# AI Assisted Coding Lab 5

Shivani Pambidi

2303A52172

Batch-41

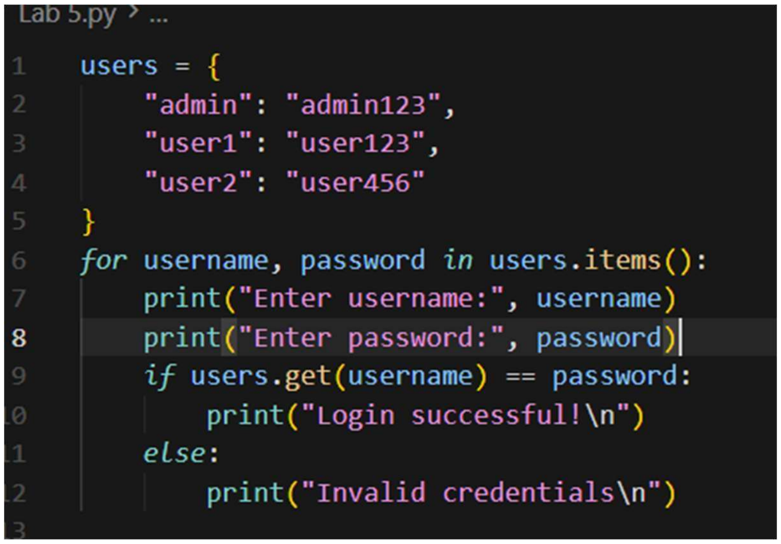
**Question 1:**

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

**AI-Generated PROMPT:**

Generate a simple Python program for user registration and login using username and password.

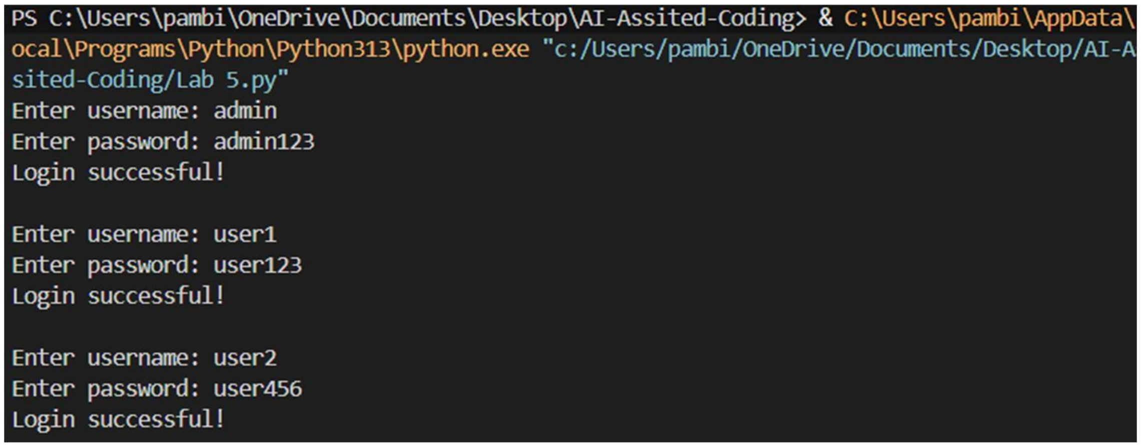
CODE:



Lab 5.py > ...

```python
users = {
    "admin": "admin123",
    "user1": "user123",
    "user2": "user456"
}
for username, password in users.items():
    print("Enter username:", username)
    print("Enter password:", password)
    if users.get(username) == password:
        print("Login successful!\n")
    else:
        print("Invalid credentials\n")
```

OUTPUT:



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\
ocal\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-A
sited-Coding/Lab 5.py"
Enter username: admin
Enter password: admin123
Login successful!

Enter username: user1
Enter password: user123
Login successful!

Enter username: user2
Enter password: user456
Login successful!
```

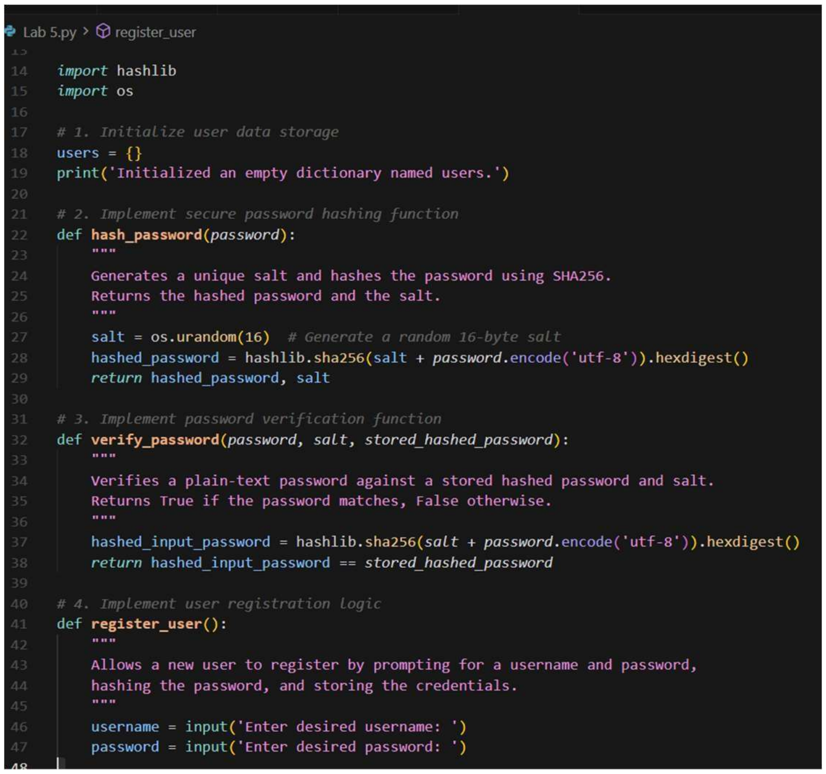
## JUSTIFICATION:

In the insecure code, usernames and passwords are hardcoded directly in the program and stored in plain text. This makes the user credentials visible to anyone who accesses the code. There is no protection for privacy, and passwords are even displayed during login. Such coding practices can easily lead to misuse or hacking. This shows that blindly trusting AIgenerated code without review can create serious security risks.

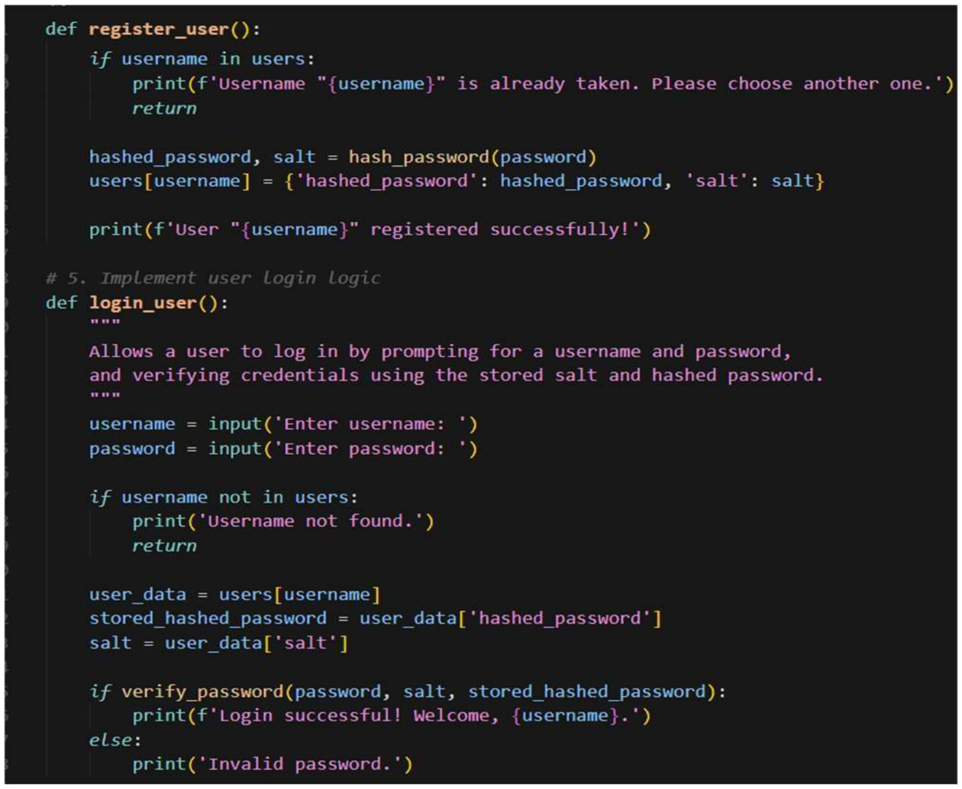
## REVISED PROMPT:

Generate a Python program for a secure user registration and login system.The program should allow users to choose between register, login, or exit.During registration, the username and password should be taken from the user and stored securely.During login, the entered username and password should be verified and a success message should be shown.The program should work for multiple users and keep running until the user chooses exit.

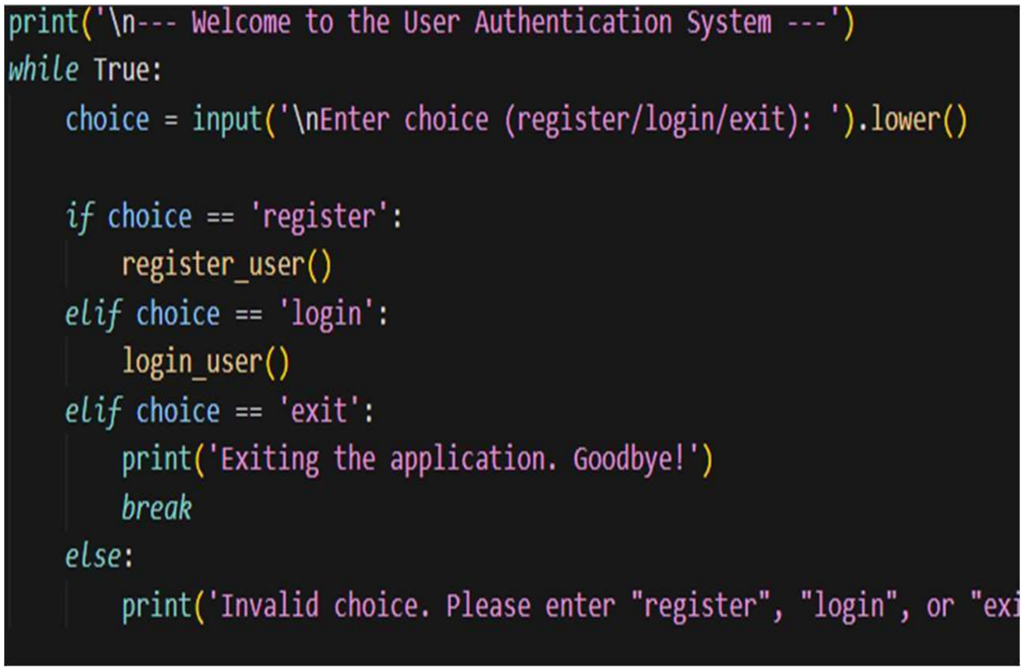
## CODE:


Lab 5.py > register_user

```python
import hashlib
import os

# 1. Initialize user data storage
users = {}
print('Initialized an empty dictionary named users.')

# 2. Implement secure password hashing function
def hash_password(password):
    """
    Generates a unique salt and hashes the password using SHA256.
    Returns the hashed password and the salt.
    """
    salt = os.urandom(16)  # Generate a random 16-byte salt
    hashed_password = hashlib.sha256(salt + password.encode('utf-8')).hexdigest()
    return hashed_password, salt

# 3. Implement password verification function
def verify_password(password, salt, stored_hashed_password):
    """
    Verifies a plain-text password against a stored hashed password and salt.
    Returns True if the password matches, False otherwise.
    """
    hashed_input_password = hashlib.sha256(salt + password.encode('utf-8')).hexdigest()
    return hashed_input_password == stored_hashed_password

# 4. Implement user registration logic
def register_user():
    """
    Allows a new user to register by prompting for a username and password,
    hashing the password, and storing the credentials.
    """
    username = input('Enter desired username: ')
    password = input('Enter desired password: ')
```

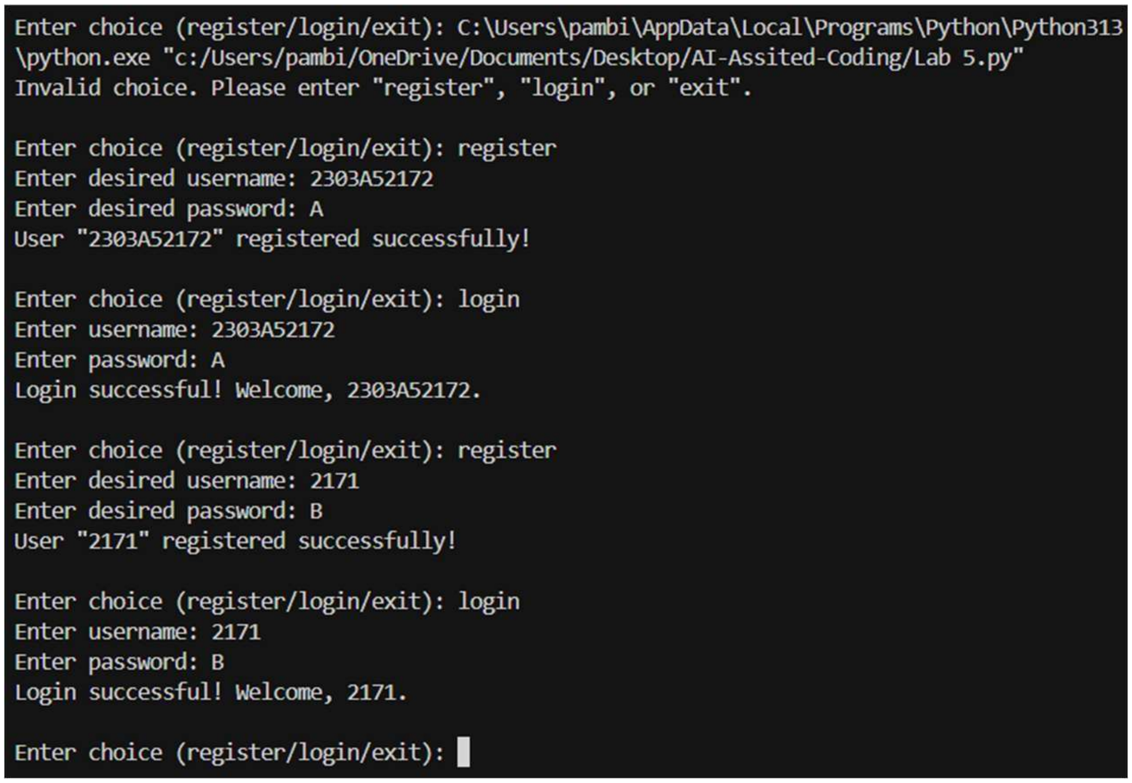


```python
def register_user():
    if username in users:
        print(f'Username "{username}" is already taken. Please choose another one.')
        return

    hashed_password, salt = hash_password(password)
    users[username] = {'hashed_password': hashed_password, 'salt': salt}

    print(f'User "{username}" registered successfully!')

# 5. Implement user login logic
def login_user():
    """
    Allows a user to log in by prompting for a username and password,
    and verifying credentials using the stored salt and hashed password.
    """
    username = input('Enter username: ')
    password = input('Enter password: ')

    if username not in users:
        print('Username not found.')
        return

    user_data = users[username]
    stored_hashed_password = user_data['hashed_password']
    salt = user_data['salt']

    if verify_password(password, salt, stored_hashed_password):
        print(f'Login successful! Welcome, {username}.')
    else:
        print('Invalid password.')
```



```
print('\n--- Welcome to the User Authentication System ---')
while True:
    choice = input('\nEnter choice (register/login/exit): ').lower()

    if choice == 'register':
        register_user()
    elif choice == 'login':
        login_user()
    elif choice == 'exit':
        print('Exiting the application. Goodbye!')
        break
    else:
        print('Invalid choice. Please enter "register", "login", or "exi
```

**OUTPUT:**



```
Enter choice (register/login/exit): C:\Users\pambi\AppData\Local\Programs\Python\Python313
\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 5.py"
Invalid choice. Please enter "register", "login", or "exit".

Enter choice (register/login/exit): register
Enter desired username: 2303A52172
Enter desired password: A
User "2303A52172" registered successfully!

Enter choice (register/login/exit): login
Enter username: 2303A52172
Enter password: A
Login successful! Welcome, 2303A52172.

Enter choice (register/login/exit): register
Enter desired username: 2171
Enter desired password: B
User "2171" registered successfully!

Enter choice (register/login/exit): login
Enter username: 2171
Enter password: B
Login successful! Welcome, 2171.

Enter choice (register/login/exit): 
```

**JUSTIFICATION:**

In the secure code, passwords are protected using hashing instead of storing them in plain text. The actual passwords are never stored or displayed, which helps in maintaining user privacy. Even if someone gains access to the data, the passwords cannot be easily understood. This approach follows ethical and responsible coding prac ces. It highlights the importance of human involvement in reviewing and improving AIgenerated code.

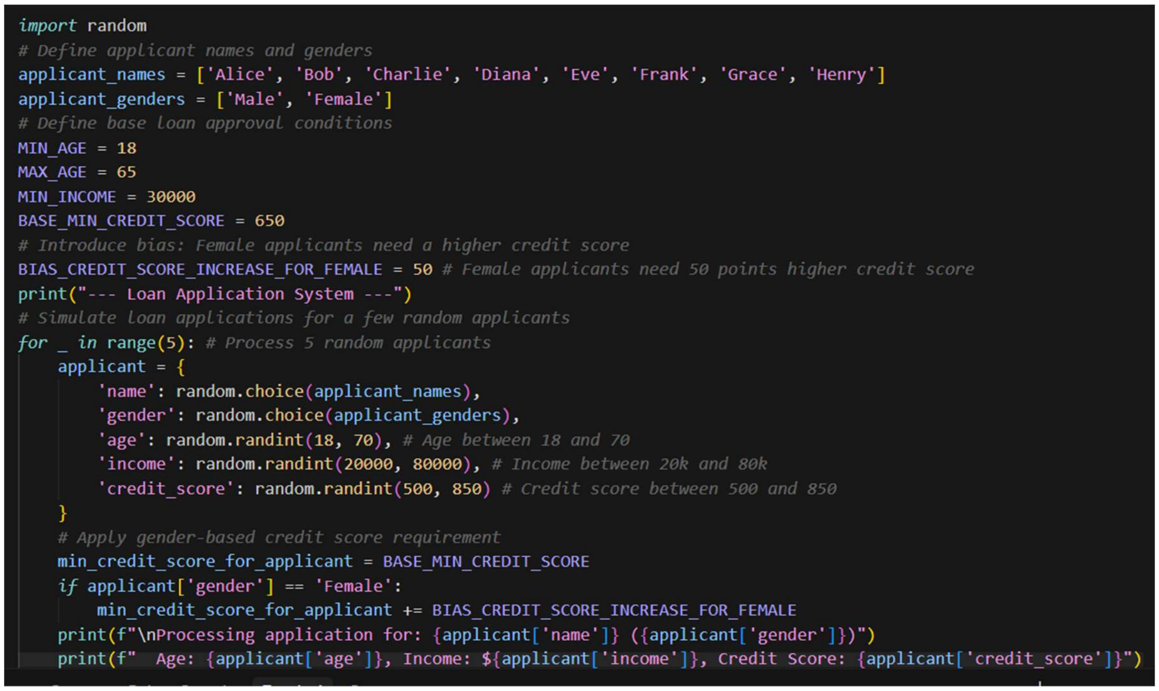
## Question 2:

**Task 2:** Bias Detection in AI-Generated Decision Systems

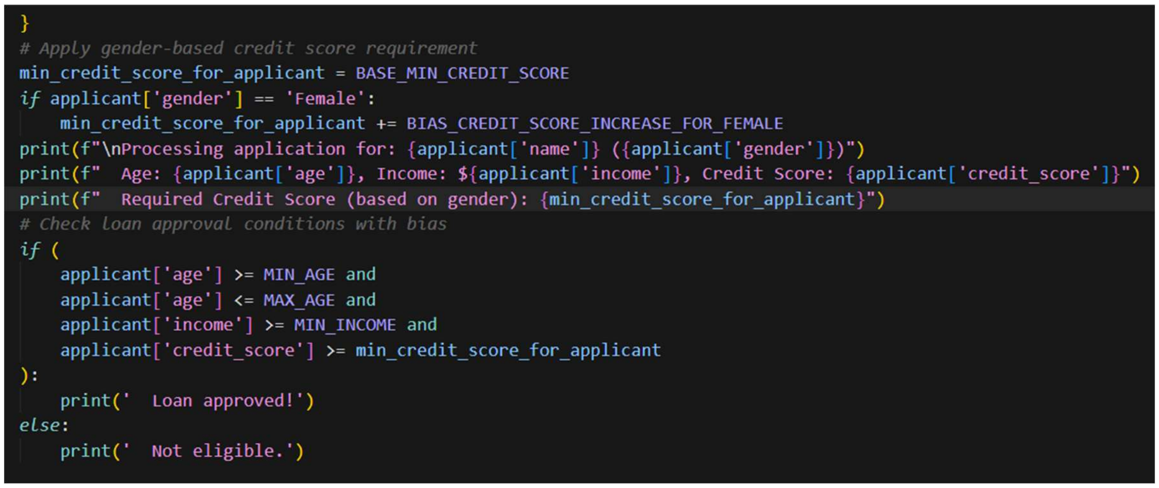
**AI-Generated(Biased Version)PROMPT:**

Create a simple Python loan approval system.Approve loans based on applicant name, gender, income, and credit score.

**CODE:**

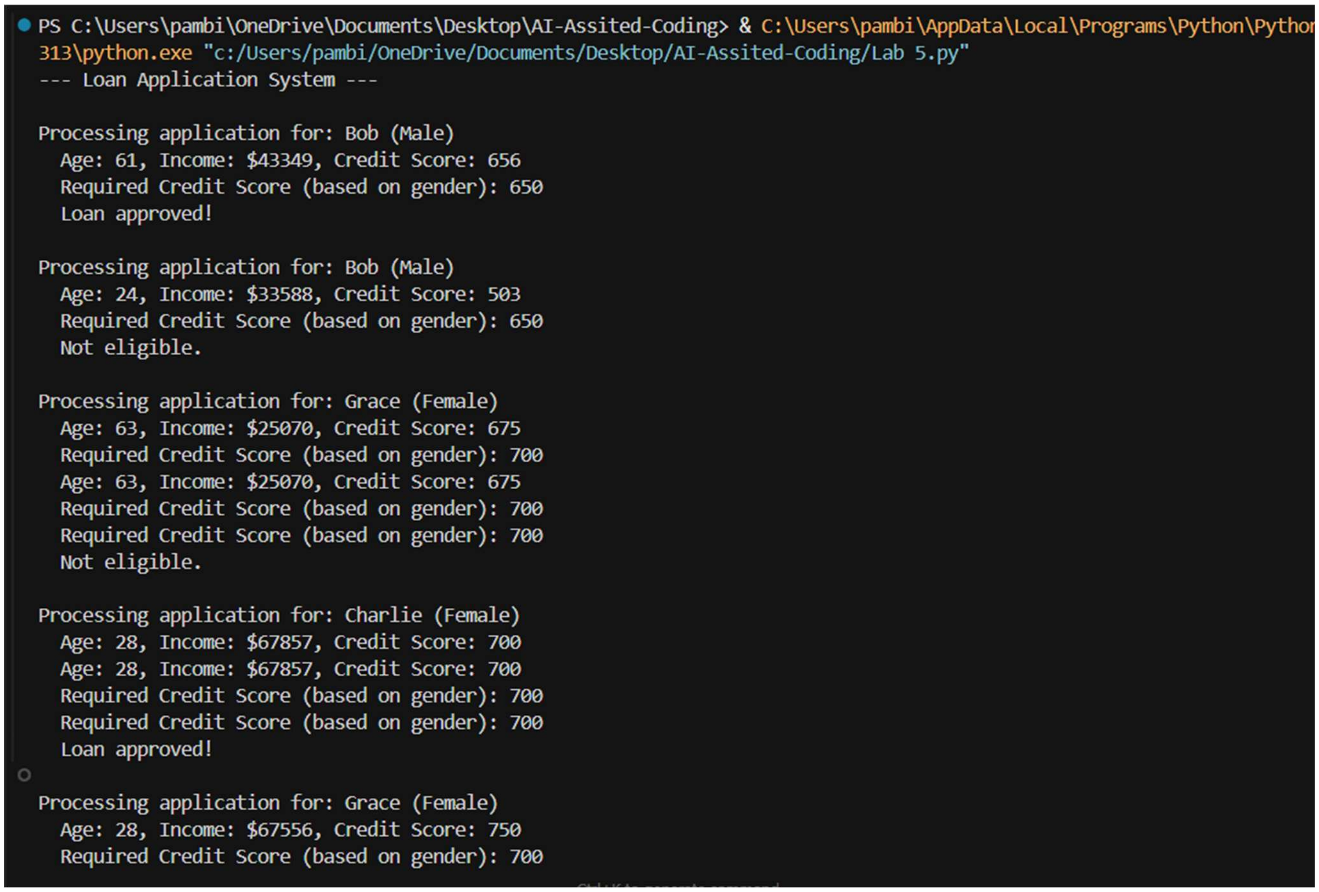


```python
import random
# Define applicant names and genders
applicant_names = ['Alice', 'Bob', 'Charlie', 'Diana', 'Eve', 'Frank', 'Grace', 'Henry']
applicant_genders = ['Male', 'Female']
# Define base loan approval conditions
MIN_AGE = 18
MAX_AGE = 65
MIN_INCOME = 30000
BASE_MIN_CREDIT_SCORE = 650
# Introduce bias: Female applicants need a higher credit score
BIAS_CREDIT_SCORE_INCREASE_FOR_FEMALE = 50 # Female applicants need 50 points higher credit score
print("--- Loan Application System ---")
# Simulate loan applications for a few random applicants
for _ in range(5): # Process 5 random applicants
    applicant = {
        'name': random.choice(applicant_names),
        'gender': random.choice(applicant_genders),
        'age': random.randint(18, 70), # Age between 18 and 70
        'income': random.randint(20000, 80000), # Income between 20k and 80k
        'credit_score': random.randint(500, 850) # Credit score between 500 and 850
    }
    # Apply gender-based credit score requirement
    min_credit_score_for_applicant = BASE_MIN_CREDIT_SCORE
    if applicant['gender'] == 'Female':
        min_credit_score_for_applicant += BIAS_CREDIT_SCORE_INCREASE_FOR_FEMALE
    print(f"\nProcessing application for: {applicant['name']} ({applicant['gender']})")
    print(f"  Age: {applicant['age']}, Income: ${applicant['income']}, Credit Score: {applicant['credit_score']}")
```



```
}
# Apply gender-based credit score requirement
min_credit_score_for_applicant = BASE_MIN_CREDIT_SCORE
if applicant['gender'] == 'Female':
    min_credit_score_for_applicant += BIAS_CREDIT_SCORE_INCREASE_FOR_FEMALE
print(f"\nProcessing application for: {applicant['name']} ({applicant['gender']})")
print(f"  Age: {applicant['age']}, Income: ${applicant['income']}, Credit Score: {applicant['credit_score']}")
print(f"  Required Credit Score (based on gender): {min_credit_score_for_applicant}")
# Check loan approval conditions with bias
if (
    applicant['age'] >= MIN_AGE and
    applicant['age'] <= MAX_AGE and
    applicant['income'] >= MIN_INCOME and
    applicant['credit_score'] >= min_credit_score_for_applicant
):
    print('  Loan approved!')
else:
    print('  Not eligible.')
```

**OUTPUT:**



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Pythor
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 5.py"
--- Loan Application System ---

Processing application for: Bob (Male)
  Age: 61, Income: $43349, Credit Score: 656
  Required Credit Score (based on gender): 650
  Loan approved!

Processing application for: Bob (Male)
  Age: 24, Income: $33588, Credit Score: 503
  Required Credit Score (based on gender): 650
  Not eligible.

Processing application for: Grace (Female)
  Age: 63, Income: $25070, Credit Score: 675
  Required Credit Score (based on gender): 700
  Age: 63, Income: $25070, Credit Score: 675
  Required Credit Score (based on gender): 700
  Required Credit Score (based on gender): 700
  Not eligible.

Processing application for: Charlie (Female)
  Age: 28, Income: $67857, Credit Score: 700
  Age: 28, Income: $67857, Credit Score: 700
  Required Credit Score (based on gender): 700
  Required Credit Score (based on gender): 700
  Loan approved!

Processing application for: Grace (Female)
  Age: 28, Income: $67556, Credit Score: 750
  Required Credit Score (based on gender): 700
```

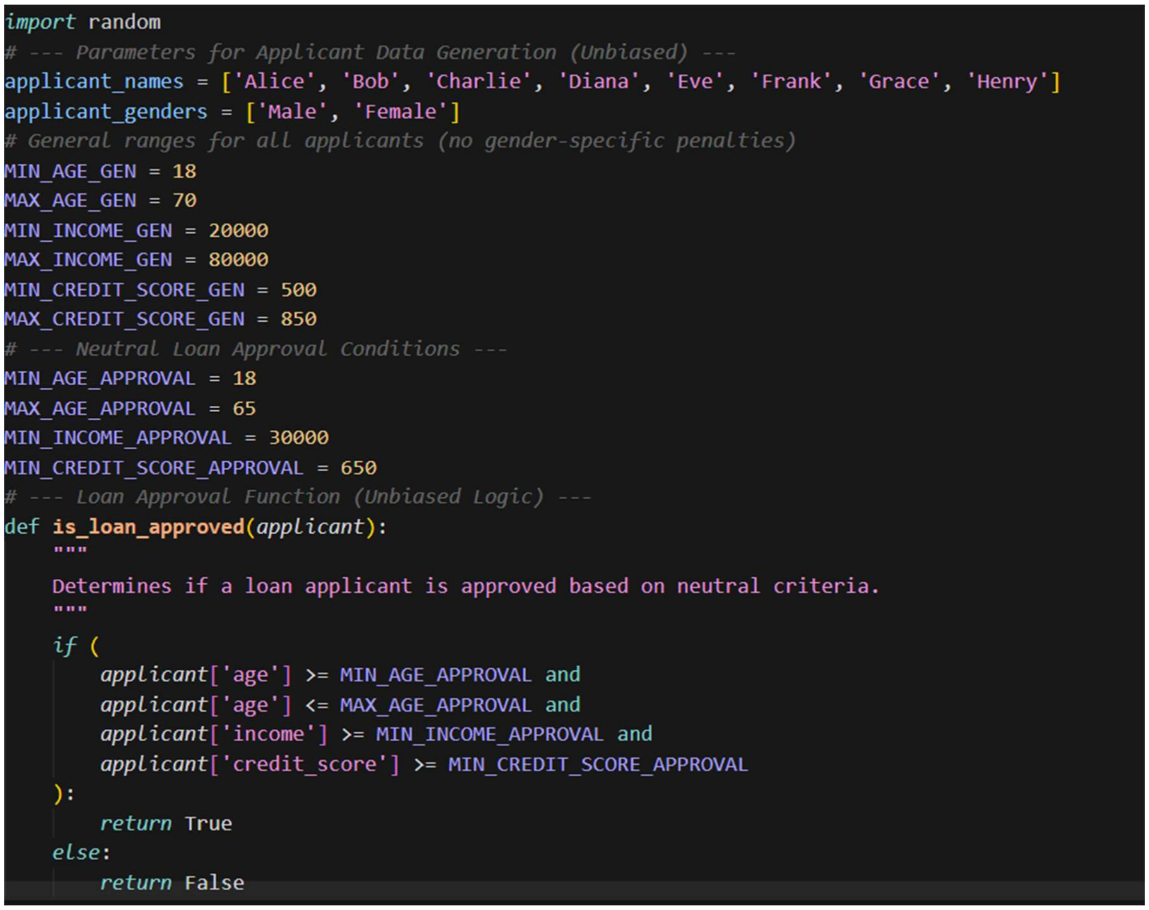
**JUSTIFICATION:**

The biased loan approval code uses gender as a factor for approving loans. Applicants with the same income and credit score receive different decisions, which is unfair. Gender has no connection to a person's ability to repay a loan. This type of logic can lead to discrimination. It shows the risk of blindly trusting AI-generated code.

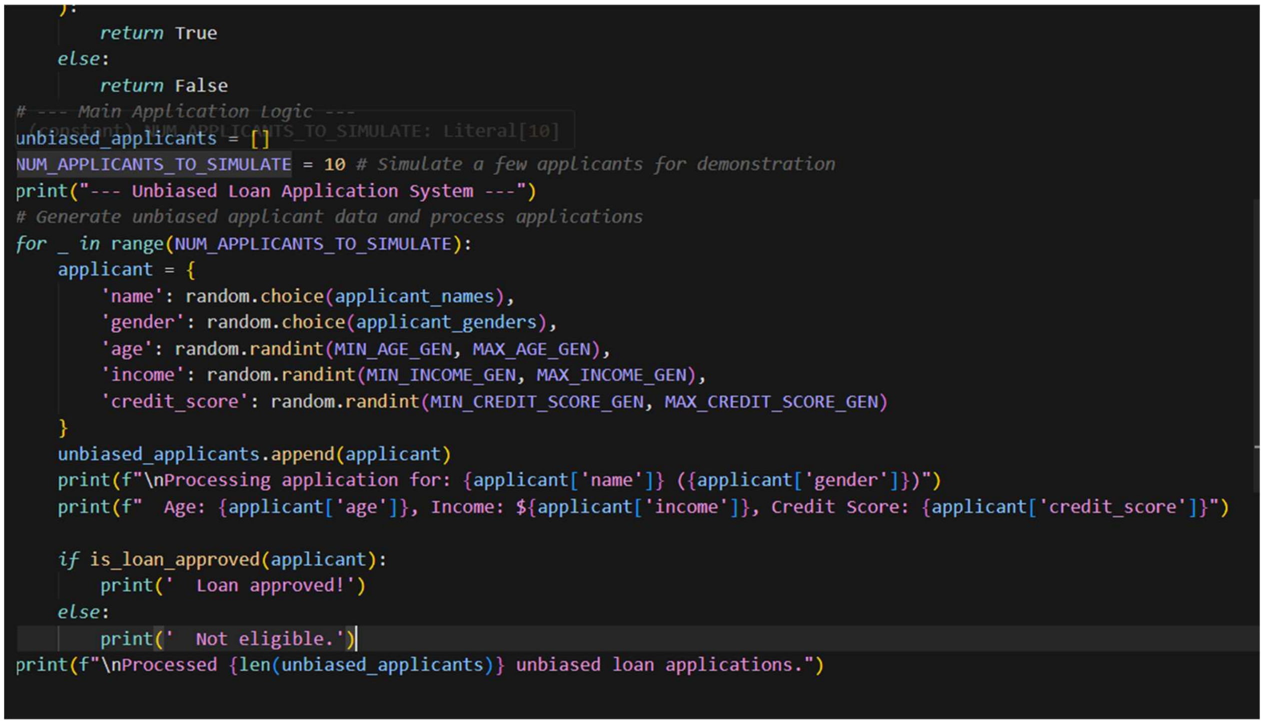
**Revised Python (Bias-Free Version)PROMPT:**

Create a fair Python loan approval system.Loan approval should depend only on income and credit score.Do not use gender or name in decision making.

**CODE:**

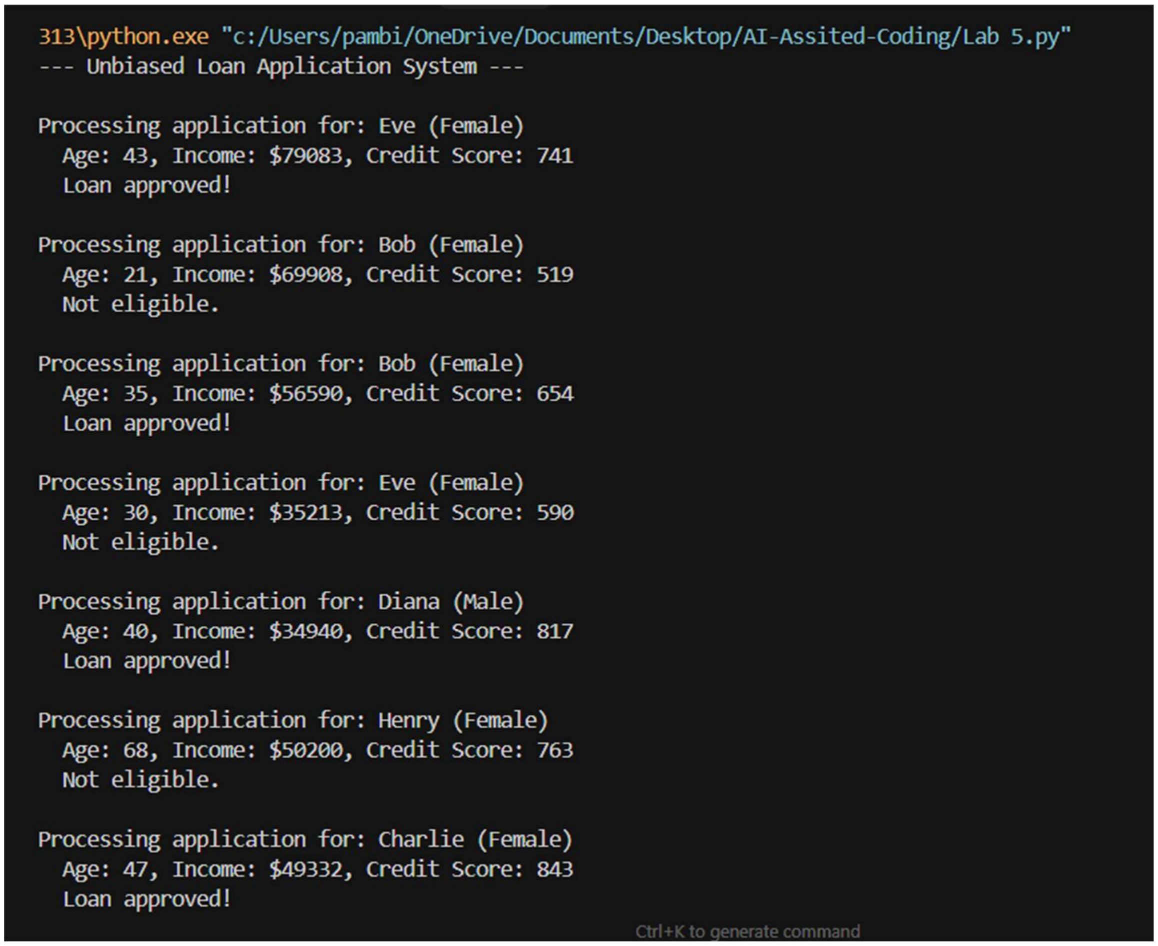


```python
import random
# --- Parameters for Applicant Data Generation (Unbiased) ---
applicant_names = ['Alice', 'Bob', 'Charlie', 'Diana', 'Eve', 'Frank', 'Grace', 'Henry']
applicant_genders = ['Male', 'Female']
# General ranges for all applicants (no gender-specific penalties)
MIN_AGE_GEN = 18
MAX_AGE_GEN = 70
MIN_INCOME_GEN = 20000
MAX_INCOME_GEN = 80000
MIN_CREDIT_SCORE_GEN = 500
MAX_CREDIT_SCORE_GEN = 850
# --- Neutral Loan Approval Conditions ---
MIN_AGE_APPROVAL = 18
MAX_AGE_APPROVAL = 65
MIN_INCOME_APPROVAL = 30000
MIN_CREDIT_SCORE_APPROVAL = 650
# --- Loan Approval Function (Unbiased Logic) ---
def is_loan_approved(applicant):
    """
    Determines if a loan applicant is approved based on neutral criteria.
    """
    if (
        applicant['age'] >= MIN_AGE_APPROVAL and
        applicant['age'] <= MAX_AGE_APPROVAL and
        applicant['income'] >= MIN_INCOME_APPROVAL and
        applicant['credit_score'] >= MIN_CREDIT_SCORE_APPROVAL
    ):
        return True
    else:
        return False
```



```
        return True
    else:
        return False
# --- Main Application Logic ---
unbiased_applicants = []
NUM_APPLICANTS_TO_SIMULATE = 10 # Simulate a few applicants for demonstration
print("--- Unbiased Loan Application System ---")
# Generate unbiased applicant data and process applications
for _ in range(NUM_APPLICANTS_TO_SIMULATE):
    applicant = {
        'name': random.choice(applicant_names),
        'gender': random.choice(applicant_genders),
        'age': random.randint(MIN_AGE_GEN, MAX_AGE_GEN),
        'income': random.randint(MIN_INCOME_GEN, MAX_INCOME_GEN),
        'credit_score': random.randint(MIN_CREDIT_SCORE_GEN, MAX_CREDIT_SCORE_GEN)
    }
    unbiased_applicants.append(applicant)
    print(f"\nProcessing application for: {applicant['name']} ({applicant['gender']})")
    print(f"  Age: {applicant['age']}, Income: ${applicant['income']}, Credit Score: {applicant['credit_score']}")

    if is_loan_approved(applicant):
        print('  Loan approved!')
    else:
        print('  Not eligible.')
print(f"\nProcessed {len(unbiased_applicants)} unbiased loan applications.")
```

**OUTPUT:**



```
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 5.py"
--- Unbiased Loan Application System ---

Processing application for: Eve (Female)
  Age: 43, Income: $79083, Credit Score: 741
  Loan approved!

Processing application for: Bob (Female)
  Age: 21, Income: $69908, Credit Score: 519
  Not eligible.

Processing application for: Bob (Female)
  Age: 35, Income: $56590, Credit Score: 654
  Loan approved!

Processing application for: Eve (Female)
  Age: 30, Income: $35213, Credit Score: 590
  Not eligible.

Processing application for: Diana (Male)
  Age: 40, Income: $34940, Credit Score: 817
  Loan approved!

Processing application for: Henry (Female)
  Age: 68, Income: $50200, Credit Score: 763
  Not eligible.

Processing application for: Charlie (Female)
  Age: 47, Income: $49332, Credit Score: 843
  Loan approved!
```

**JUSTIFICATION:**

The bias-free loan approval code makes decisions only using income and credit score. It does not consider gender or name, so all applicants are treated equally. People with the same financial details get the same result. This approach avoids discrimination. It follows ethical and responsible AI practices.

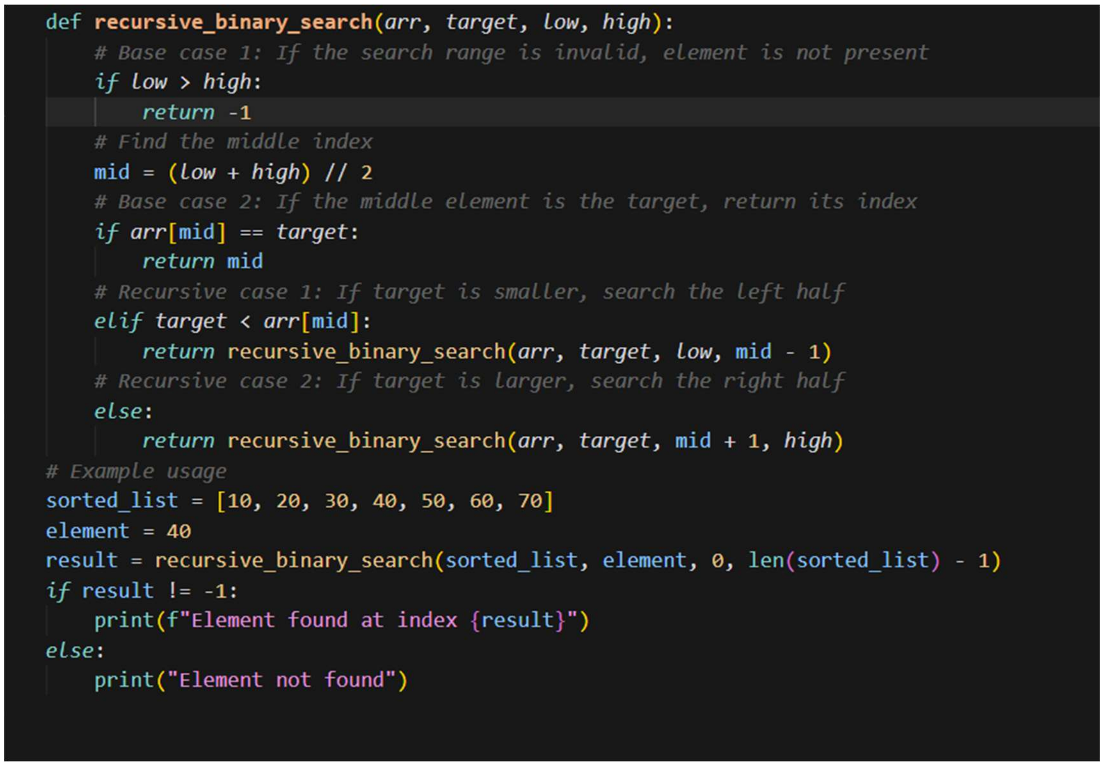
## Question 3:

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

**PROMPT:**

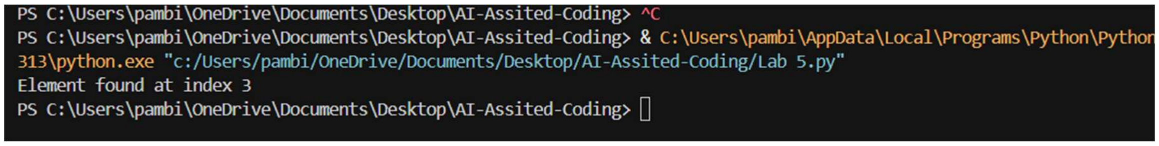
Generate a Python program that uses recursive binary search to find an element in a sorted list.Add clear comments in the code and give a simple explanation of how recursion works, including the base case and recursive case, so that beginners can understand easily.

**CODE:**



```python
def recursive_binary_search(arr, target, low, high):
    # Base case 1: If the search range is invalid, element is not present
    if low > high:
        return -1
    # Find the middle index
    mid = (low + high) // 2
    # Base case 2: If the middle element is the target, return its index
    if arr[mid] == target:
        return mid
    # Recursive case 1: If target is smaller, search the left half
    elif target < arr[mid]:
        return recursive_binary_search(arr, target, low, mid - 1)
    # Recursive case 2: If target is larger, search the right half
    else:
        return recursive_binary_search(arr, target, mid + 1, high)
# Example usage
sorted_list = [10, 20, 30, 40, 50, 60, 70]
element = 40
result = recursive_binary_search(sorted_list, element, 0, len(sorted_list) - 1)
if result != -1:
    print(f"Element found at index {result}")
else:
    print("Element not found")
```

**OUTPUT:**



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> ^C
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Python
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 5.py"
Element found at index 3
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> 
```

**JUSTIFICATION:**

Recursive binary search is efficient because it halves the search space each time. Recursion makes the logic simple and easy to follow. Inline comments, along with clear explanations of the base case and recursive case, help beginners understand how the program works. Overall, it is a clear and correct approach for beginner-level learners.

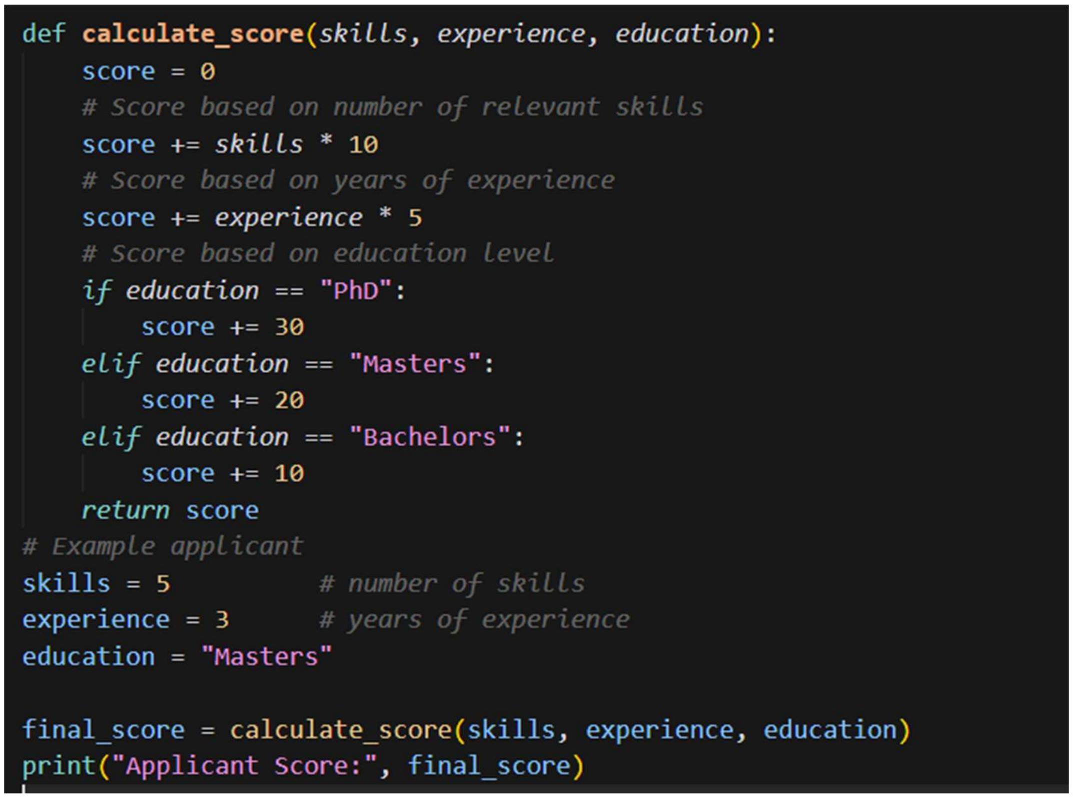
## Question 4:

**Task 4:** Ethical Evaluation of AI-Based Scoring Systems

**PROMPT:**

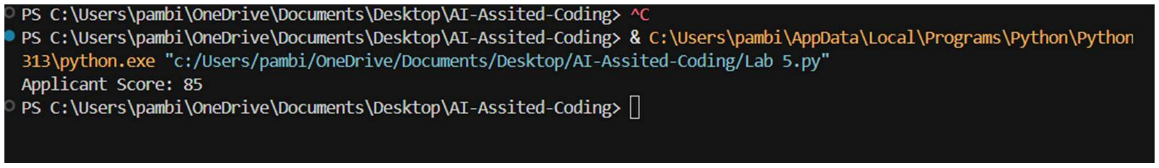
Generate a Python program that scores job applicants based only on skills, experience, and education.Do not use gender, name, age, or any unrelated personal details.Add clear comments explaining the scoring logic.Ensure the system is fair, objective, and easy for beginners to understand.

**CODE:**



```python
def calculate_score(skills, experience, education):
    score = 0
    # Score based on number of relevant skills
    score += skills * 10
    # Score based on years of experience
    score += experience * 5
    # Score based on education level
    if education == "PhD":
        score += 30
    elif education == "Masters":
        score += 20
    elif education == "Bachelors":
        score += 10
    return score
# Example applicant
skills = 5          # number of skills
experience = 3      # years of experience
education = "Masters"

final_score = calculate_score(skills, experience, education)
print("Applicant Score:", final_score)
```

**OUTPUT:**



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> ^C
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Python
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 5.py"
Applicant Score: 85
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> 
```

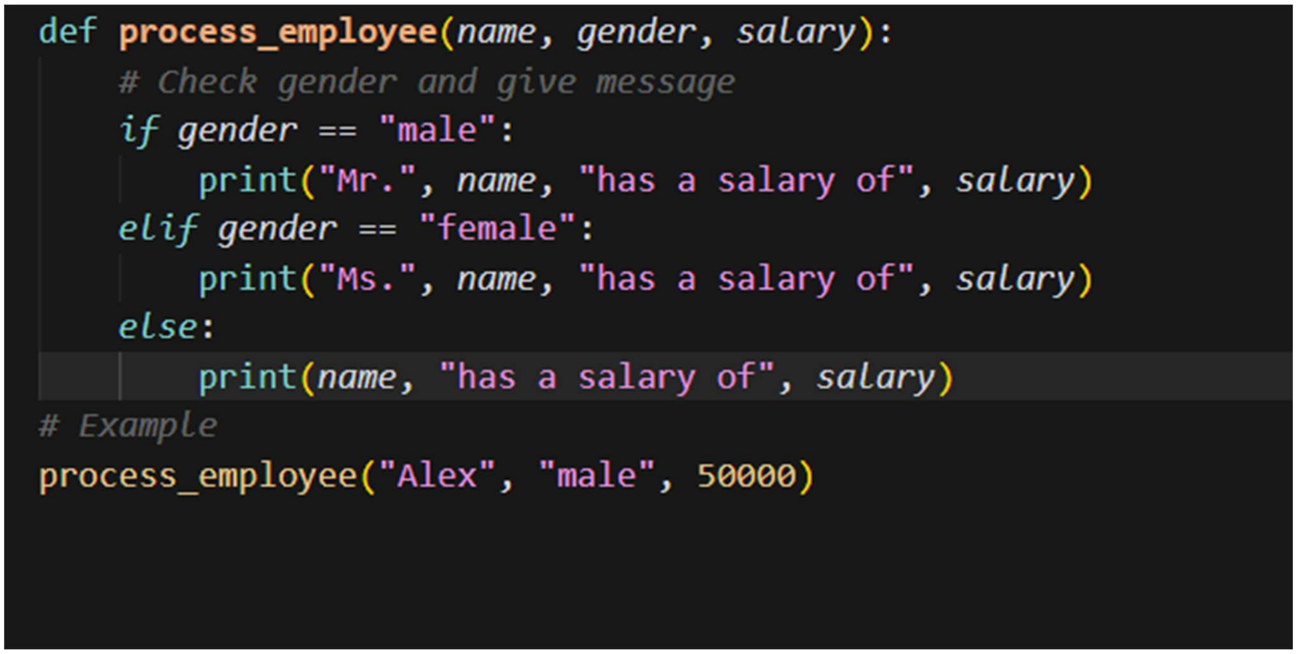
**JUSTIFICATION:**

The scoring system is fair and ethical because it uses only skills, experience, and education. It ignores personal details like gender and name, applies the same rules to everyone, and keeps the logic clear, objective, and transparent.

**QUESTION 5**
**Task 5:** Inclusiveness and Ethical Variable Design

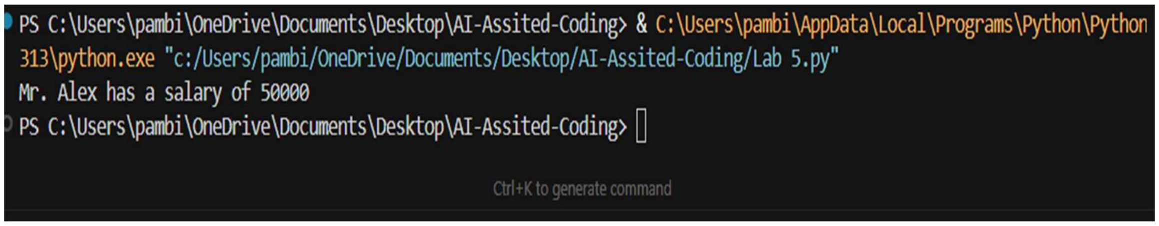
**PROMPT:**
Generate a Python code snippet that processes employee details such as name, gender, and salary.Use conditional logic based on gender and display employee information clearly.

**CODE:**



```python
def process_employee(name, gender, salary):
    # Check gender and give message
    if gender == "male":
        print("Mr.", name, "has a salary of", salary)
    elif gender == "female":
        print("Ms.", name, "has a salary of", salary)
    else:
        print(name, "has a salary of", salary)
# Example
process_employee("Alex", "male", 50000)
```

**OUTPUT:**



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Python
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 5.py"
Mr. Alex has a salary of 50000
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> 
```

## JUSTIFICATION:

The revised code follows ethical practices by removing gender-specific variables and avoiding unfair assumptions. It respects all gender identities, including non-binary and undisclosed users, and treats every employee equally using the same logic. By focusing only on relevant information, the code becomes more fair and objective.