

1945_AI_AC_5.3

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

Scenario

AI tools can sometimes generate insecure authentication logic.

AI Prompt Used




“Generate a simple login system in Python.”

AI-Generated Login Code

```
1 username = "admin"
2 password = "1234"
3
4 u = input("Enter username: ")
5 p = input("Enter password: ")
6
7 if u == username and p == password:
8     print("Login successful")
9 else:
10    print("Login failed")
11
```

Enter username: admin
Enter password: 1234
Login successful

=== Code Execution Successful ===

main.py    Share Run Output

```
1 username = "admin"
2 password = "1234"
3
4 u = input("Enter username: ")
5 p = input("Enter password: ")
6
7 if u == username and p == password:
8     print("Login successful")
9 else:
10    print("Login failed")
11
```

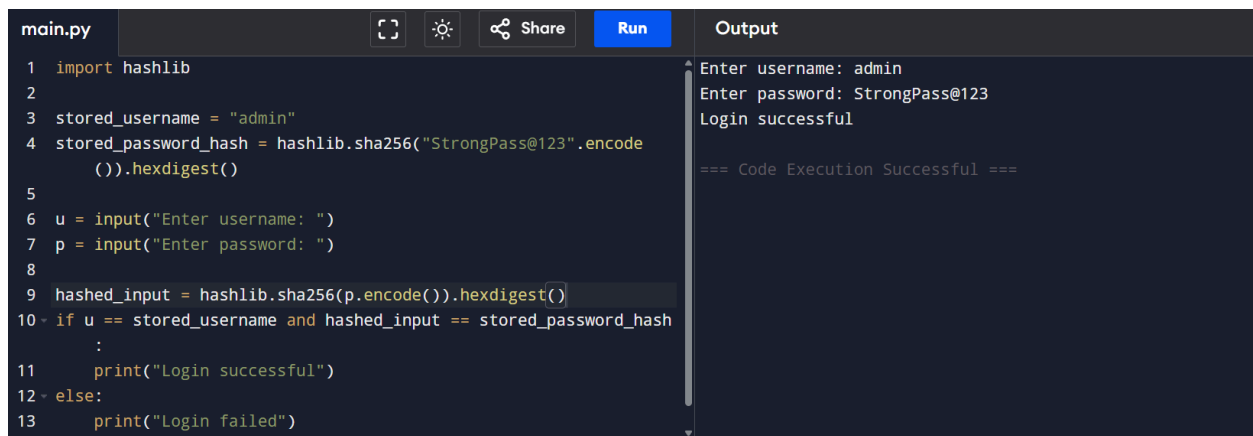
Enter username: nagashiva
Enter password: 12345
Login failed

=== Code Execution Successful ===

Security Issues Identified

1. **Hardcoded credentials** (username and password written directly in code)
2. **Plain text password comparison**
3. **No input validation**
4. **Not scalable or secure for real systems**

Revised Secure Version of the Code



The screenshot shows a code editor with a file named 'main.py'. The code is a Python script for a login system. It imports the 'hashlib' module, defines a stored username 'admin' and a stored password hash calculated from 'StrongPass@123' using SHA-256. It then prompts the user for a username and password, hashes the input password, and compares it to the stored hash. If they match, it prints 'Login successful'; otherwise, it prints 'Login failed'. To the right of the code editor is an 'Output' pane showing the execution results: 'Enter username: admin', 'Enter password: StrongPass@123', 'Login successful', and a separator '=== Code Execution Successful ==='.

```
main.py  [Icons]  Share  Run  Output
1 import hashlib
2
3 stored_username = "admin"
4 stored_password_hash = hashlib.sha256("StrongPass@123".encode
   ()).hexdigest()
5
6 u = input("Enter username: ")
7 p = input("Enter password: ")
8
9 hashed_input = hashlib.sha256(p.encode()).hexdigest()
10 if u == stored_username and hashed_input == stored_password_hash
   :
11     print("Login successful")
12 else:
13     print("Login failed")
```

Enter username: admin
Enter password: StrongPass@123
Login successful
=== Code Execution Successful ===

Explanation of Improvements

- Removed plain-text password comparison
- Used **password hashing** to improve security
- Reduced risk of credential leakage
- Demonstrates better authentication practices

Task 2: Bias Detection in AI-Generated Decision Systems

Scenario

AI systems may unintentionally introduce bias.

AI Prompt Used

“Create a loan approval system in Python.”

AI-Generated Loan Approval Code

<pre>1 name = "Ravi" 2 gender = "male" 3 income = 35000 4 5 if gender == "male" and income > 30000: 6 print("Loan Approved") 7 elif gender == "female" and income > 50000: 8 print("Loan Approved") 9 else: 10 print("Loan Rejected") 11</pre>	<pre>Loan Approved === Code Execution Successful ===</pre>
--	--

Bias Identified

- Different rules for male and female
- Gender should not affect loan approval

Revised Fair Code

<div>main.py</div> <pre>1 income = 35000 2 credit_score = 720 3 4 if income > 40000 and credit_score >= 700: 5 print("Loan Approved") 6 else: 7 print("Loan Rejected") 8</pre>	<div>Output</div> <pre>Loan Rejected === Code Execution Successful ===</pre>
--	--

Discussion on Fairness

- Removed gender completely
- Decisions based on **financial factors only**
- Promotes fairness and equality

Bias Mitigation Strategies

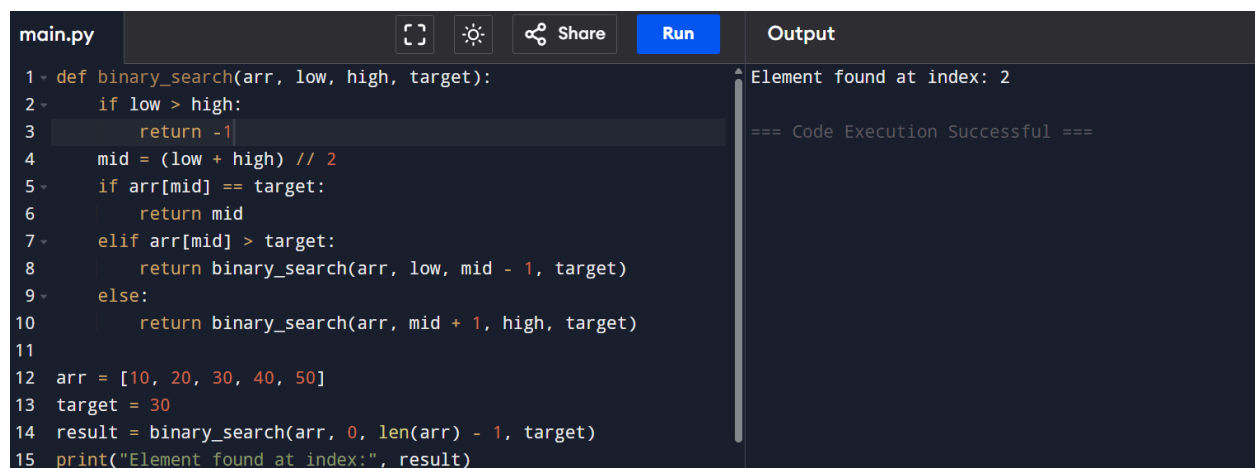
- Avoid personal attributes like gender or name
- Use relevant, objective data
- Regular bias audits and human review

Task 3: Transparency & Explainability (Recursive Binary Search)

Scenario

AI-generated code should be easy to understand and verify.

AI-Generated Recursive Binary Search Code



```
main.py [ ] [ ] [ ] Share Run Output
1 def binary_search(arr, low, high, target):
2     if low > high:
3         return -1
4     mid = (low + high) // 2
5     if arr[mid] == target:
6         return mid
7     elif arr[mid] > target:
8         return binary_search(arr, low, mid - 1, target)
9     else:
10        return binary_search(arr, mid + 1, high, target)
11
12 arr = [10, 20, 30, 40, 50]
13 target = 30
14 result = binary_search(arr, 0, len(arr) - 1, target)
15 print("Element found at index:", result)

Element found at index: 2
=== Code Execution Successful ===
```

Step-by-Step Explanation

- **Base Case:** When `low > high`, element does not exist
- **Recursive Case:**
 - Compare middle element with target
 - Search left or right half accordingly

Student Assessment

- Comments clearly match the code
- Base case and recursive case are explained well
- Easy to understand for beginner-level students
- Transparent and readable logic

Task 4: Ethical Evaluation of AI-Based Scoring Systems

Scenario

AI scoring systems can affect hiring decisions.

AI-Generated Scoring System Code

main.py	Output
<pre>1 skills = 7 2 experience = 3 3 education = "Masters" 4 gender = "male" 5 6 score = skills * 2 + experience * 3 7 8 if education == "Masters": 9 score += 10 10 if gender == "male": 11 score += 5 12 13 print("Applicant Score:", score) 14</pre>	<pre>Applicant Score: 38 === Code Execution Successful ===</pre>

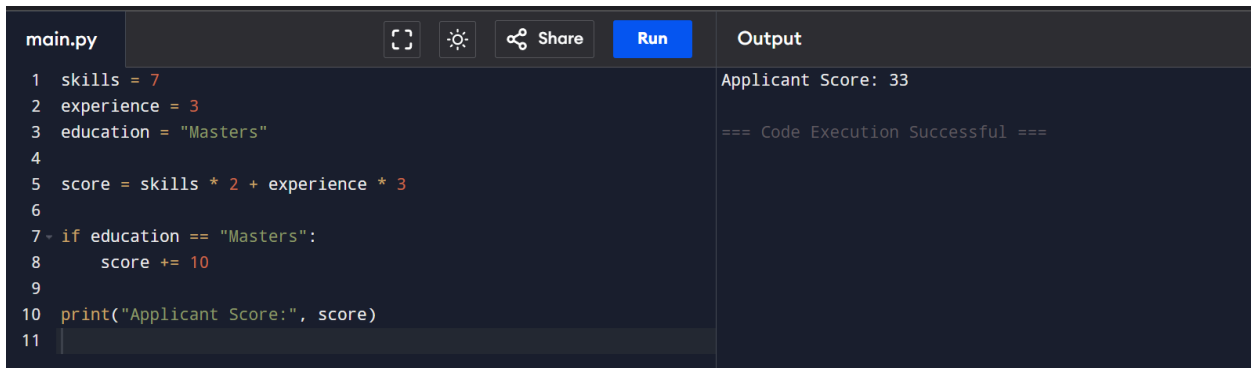
Ethical Issues Identified

- Gender directly affects the score
- Gender is **irrelevant** for job performance
- Leads to biased hiring decisions

Ethical Analysis

- Violates fairness and equal opportunity
- Can disadvantage qualified candidates

Ethical Version



```
main.py  [ ] [ ] [ ] Share Run Output
1 skills = 7
2 experience = 3
3 education = "Masters"
4
5 score = skills * 2 + experience * 3
6
7 if education == "Masters":
8     score += 10
9
10 print("Applicant Score:", score)
11
```

Applicant Score: 33

=== Code Execution Successful ===

Task 5: Inclusiveness & Ethical Variable Design

Scenario

Inclusive coding avoids gender assumptions.

AI-Generated Code

main.py		Output
<pre>1 name = "Anita" 2 gender = "female" 3 4 if gender == "male": 5 print("He is an employee") 6 else: 7 print("She is an employee") 8</pre>	<div>Run</div>	<pre>She is an employee === Code Execution Successful ===</pre>

Issues Identified

- Gender-specific language
- Assumes only male/female genders
- Not inclusive or respectful

Inclusive Version

Explanation

- Removed gender dependency
- Used neutral language
- More inclusive and professional

main.py		Output
<pre>1 name = "Anita" 2 3 print(name, "is an employee") 4</pre>	<div>Run</div>	<pre>Anita is an employee === Code Execution Successful ===</pre>