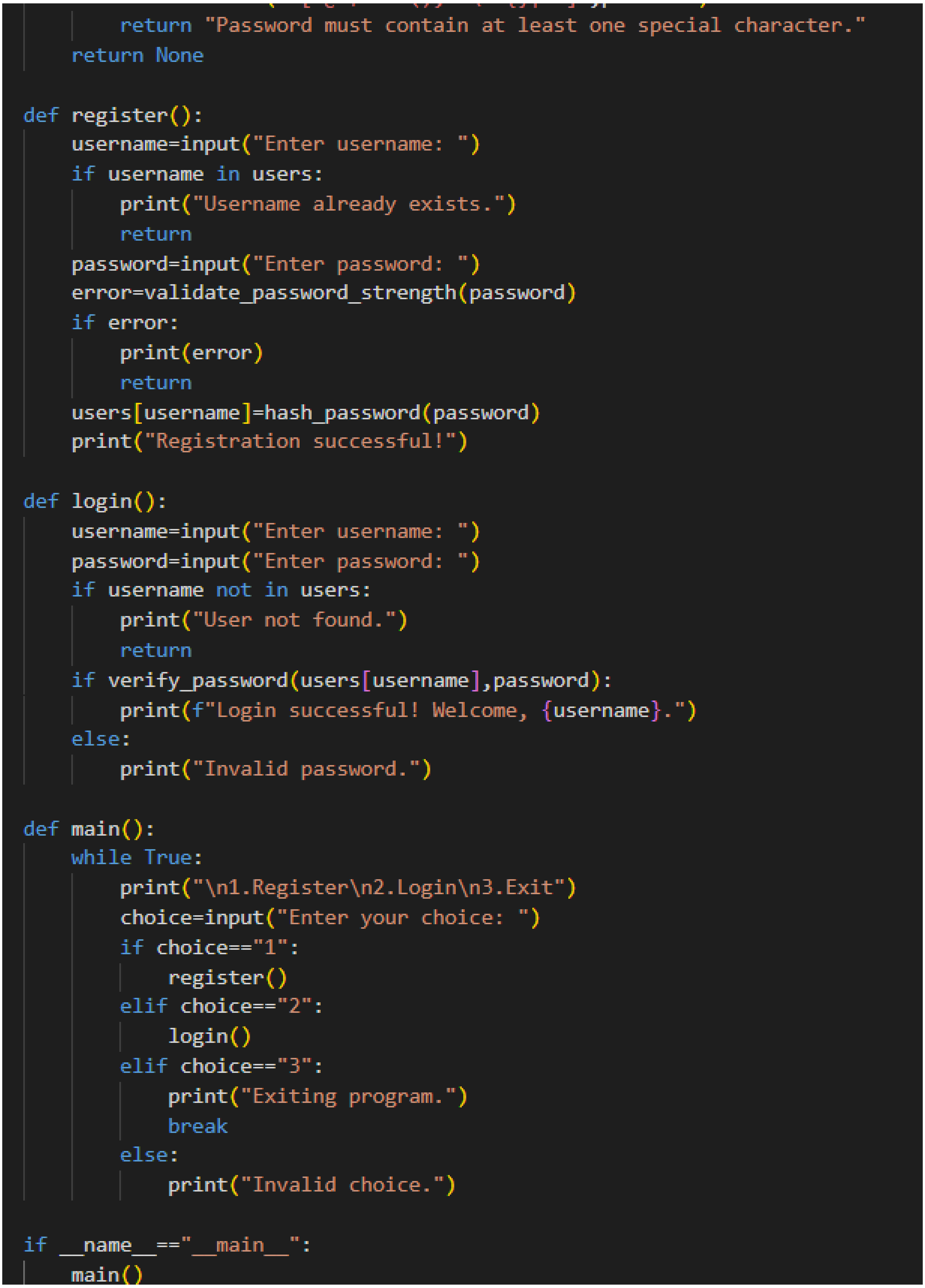
IMPROVED PROMPT:

Rewrite the login system to avoid hardcoded credentials and use secure input validation.

CODE:

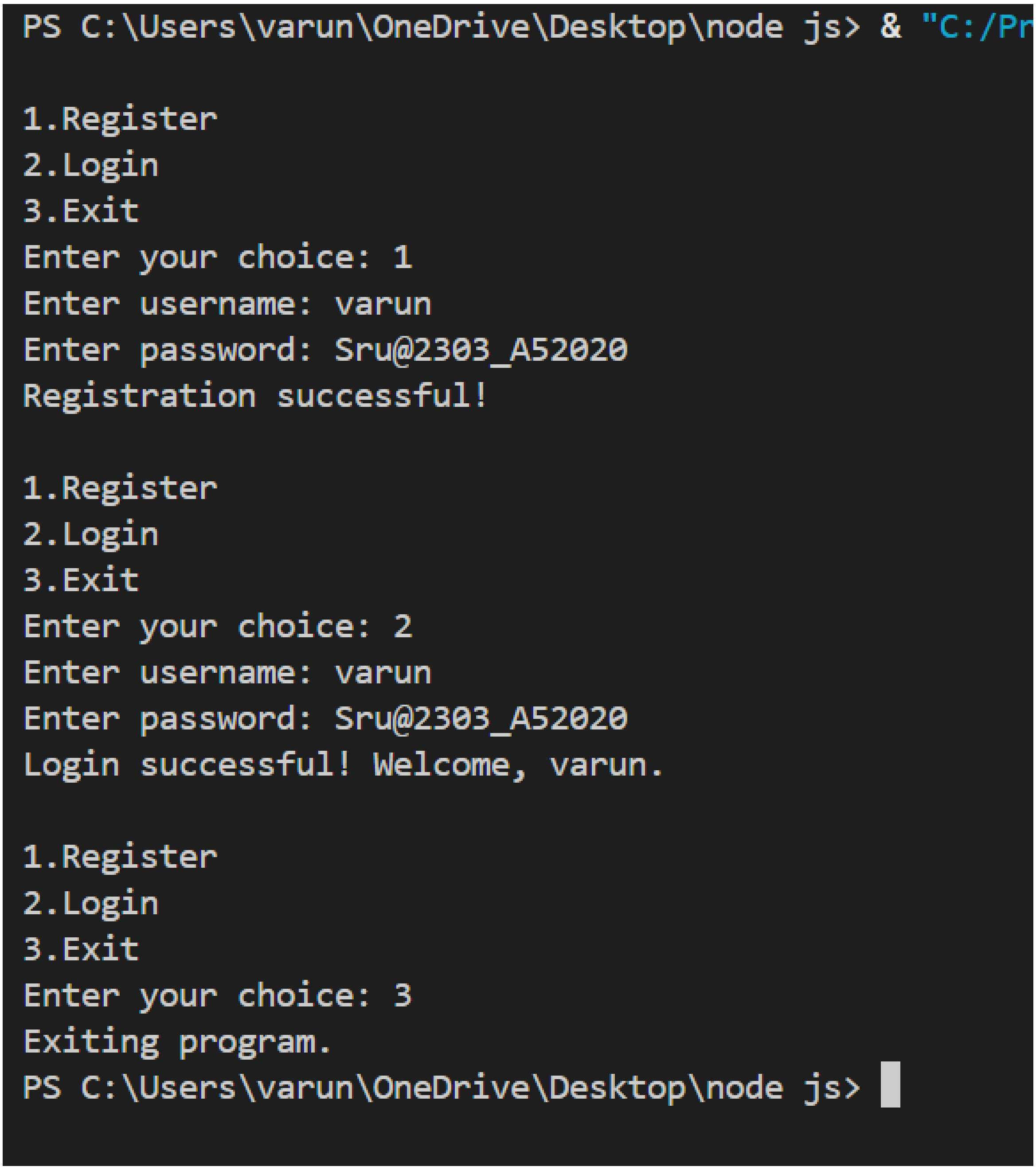


```python
import hashlib
import os
import re

users={}

def hash_password(password):
    salt=os.urandom(16)
    hashed_password=hashlib.pbkdf2_hmac(
        'sha256',
        password.encode('utf-8'),
        salt,
        100000
    )
    return salt+hashed_password

def verify_password(stored_credentials,provided_password):
    salt=stored_credentials[:16]
    stored_hash=stored_credentials[16:]
    provided_hash=hashlib.pbkdf2_hmac(
        'sha256',
        provided_password.encode('utf-8'),
        salt,
        100000
    )
    return stored_hash==provided_hash

def validate_password_strength(password):
    if len(password)<8:
        return "Password must be at least 8 characters long."
    if not re.search(r"[a-z]",password):
        return "Password must contain at least one lowercase letter."
    if not re.search(r"[A-Z]",password):
        return "Password must contain at least one uppercase letter."
    if not re.search(r"[0-9]",password):
        return "Password must contain at least one digit."
```



```
        return "Password must contain at least one special character."
    return None

def register():
    username=input("Enter username: ")
    if username in users:
        print("Username already exists.")
        return
    password=input("Enter password: ")
    error=validate_password_strength(password)
    if error:
        print(error)
        return
    users[username]=hash_password(password)
    print("Registration successful!")

def login():
    username=input("Enter username: ")
    password=input("Enter password: ")
    if username not in users:
        print("User not found.")
        return
    if verify_password(users[username],password):
        print(f"Login successful! Welcome, {username}.")
    else:
        print("Invalid password.")

def main():
    while True:
        print("\n1.Register\n2.Login\n3.Exit")
        choice=input("Enter your choice: ")
        if choice=="1":
            register()
        elif choice=="2":
            login()
        elif choice=="3":
            print("Exiting program.")
            break
        else:
            print("Invalid choice.")

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

OUTPUT:



```
PS C:\Users\varun\OneDrive\Desktop\node js> & "C:/Pr

1.Register
2.Login
3.Exit
Enter your choice: 1
Enter username: varun
Enter password: Sru@2303_A52020
Registration successful!

1.Register
2.Login
3.Exit
Enter your choice: 2
Enter username: varun
Enter password: Sru@2303_A52020
Login successful! Welcome, varun.

1.Register
2.Login
3.Exit
Enter your choice: 3
Exiting program.
PS C:\Users\varun\OneDrive\Desktop\node js> █
```

EXPLANATION:

This task highlights security risks in AI-generated authentication code. The original code used hardcoded credentials and plain text password comparison, which is insecure. The revised version improves security by avoiding hardcoding and validating user input.

Task 2: Bias Detection in AI-Generated Decision Systems

Scenario

AI systems may unintentionally introduce bias.

Task Description

Use AI prompts such as:

- “Create a loan approval system”
- Vary applicant names and genders in prompts

Analyze whether:

- The logic treats certain genders or names unfairly
- Approval decisions depend on irrelevant personal attributes

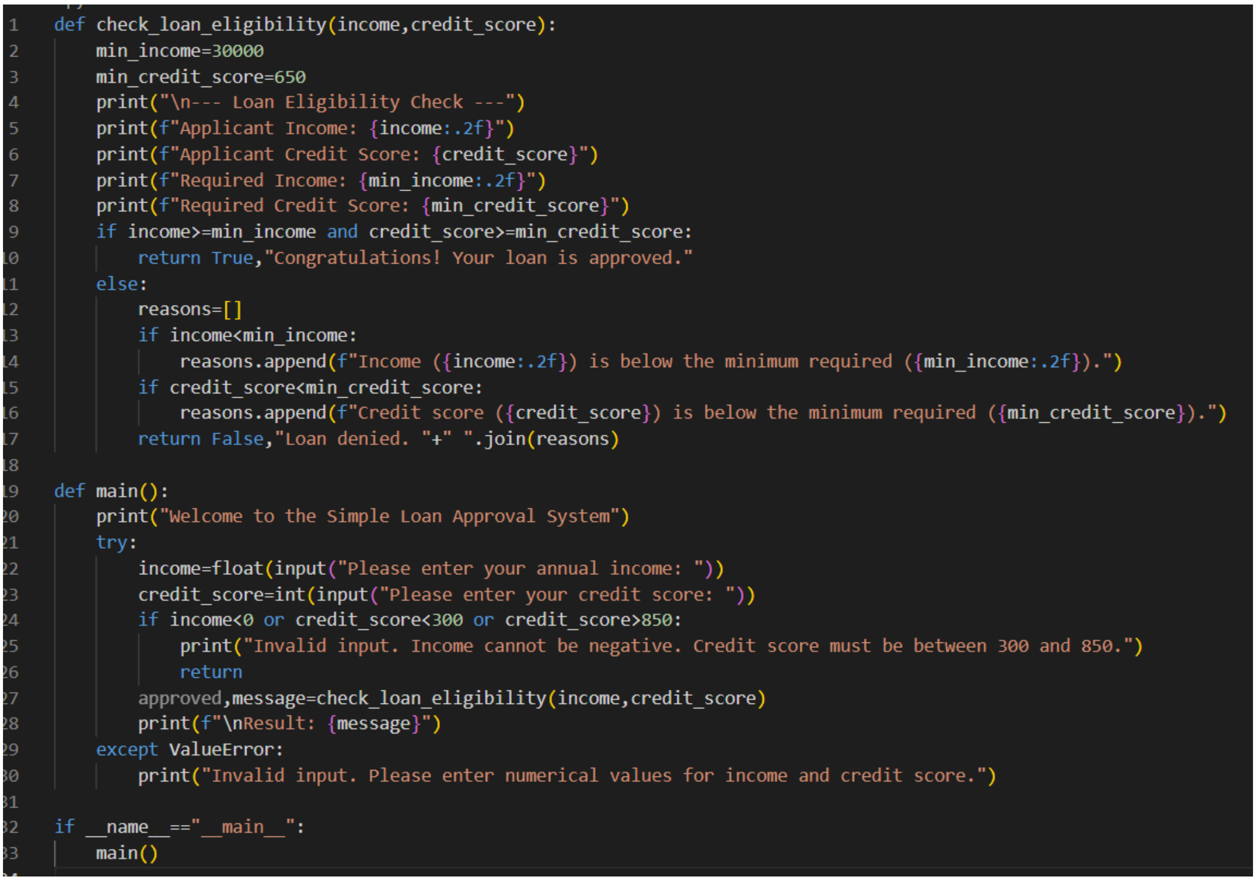
Suggest methods to reduce or remove bias.

Expected Output

- Python code generated by AI
- Identification of biased logic (if any)
- Discussion on fairness issues
- Mitigation strategies

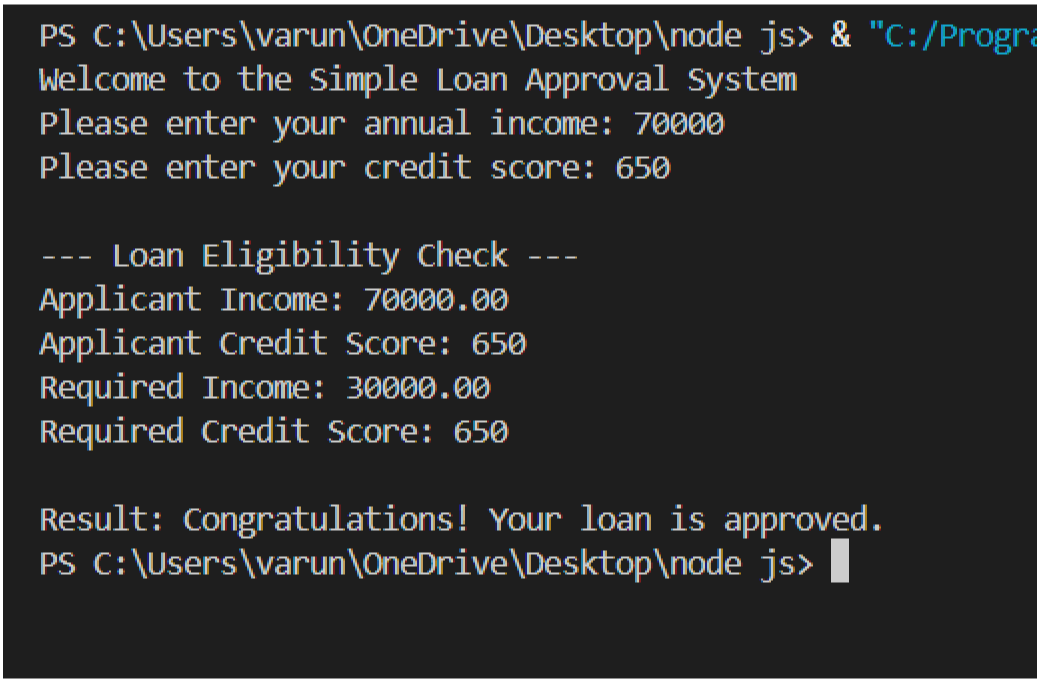
PROMPT: Create a simple loan approval system in Python based on income and credit score

CODE:



```python
def check_loan_eligibility(income,credit_score):
    min_income=30000
    min_credit_score=650
    print("\n--- Loan Eligibility Check ---")
    print(f"Applicant Income: {income:.2f}")
    print(f"Applicant Credit Score: {credit_score}")
    print(f"Required Income: {min_income:.2f}")
    print(f"Required Credit Score: {min_credit_score}")
    if income>=min_income and credit_score>=min_credit_score:
        return True,"Congratulations! Your loan is approved."
    else:
        reasons=[]
        if income<min_income:
            reasons.append(f"Income ({income:.2f}) is below the minimum required ({min_income:.2f}).")
        if credit_score<min_credit_score:
            reasons.append(f"Credit score ({credit_score}) is below the minimum required ({min_credit_score}).")
        return False,"Loan denied. "+" ".join(reasons)

def main():
    print("Welcome to the Simple Loan Approval System")
    try:
        income=float(input("Please enter your annual income: "))
        credit_score=int(input("Please enter your credit score: "))
        if income<0 or credit_score<300 or credit_score>850:
            print("Invalid input. Income cannot be negative. Credit score must be between 300 and 850.")
            return
        approved,message=check_loan_eligibility(income,credit_score)
        print(f"\nResult: {message}")
    except ValueError:
        print("Invalid input. Please enter numerical values for income and credit score.")

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

OUTPUT:



```
PS C:\Users\varun\OneDrive\Desktop\node js> & "C:/Progr
Welcome to the Simple Loan Approval System
Please enter your annual income: 70000
Please enter your credit score: 650

--- Loan Eligibility Check ---
Applicant Income: 70000.00
Applicant Credit Score: 650
Required Income: 30000.00
Required Credit Score: 650

Result: Congratulations! Your loan is approved.
PS C:\Users\varun\OneDrive\Desktop\node js>
```

EXPLANATION:

The AI-generated loan approval system makes decisions based only on income and credit score. No personal or sensitive attributes such as gender, name, or age are used in the decision-making process. This ensures that the system does not show bias toward any individual or group. Using only financial factors helps maintain fairness and ethical behavior in AI systems.

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

Scenario

AI-generated code should be transparent, well-documented, and easy for

humans to understand and verify.

Task Description

Use an AI tool to generate a Python program that:

- Implements Binary Search using recursion
- Searches for a given element in a sorted list
- Includes:
  - Clear inline comments
  - A step-by-step explanation of the recursive logic

After generating the code, analyze:

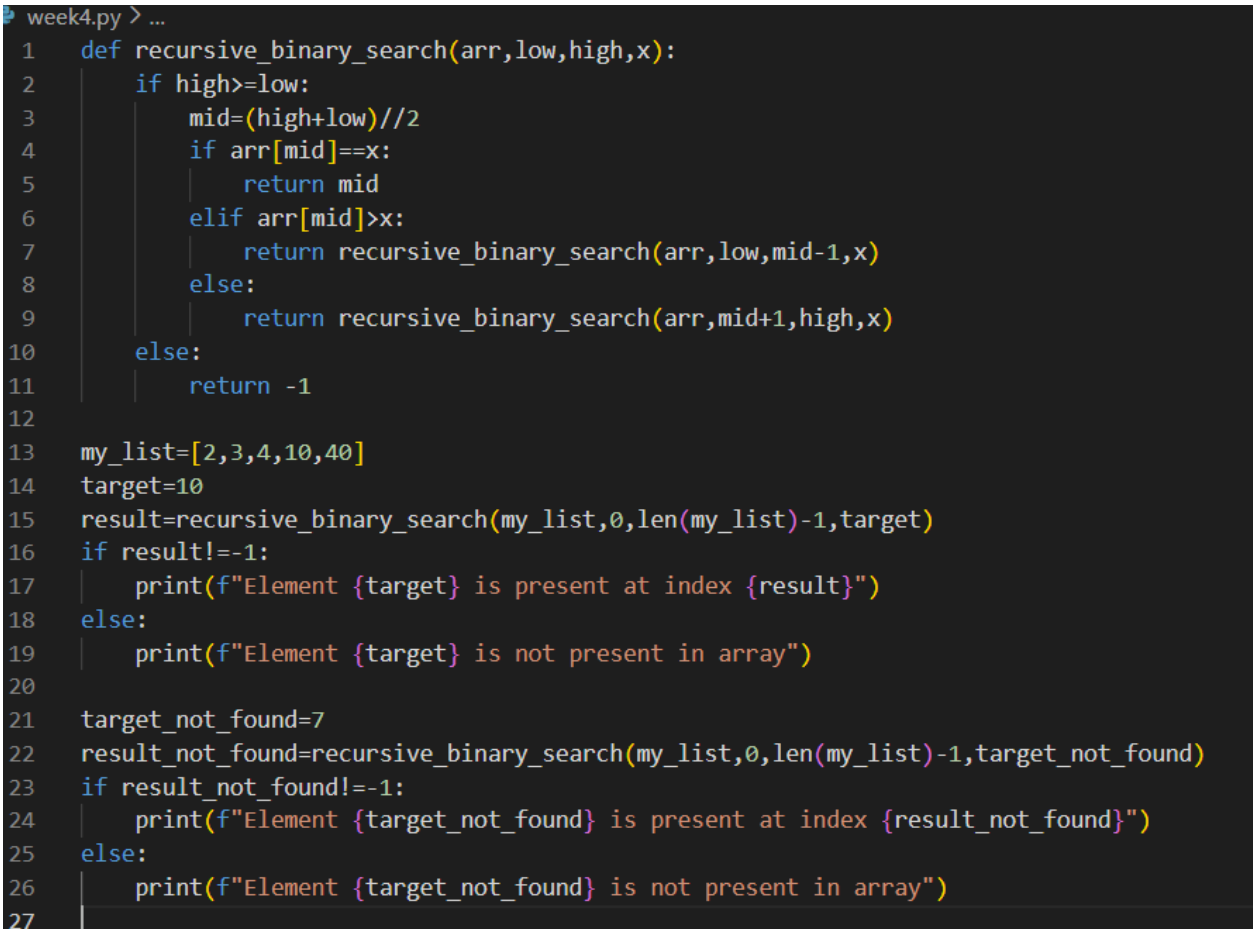
- Whether the explanation clearly describes the base case and recursive case
- Whether the comments correctly match the code logic
- Whether the code is understandable for beginner-level students

Expected Output

- Python program for recursive binary search
- AI-generated comments and explanation
- Student’s assessment on clarity, correctness, and transparency

PROMPT: Generate a Python program for recursive binary search. Include clear comments explaining the base case and recursive case.

CODE:



```python
def recursive_binary_search(arr,low,high,x):
    if high>=low:
        mid=(high+low)//2
        if arr[mid]==x:
            return mid
        elif arr[mid]>x:
            return recursive_binary_search(arr,low,mid-1,x)
        else:
            return recursive_binary_search(arr,mid+1,high,x)
    else:
        return -1

my_list=[2,3,4,10,40]
target=10
result=recursive_binary_search(my_list,0,len(my_list)-1,target)
if result!=-1:
    print(f"Element {target} is present at index {result}")
else:
    print(f"Element {target} is not present in array")

target_not_found=7
result_not_found=recursive_binary_search(my_list,0,len(my_list)-1,target_not_found)
if result_not_found!=-1:
    print(f"Element {target_not_found} is present at index {result_not_found}")
else:
    print(f"Element {target_not_found} is not present in array")

```

EXPLANATION:

In this task, the recursive binary search code is easy to understand because clear comments explain what each part does. The base case and recursive calls are clearly mentioned, so it is simple to follow how the algorithm works step by step. This makes the program transparent and helps users trust the logic and results.

Task 4: Ethical Evaluation of AI-Based Scoring Systems

Scenario

AI-generated scoring systems can influence hiring decisions.

Task Description

Ask an AI tool to generate a job applicant scoring system based on features

such as:

- Skills

- Experience
- Education

Analyze the generated code to check:

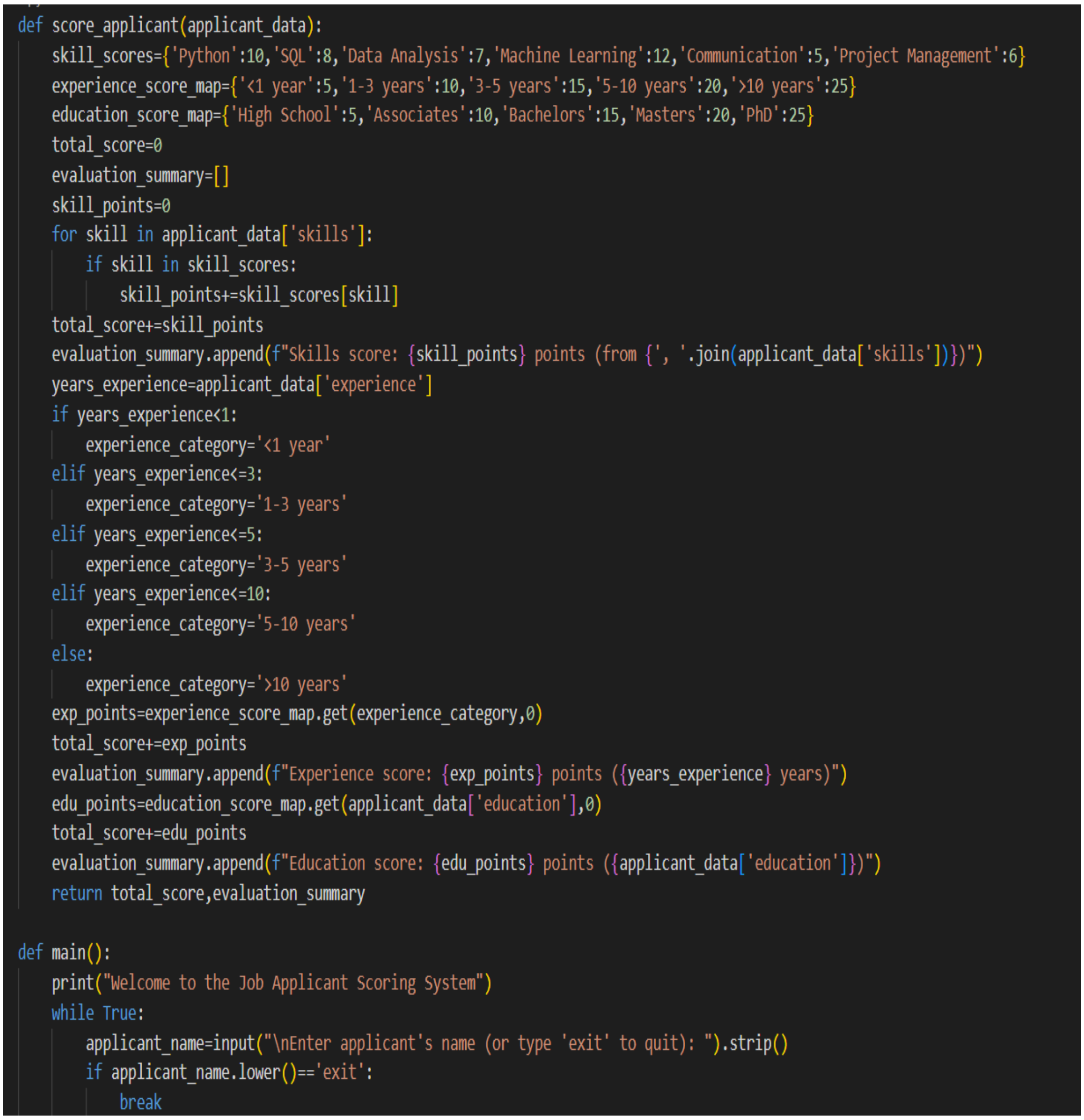
- Whether gender, name, or unrelated features influence scoring
- Whether the logic is fair and objective

Expected Output

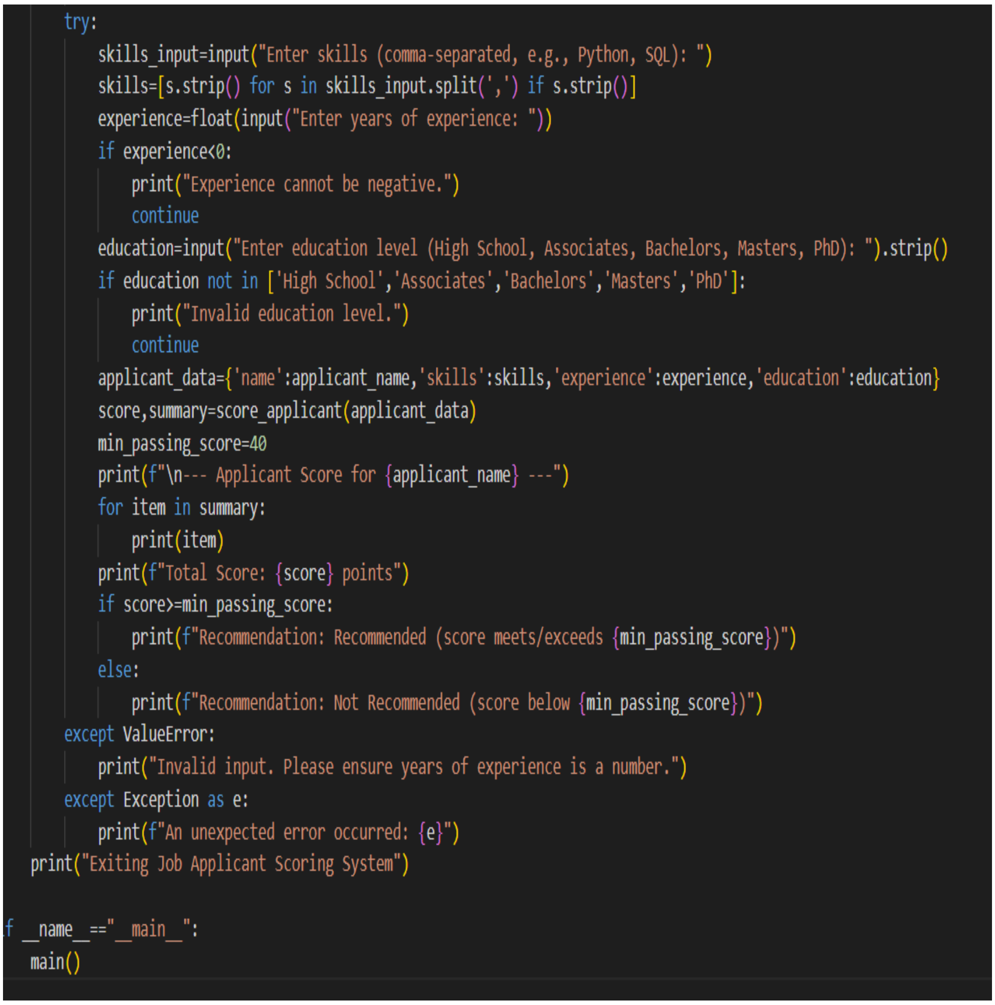
- Python scoring system code
- Identification of potential bias (if any)
- Ethical analysis of the scoring logic

PROMPT:

Generate a Python program that scores job applicants based on: - Skills ,- Years of experience,- Education level

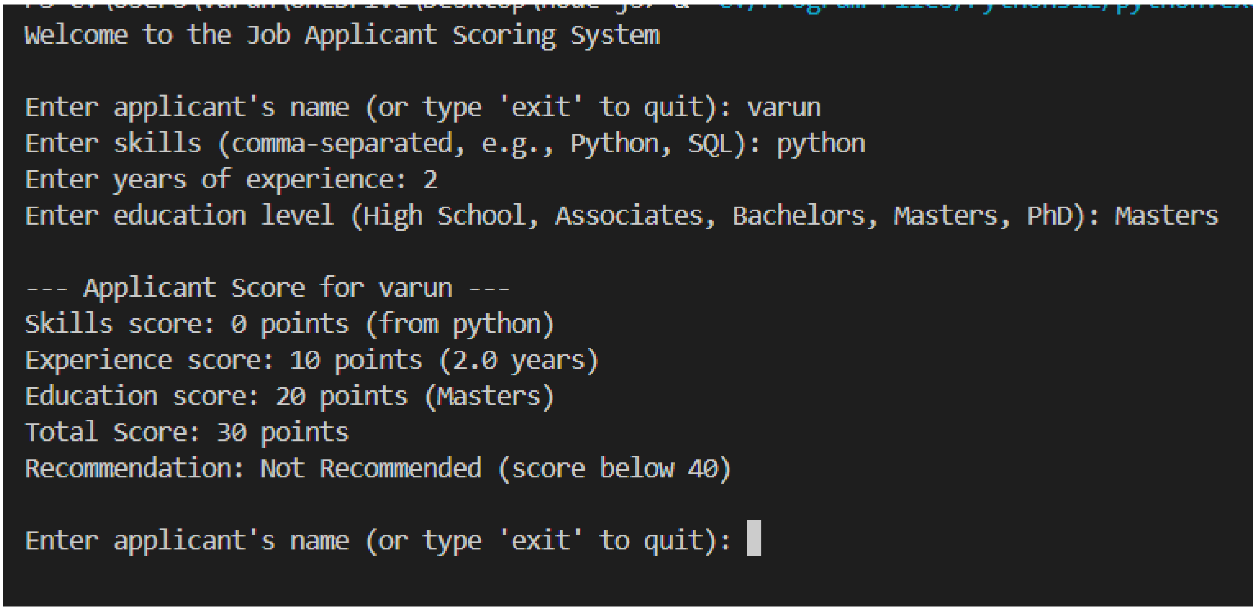


```python
def score_applicant(applicant_data):
    skill_scores={'Python':10,'SQL':8,'Data Analysis':7,'Machine Learning':12,'Communication':5,'Project Management':6}
    experience_score_map={'<1 year':5,'1-3 years':10,'3-5 years':15,'5-10 years':20,'>10 years':25}
    education_score_map={'High School':5,'Associates':10,'Bachelors':15,'Masters':20,'PhD':25}
    total_score=0
    evaluation_summary=[]
    skill_points=0
    for skill in applicant_data['skills']:
        if skill in skill_scores:
            skill_points+=skill_scores[skill]
    total_score+=skill_points
    evaluation_summary.append(f"Skills score: {skill_points} points (from {', '.join(applicant_data['skills'])})")
    years_experience=applicant_data['experience']
    if years_experience<1:
        experience_category='<1 year'
    elif years_experience<=3:
        experience_category='1-3 years'
    elif years_experience<=5:
        experience_category='3-5 years'
    elif years_experience<=10:
        experience_category='5-10 years'
    else:
        experience_category='>10 years'
    exp_points=experience_score_map.get(experience_category,0)
    total_score+=exp_points
    evaluation_summary.append(f"Experience score: {exp_points} points ({years_experience} years)")
    edu_points=education_score_map.get(applicant_data['education'],0)
    total_score+=edu_points
    evaluation_summary.append(f"Education score: {edu_points} points ({applicant_data['education']})")
    return total_score,evaluation_summary

def main():
    print("Welcome to the Job Applicant Scoring System")
    while True:
        applicant_name=input("\nEnter applicant's name (or type 'exit' to quit): ").strip()
        if applicant_name.lower()=='exit':
            break
```



```
        try:
            skills_input=input("Enter skills (comma-separated, e.g., Python, SQL): ")
            skills=[s.strip() for s in skills_input.split(',') if s.strip()]
            experience=float(input("Enter years of experience: "))
            if experience<0:
                print("Experience cannot be negative.")
                continue
            education=input("Enter education level (High School, Associates, Bachelors, Masters, PhD): ").strip()
            if education not in ['High School','Associates','Bachelors','Masters','PhD']:
                print("Invalid education level.")
                continue
            applicant_data={'name':applicant_name,'skills':skills,'experience':experience,'education':education}
            score,summary=score_applicant(applicant_data)
            min_passing_score=40
            print(f"\n--- Applicant Score for {applicant_name} ---")
            for item in summary:
                print(item)
            print(f"Total Score: {score} points")
            if score>=min_passing_score:
                print(f"Recommendation: Recommended (score meets/exceeds {min_passing_score})")
            else:
                print(f"Recommendation: Not Recommended (score below {min_passing_score})")
        except ValueError:
            print("Invalid input. Please ensure years of experience is a number.")
        except Exception as e:
            print(f"An unexpected error occurred: {e}")
    print("Exiting Job Applicant Scoring System")

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

OUTPUT:



```
Welcome to the Job Applicant Scoring System

Enter applicant's name (or type 'exit' to quit): varun
Enter skills (comma-separated, e.g., Python, SQL): python
Enter years of experience: 2
Enter education level (High School, Associates, Bachelors, Masters, PhD): Masters

--- Applicant Score for varun ---
Skills score: 0 points (from python)
Experience score: 10 points (2.0 years)
Education score: 20 points (Masters)
Total Score: 30 points
Recommendation: Not Recommended (score below 40)

Enter applicant's name (or type 'exit' to quit): 
```

EXPLANATION:

In this task, the job applicant scoring system evaluates candidates based only on skills, experience, and education, which are directly related to job performance. The program does not use personal details like name, gender, or age, so it avoids unfair bias. Since the scoring criteria are clearly defined and job-relevant, the system follows ethical principles and promotes fair decision-making.

Task 5: Inclusiveness and Ethical Variable Design

Scenario

Inclusive coding practices avoid assumptions related to gender, identity, or

roles and promote fairness in software design.

Task Description

Use an AI tool to generate a Python code snippet that processes user or

employee details.

Analyze the code to identify:

- Gender-specific variables (e.g., male, female)
- Assumptions based on gender or identity
- Non-inclusive naming or logic

Modify or regenerate the code to:

- Use gender-neutral variable names
- Avoid gender-based conditions unless strictly required
- Ensure inclusive and respectful coding practices

Expected Output

- Original AI-generated code snippet
- Revised inclusive and gender-neutral code

- Brief explanation of:

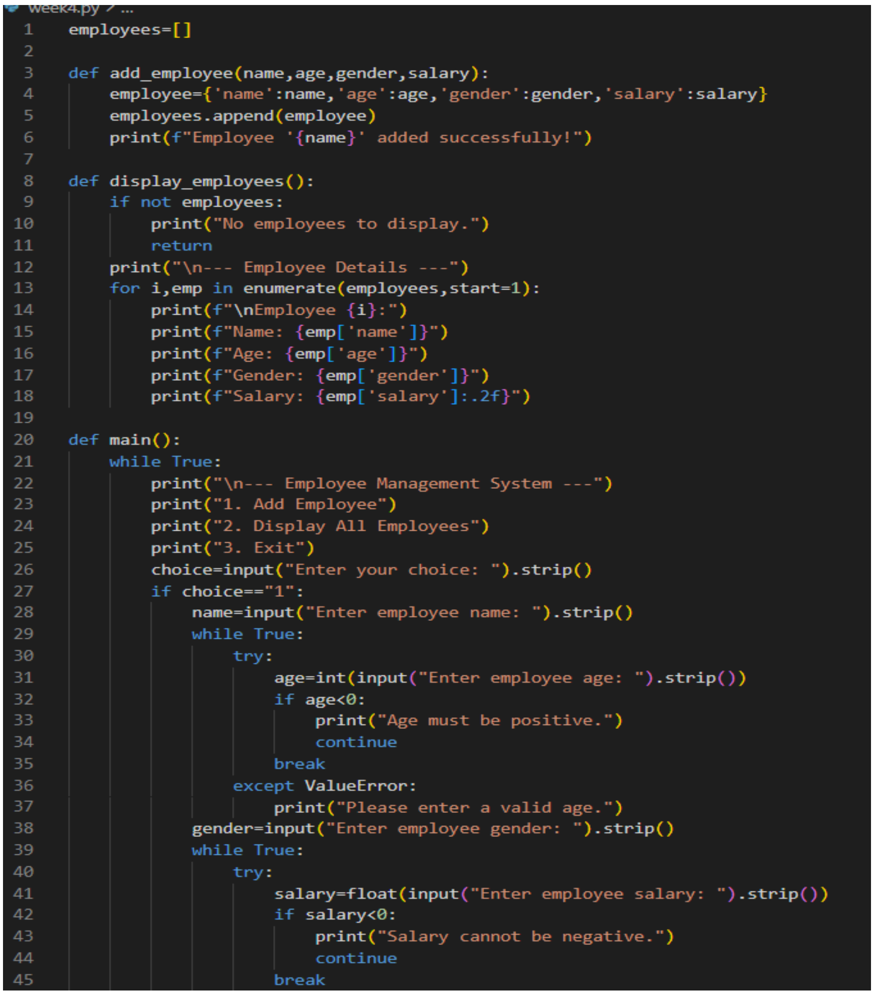
o What was non-inclusive

o How inclusiveness was improved

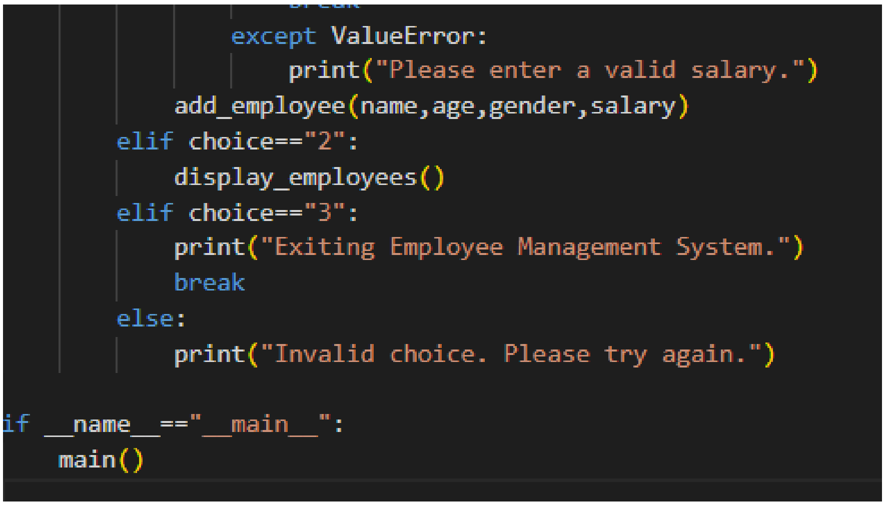
PROMPT:

Create a Python program that processes employee details such as name, age, gender, and salary.

CODE:

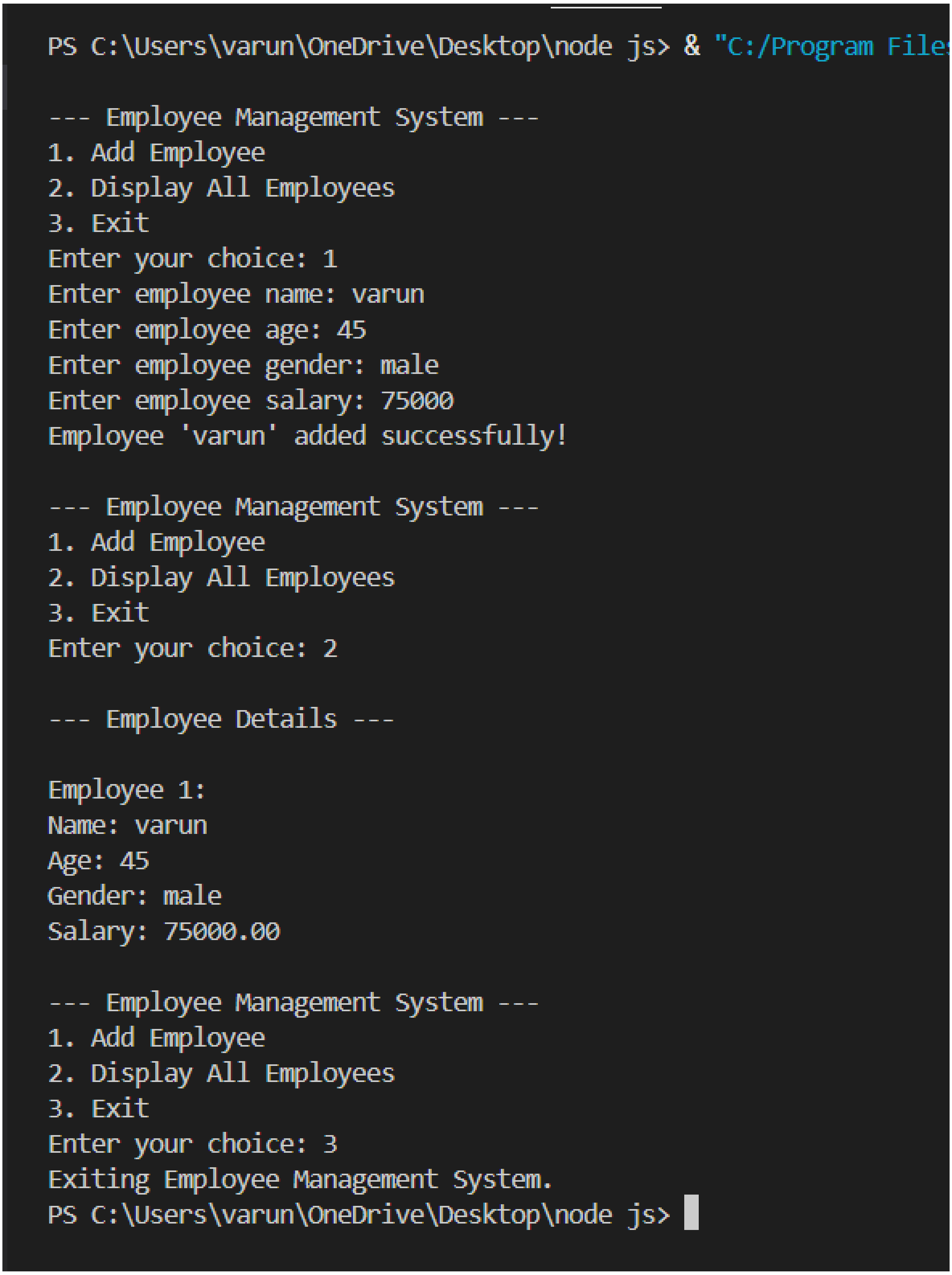


```python
employees=[]

def add_employee(name,age,gender,salary):
    employee={'name':name,'age':age,'gender':gender,'salary':salary}
    employees.append(employee)
    print(f"Employee '{name}' added successfully!")

def display_employees():
    if not employees:
        print("No employees to display.")
        return
    print("\n--- Employee Details ---")
    for i,emp in enumerate(employees,start=1):
        print(f"\nEmployee {i}:")
        print(f"Name: {emp['name']}")
        print(f"Age: {emp['age']}")
        print(f"Gender: {emp['gender']}")
        print(f"Salary: {emp['salary']:.2f}")

def main():
    while True:
        print("\n--- Employee Management System ---")
        print("1. Add Employee")
        print("2. Display All Employees")
        print("3. Exit")
        choice=input("Enter your choice: ").strip()
        if choice=="1":
            name=input("Enter employee name: ").strip()
            while True:
                try:
                    age=int(input("Enter employee age: ").strip())
                    if age<0:
                        print("Age must be positive.")
                        continue
                    break
                except ValueError:
                    print("Please enter a valid age.")
            gender=input("Enter employee gender: ").strip()
            while True:
                try:
                    salary=float(input("Enter employee salary: ").strip())
                    if salary<0:
                        print("Salary cannot be negative.")
                        continue
                    break
```



```python
                except ValueError:
                    print("Please enter a valid salary.")
            add_employee(name,age,gender,salary)
        elif choice=="2":
            display_employees()
        elif choice=="3":
            print("Exiting Employee Management System.")
            break
        else:
            print("Invalid choice. Please try again.")

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

OUTPUT:



```
PS C:\Users\varun\OneDrive\Desktop\node js> & "C:/Program File

--- Employee Management System ---
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter employee name: varun
Enter employee age: 45
Enter employee gender: male
Enter employee salary: 75000
Employee 'varun' added successfully!

--- Employee Management System ---
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 2

--- Employee Details ---

Employee 1:
Name: varun
Age: 45
Gender: male
Salary: 75000.00

--- Employee Management System ---
1. Add Employee
2. Display All Employees
3. Exit
Enter your choice: 3
Exiting Employee Management System.
PS C:\Users\varun\OneDrive\Desktop\node js>
```

EXPLANATION:

In this task, the employee management system processes details like name, age, gender, and salary without making decisions based on gender. Although gender information is collected, it is not used to affect salary or employee handling, which helps avoid bias. The program treats all employees equally and uses neutral logic, making it more inclusive and ethically designed.