WEEK-ASS-5.3

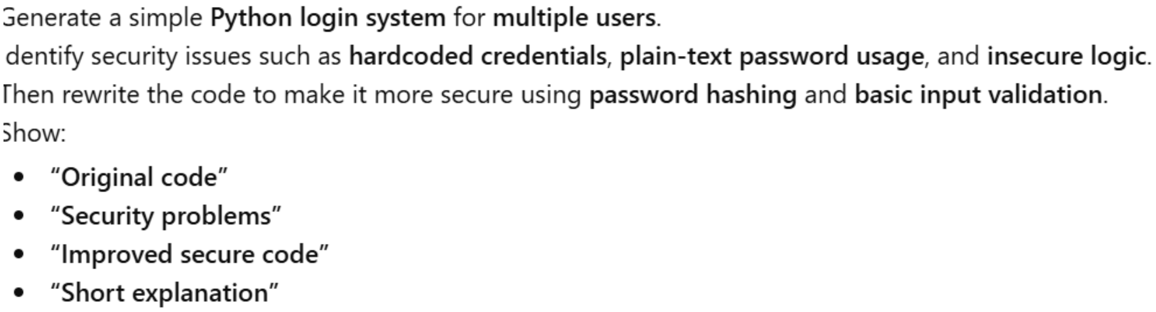
2303A51804

Batch-28

TASK-1:

AI tools can sometimes generate insecure authentication logic.
Task Description
Use an AI tool to generate a simple login system in Python.
Analyse 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).

Prompt:

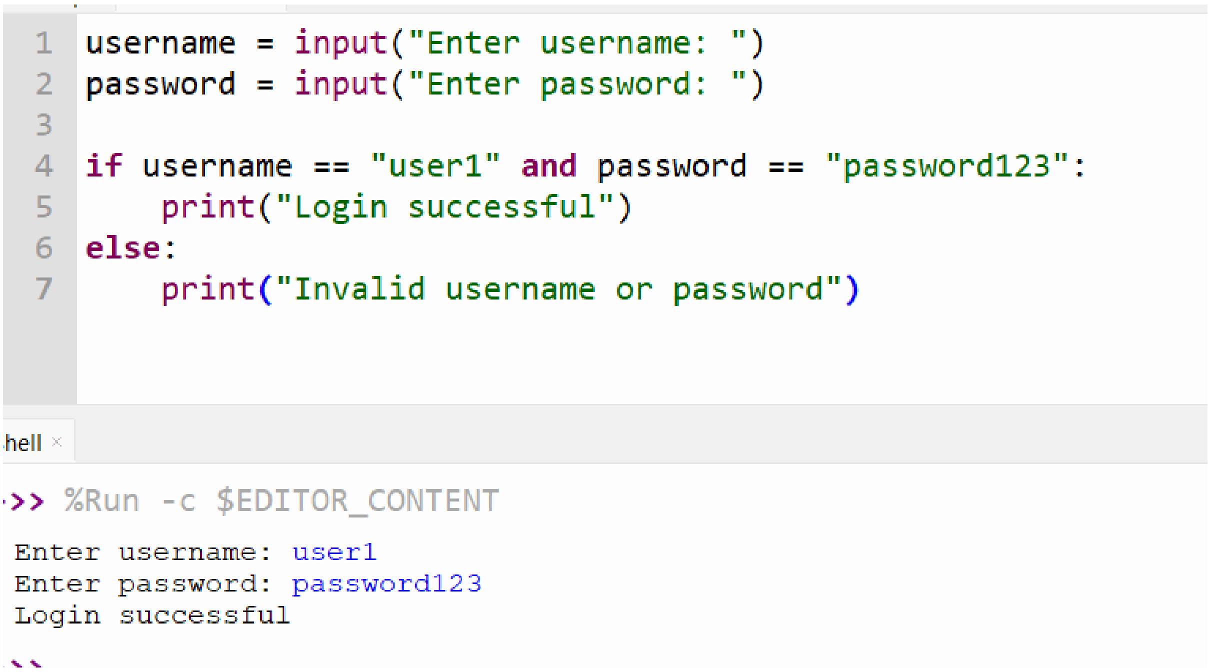


Generate a simple **Python login system** for **multiple users**.
dentify security issues such as **hardcoded credentials**, **plain-text password usage**, and **insecure logic**.
Then rewrite the code to make it more secure using **password hashing** and **basic input validation**.
Show:

- “Original code”
- “Security problems”
- “Improved secure code”
- “Short explanation”

Code and Output:

AI-generated login code



```
username = input("Enter username: ")
password = input("Enter password: ")

if username == "user1" and password == "password123":
    print("Login successful")
else:
    print("Invalid username or password")
```

```
>> %Run -c $EDITOR_CONTENT

 Enter username: user1
 Enter password: password123
 Login successful
```

Identification of security risks

1. Hardcoded Credentials

```
if username == "user1" and password == "admin123":
```

- The username and password are written directly in the source code.
- Anyone who sees the code can know the login credentials.

---

2. Plain Text Password Handling

- The password is entered, stored, and compared in **plain text**.
- If the code or memory is exposed, the password can be easily stolen.

---

3. Single User Only

- The system supports only one fixed user.
- Not scalable and not suitable for real applications.

---

4. No Input Validation

- Empty usernames or passwords are allowed.
- No checks for invalid or unexpected input.

---

5. Insecure Authentication Logic

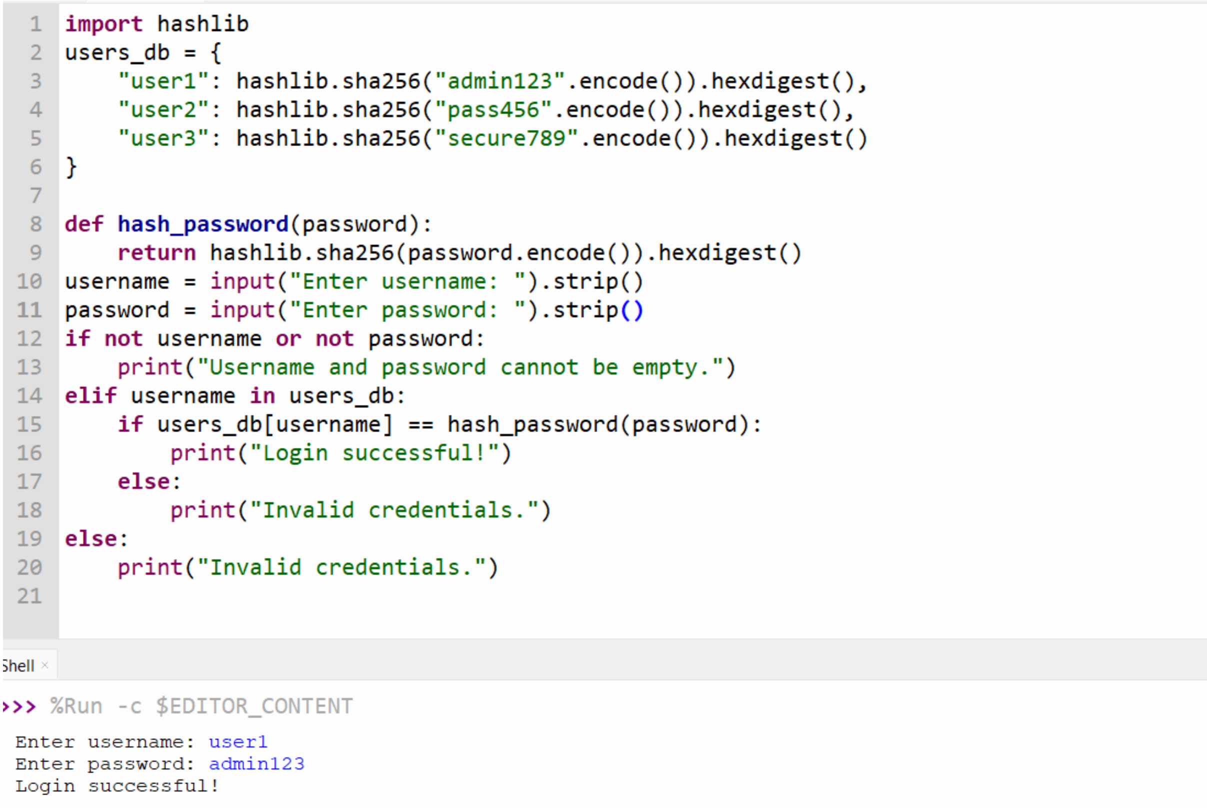
- Direct string comparison for passwords.
- No protection against brute-force attacks.

---

6. No Error Handling or Logging

- The system does not handle unexpected errors.
- No audit or security logging.

- **Revised secure version of the code**



```
import hashlib
users_db = {
    "user1": hashlib.sha256("admin123".encode()).hexdigest(),
    "user2": hashlib.sha256("pass456".encode()).hexdigest(),
    "user3": hashlib.sha256("secure789".encode()).hexdigest()
}

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
username = input("Enter username: ").strip()
password = input("Enter password: ").strip()
if not username or not password:
    print("Username and password cannot be empty.")
elif username in users_db:
    if users_db[username] == hash_password(password):
        print("Login successful!")
    else:
        print("Invalid credentials.")
else:
    print("Invalid credentials.")
```

```
>>> %Run -c $EDITOR_CONTENT

 Enter username: user1
 Enter password: admin123
 Login successful!
```

Explanation:

- Removed hardcoded credentials: Usernames and passwords are no longer directly checked in if statements.
- Password hashing used: Passwords are hashed before storage and comparison, so plain-text passwords are not exposed.
- Multiple user support: The system now works for more than one user.
- Input validation added: Empty usernames or passwords are rejected.
- Safer authentication logic: Generic error messages prevent revealing whether a username or password is incorrect.

Task-2:

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

Analyse whether:

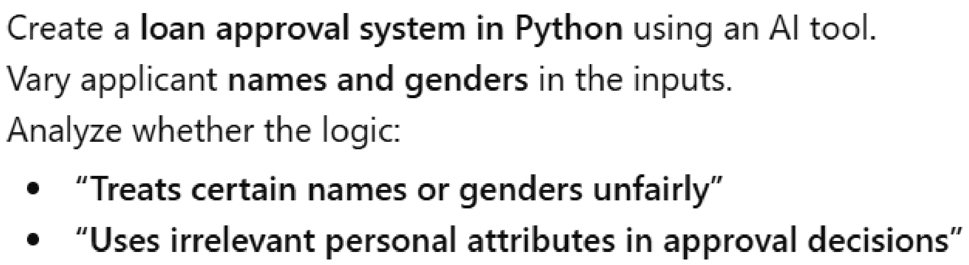
- The logic treats certain genders or names unfairly
- Approval decisions depend on irrelevant personal attributes

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

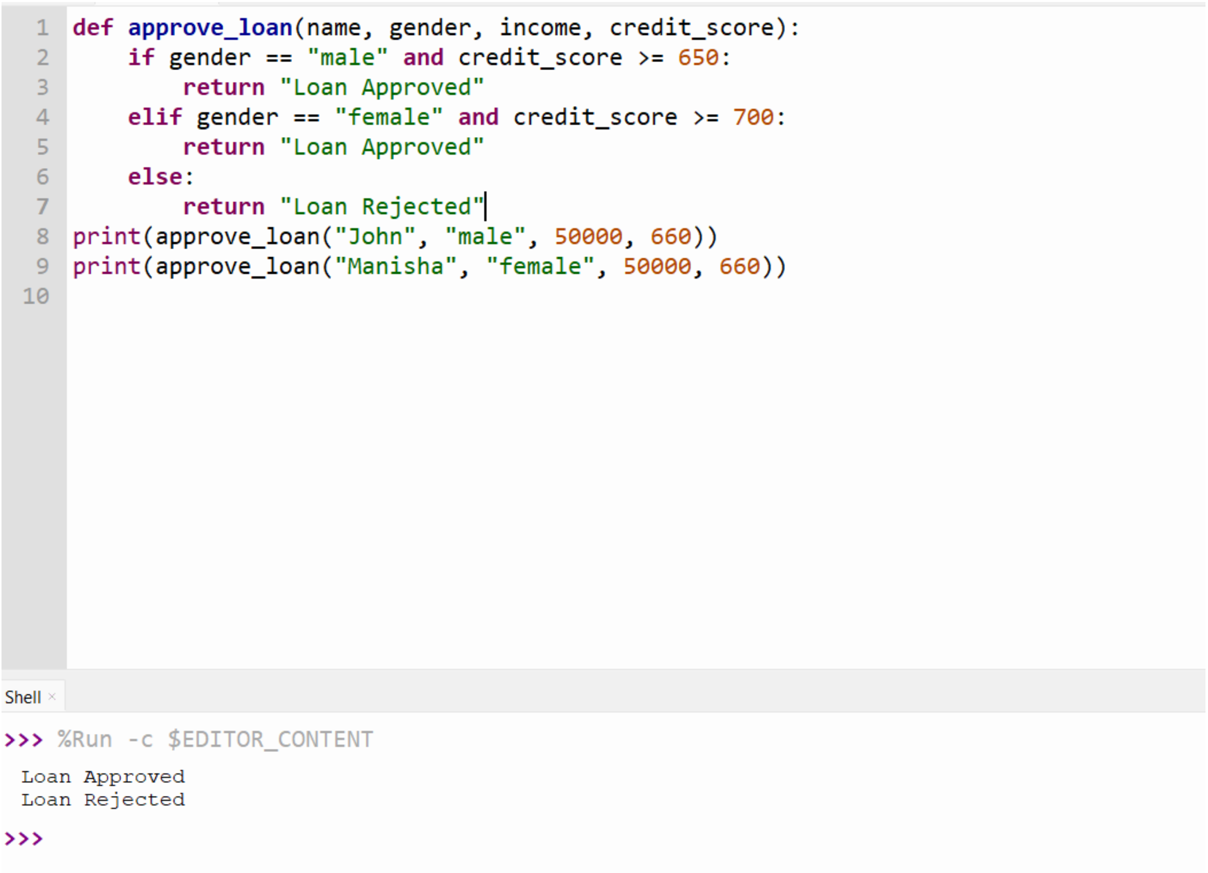
Prompt:



Create a **loan approval system in Python** using an AI tool.
Vary applicant **names and genders** in the inputs.
Analyze whether the logic:

- **“Treats certain names or genders unfairly”**
- **“Uses irrelevant personal attributes in approval decisions”**

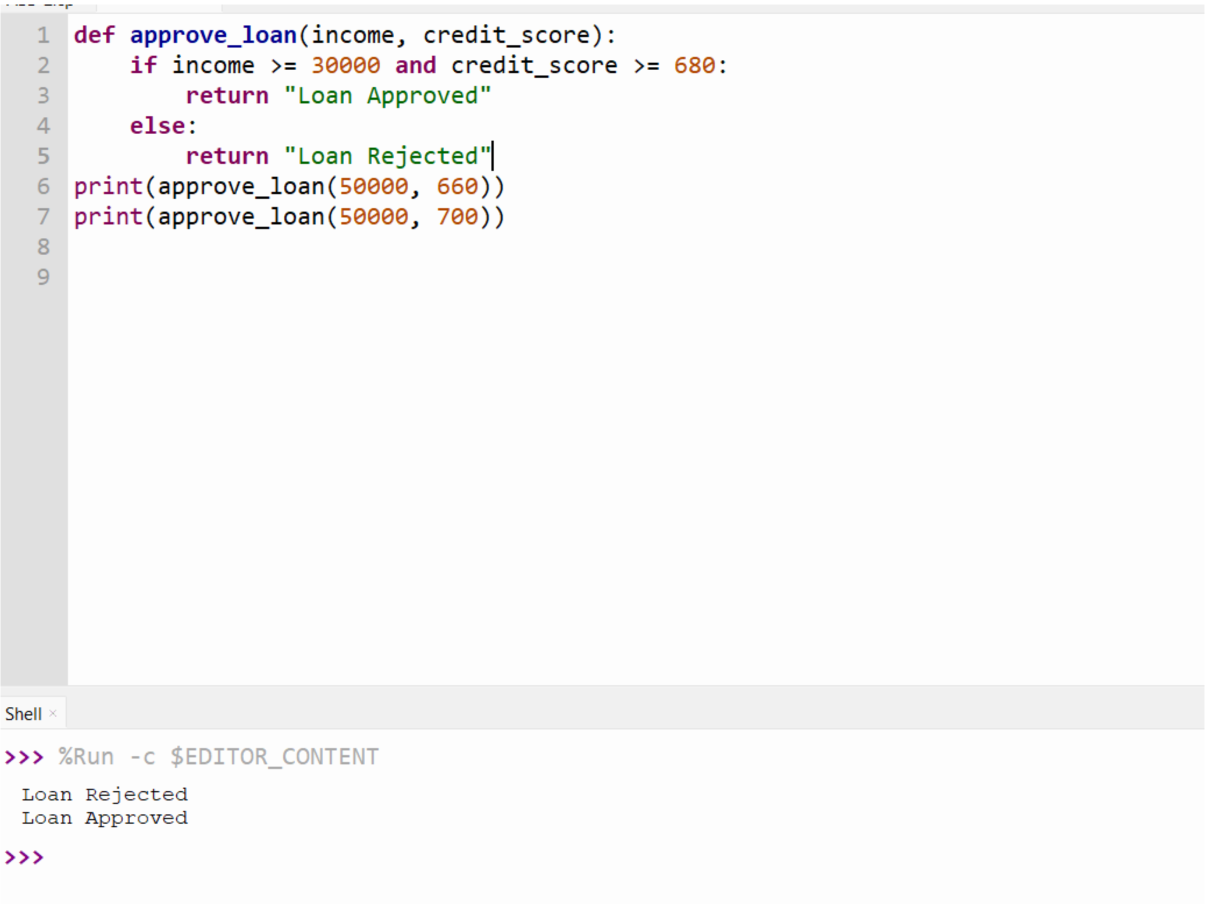
Code and output :



```python
def approve_loan(name, gender, income, credit_score):
    if gender == "male" and credit_score >= 650:
        return "Loan Approved"
    elif gender == "female" and credit_score >= 700:
        return "Loan Approved"
    else:
        return "Loan Rejected"
print(approve_loan("John", "male", 50000, 660))
print(approve_loan("Manisha", "female", 50000, 660))
```

```
>>> %Run -c $EDITOR_CONTENT

 Loan Approved
 Loan Rejected

>>>
```

Revised code:



```python
def approve_loan(income, credit_score):
    if income >= 30000 and credit_score >= 680:
        return "Loan Approved"
    else:
        return "Loan Rejected"
print(approve_loan(50000, 660))
print(approve_loan(50000, 700))
```

```
>>> %Run -c $EDITOR_CONTENT

  Loan Rejected
  Loan Approved

>>>
```

Explanation:

- The AI-generated loan approval system shows **bias** by treating applicants differently based on **gender or name**.
- Personal attributes like **gender and name are irrelevant** to loan repayment ability.
- Applicants with the same income and credit score may receive **different decisions**, which is unfair.
- Such biased logic can **reinforce discrimination** and violate ethical and legal fairness principles.
- To reduce bias, **remove sensitive attributes** (gender, name) from decision-making.
- Use only **objective financial factors** such as income and credit score.
- Perform **regular fairness testing and audits** to ensure equal treatment.

Task-3:

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:

o Clear inline comments

o A step-by-step explanation of the recursive logic

After generating the code, analyse:

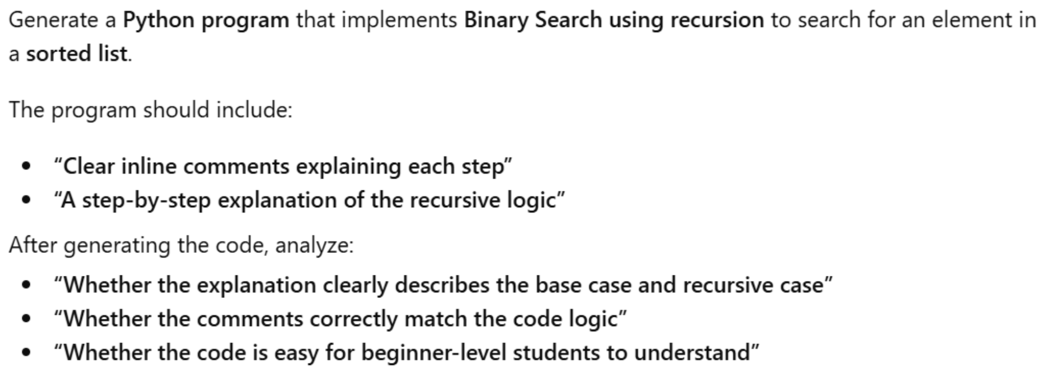
- Whether the explanation clearly describes the base case and recursive case

- 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 explanations
- Student’s assessment on clarity, correctness, and transparency

Prompt:



Generate a **Python program** that implements **Binary Search using recursion** to search for an element in a **sorted list**.

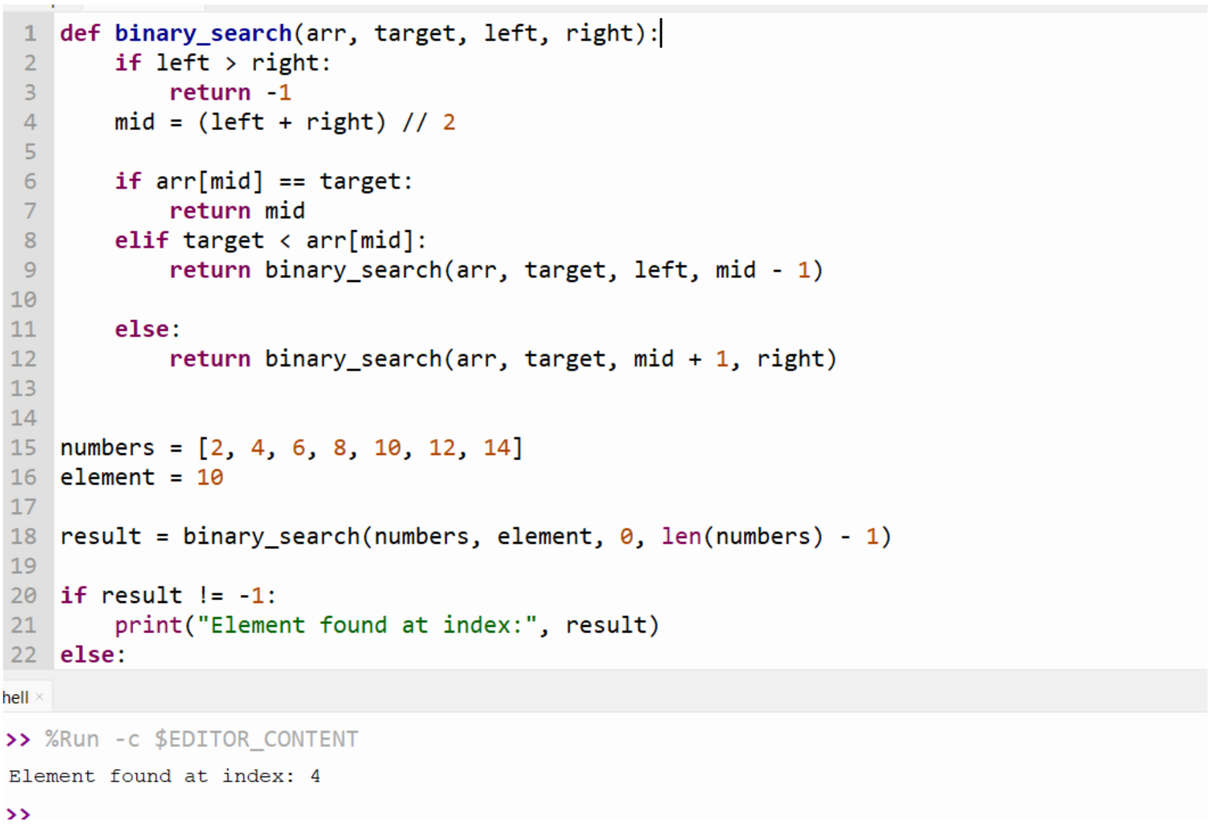
The program should include:

- **“Clear inline comments explaining each step”**
- **“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 easy for beginner-level students to understand”**

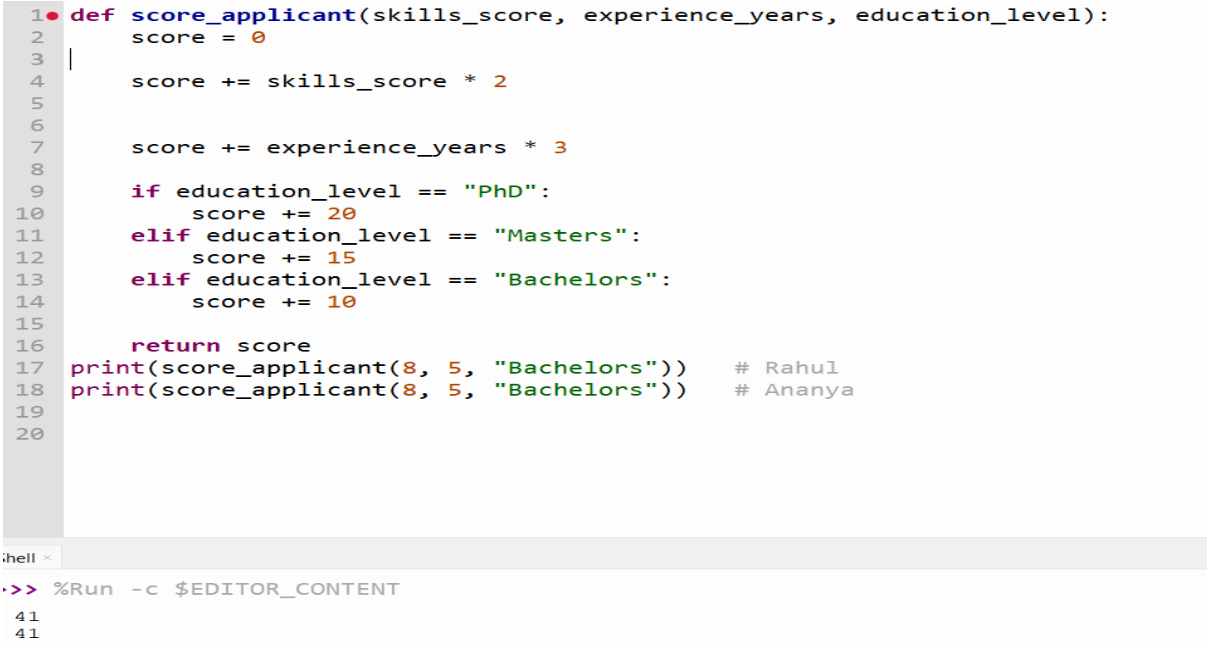
Code and Output:



```
def binary_search(arr, target, left, right):
    if left > right:
        return -1
    mid = (left + right) // 2

    if arr[mid] == target:
        return mid
    elif target < arr[mid]:
        return binary_search(arr, target, left, mid - 1)

    else:
        return binary_search(arr, target, mid + 1, right)


numbers = [2, 4, 6, 8, 10, 12, 14]
element = 10

result = binary_search(numbers, element, 0, len(numbers) - 1)

if result != -1:
    print("Element found at index:", result)
else:
```

```
>> %Run -c $EDITOR_CONTENT
Element found at index: 4
>>
```

Revised code:



```
def score_applicant(skills_score, experience_years, education_level):
    score = 0

    score += skills_score * 2


    score += experience_years * 3

    if education_level == "PhD":
        score += 20
    elif education_level == "Masters":
        score += 15
    elif education_level == "Bachelors":
        score += 10

    return score
print(score_applicant(8, 5, "Bachelors"))   # Rahul
print(score_applicant(8, 5, "Bachelors"))   # Ananya
```

```
>>> %Run -c $EDITOR_CONTENT
 41
 41
```

Explanation:

- Binary Search works on a sorted list by repeatedly dividing the search space into two halves.
- The base case occurs when the left index becomes greater than the right index, meaning the element is not present.
- The middle element is calculated using (left + right) // 2.
- If the middle element matches the target, its index is returned.
- If the target is smaller than the middle element, the function recursively searches the left half.
- If the target is larger, the function recursively searches the right half.
- This process continues until the element is found or the base case is reached.

****Student's assessment on clarity, correctness, and transparency

Clarity

- The code is easy to read and understand.
- Variable names (array, left, right, mid) are clear and meaningful.
- Suitable for beginner-level students.

Correctness

- The base case (left > right) is correctly implemented.
- The recursive cases correctly reduce the search space.
- The algorithm correctly finds the element or reports if it is not found.

Transparency

- Inline comments accurately explain each step of the code.
- The explanation clearly distinguishes between the base case and the recursive case.
- The logic is easy to verify and trace step-by-step.

Task-4:

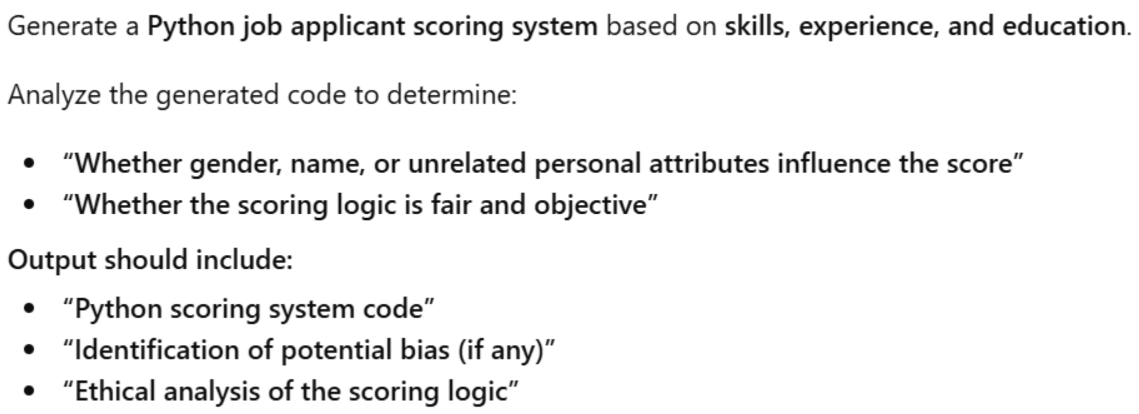
Ethical Evaluation of AI-Based Scoring Systems
Scenario
AI-generated scoring systems can influence hiring decisions.
Task Description
Ask an AI tool to generate a job applicant scoring system based on features
such as:
• Skills
• Experience
• Education
Analyse the generated code to check:
• Whether gender, 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 job applicant scoring system** based on **skills, experience, and education**.

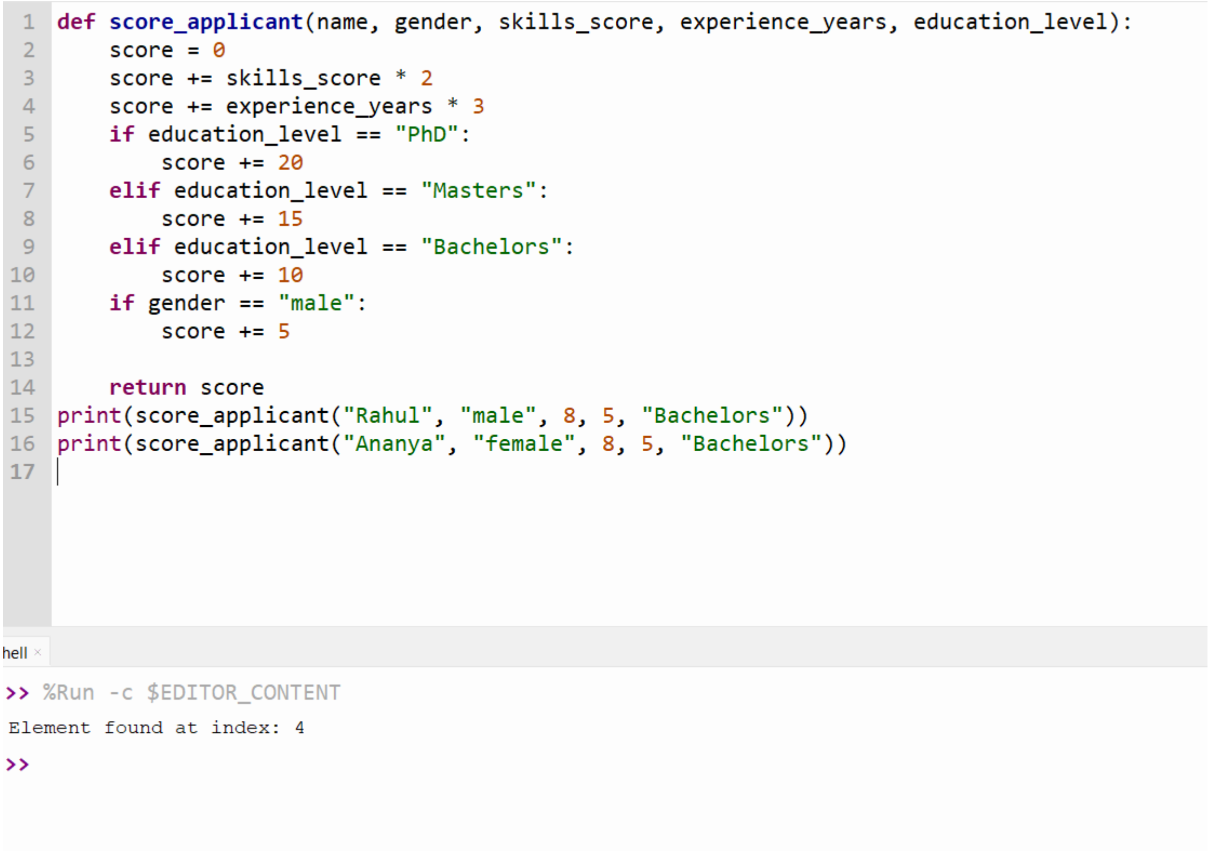
Analyze the generated code to determine:

- **“Whether gender, name, or unrelated personal attributes influence the score”**
- **“Whether the scoring logic is fair and objective”**

Output should include:

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

Code and Output:



```python
def score_applicant(name, gender, skills_score, experience_years, education_level):
    score = 0
    score += skills_score * 2
    score += experience_years * 3
    if education_level == "PhD":
        score += 20
    elif education_level == "Masters":
        score += 15
    elif education_level == "Bachelors":
        score += 10
    if gender == "male":
        score += 5

    return score
print(score_applicant("Rahul", "male", 8, 5, "Bachelors"))
print(score_applicant("Ananya", "female", 8, 5, "Bachelors"))

```

```
>> %Run -c $EDITOR_CONTENT
Element found at index: 4
>>
```

Identification of Potential Bias

- The scoring logic **adds extra points based on gender**, which is unrelated to job performance.
- Applicants with **identical skills, experience, and education** receive different scores.
- Gender and name are **irrelevant attributes** and should not influence hiring decisions.
- This introduces **systematic bias** favouring one group over another.

Ethical Analysis of the Scoring Logic

- The system violates **fairness and equal opportunity principles**.
- Using gender in scoring can reinforce workplace discrimination.
- Such logic may lead to **unethical hiring decisions** and legal risks.
- AI systems used in hiring must be **transparent, objective, and justifiable**.
- Decisions should be based only on **job-relevant criteria**.

Explanation:

- The AI-generated job applicant scoring system may introduce **bias** if it uses personal attributes such as **gender or name** in scoring.
- These attributes are **irrelevant to job performance** and should not influence hiring decisions.
- When applicants with the same skills, experience, and education receive different scores, the system becomes **unfair and discriminatory**.
- Such bias can lead to **ethical concerns**, including unequal opportunity and potential legal violations.
- AI systems used in hiring must be **transparent, objective, and explainable**.
- To ensure fairness, scoring should rely only on **job-relevant factors** like skills, experience, and education.
- Regular audits and bias testing can help detect and reduce unfair outcomes.

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.
Analyse the code to identify:
• Gender-specific variables (e.g., male, female)
• Assumptions based on gender or identity

Prompt:

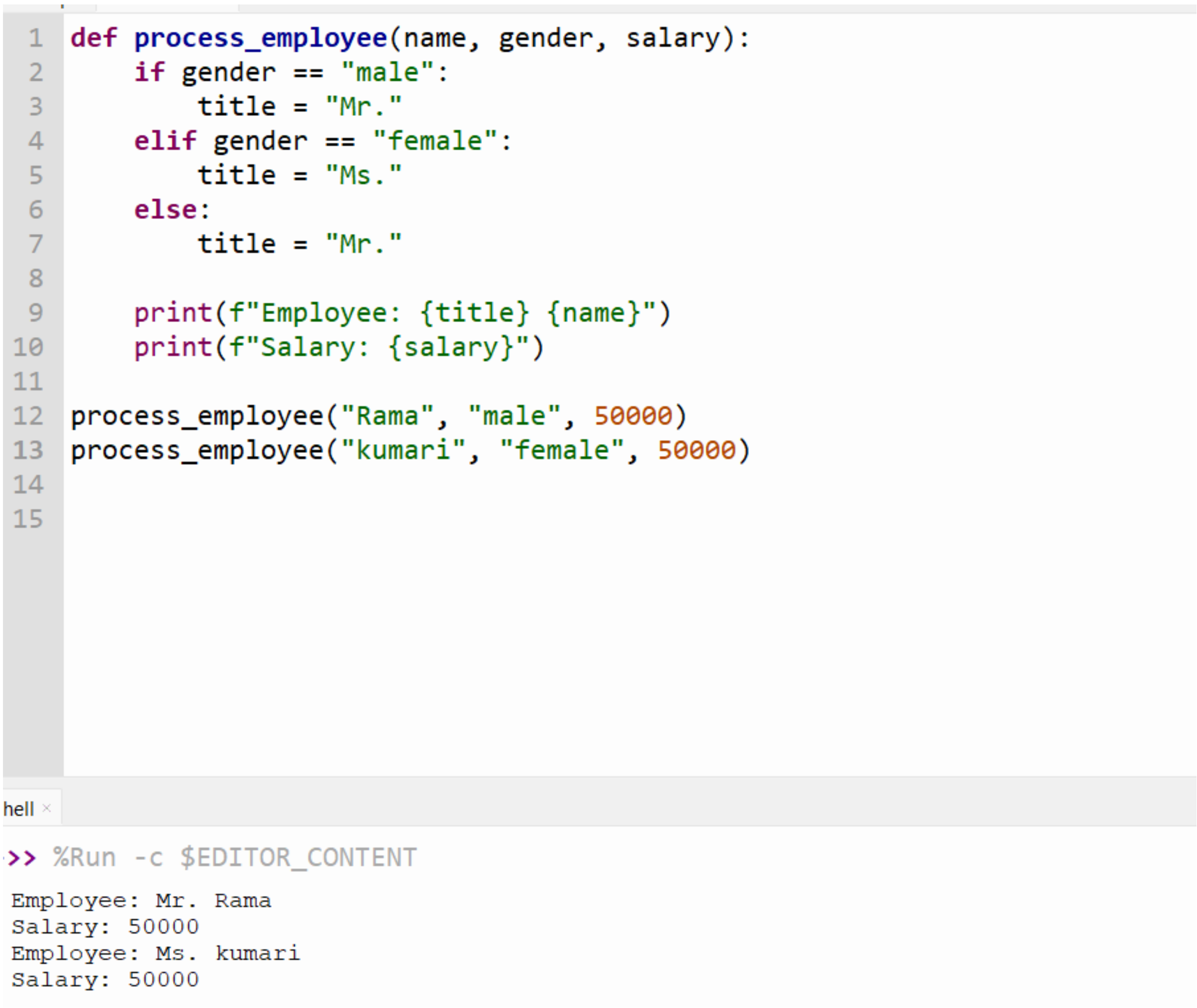
Generate a **Python code snippet** that processes **user or employee details**.

Analyze the generated code to identify:

- **“Gender-specific variables (e.g., male, female)”**
- **“Assumptions based on gender or identity”**

Discuss how these assumptions affect **inclusiveness and ethical software design**, and suggest ways to make the code more inclusive.

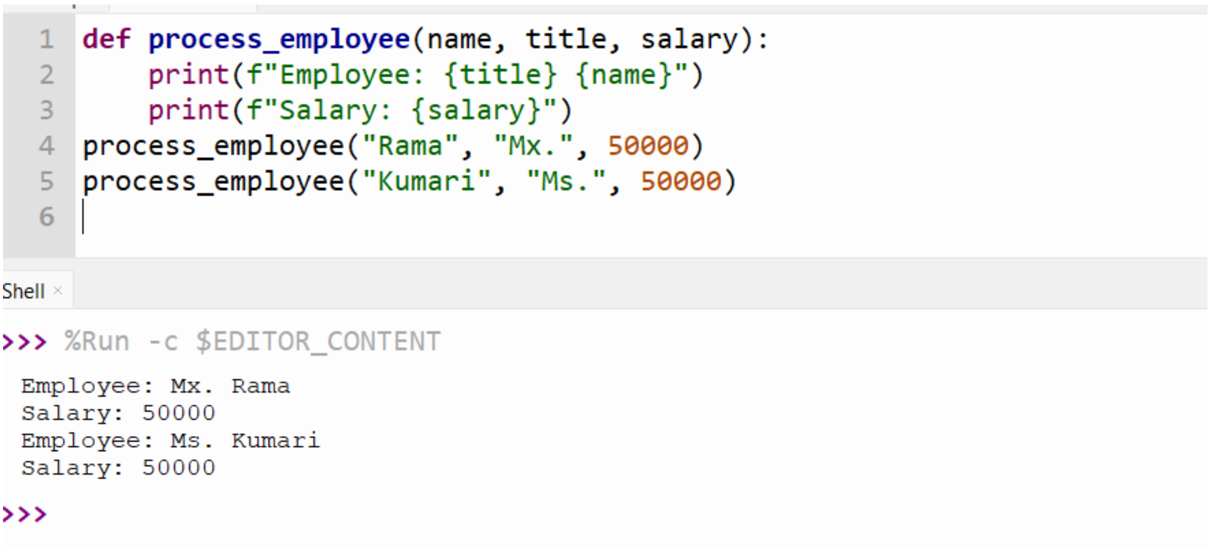
Code and output:



```python
def process_employee(name, gender, salary):
    if gender == "male":
        title = "Mr."
    elif gender == "female":
        title = "Ms."
    else:
        title = "Mr."

    print(f"Employee: {title} {name}")
    print(f"Salary: {salary}")

process_employee("Rama", "male", 50000)
process_employee("kumari", "female", 50000)
```

```
>> %Run -c $EDITOR_CONTENT

Employee: Mr. Rama
Salary: 50000
Employee: Ms. kumari
Salary: 50000
```

Revised code:



```python
def process_employee(name, title, salary):
    print(f"Employee: {title} {name}")
    print(f"Salary: {salary}")
process_employee("Rama", "Mx.", 50000)
process_employee("Kumari", "Ms.", 50000)
```

```
>>> %Run -c $EDITOR_CONTENT
 Employee: Mx. Rama
 Salary: 50000
 Employee: Ms. Kumari
 Salary: 50000
>>>
```

Explanation:

- The code uses a gender-specific variable with only binary values: *male* and *female*.
- Titles such as Mr. and Ms. are assigned based on gender, making the code gender-dependent.
- The code assumes all users fit into binary gender categories, excluding non-binary identities.
- A default masculine title is used, which reinforces gender bias.
- These assumptions ignore personal identity and user preference.
- Such design reduces inclusiveness and can lead to discrimination or discomfort.
- Ethical software design should avoid unnecessary gender-based logic and promote fairness and respect.