

1934_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")
```

Enter username: admin
Enter password: 1234
Login successful
==== Code Execution Successful ===

```
main.py
```

```
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")
```

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



```
main.py
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")
```

Output

```
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

```
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")
```

Loan Approved
==== Code Execution Successful ===

Bias Identified

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

Revised Fair Code

```
main.py
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")
```

Loan Rejected
==== Code Execution Successful ===

Discussion on Fairness

- Removed gender completely

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

Bias Mitigation Strategies

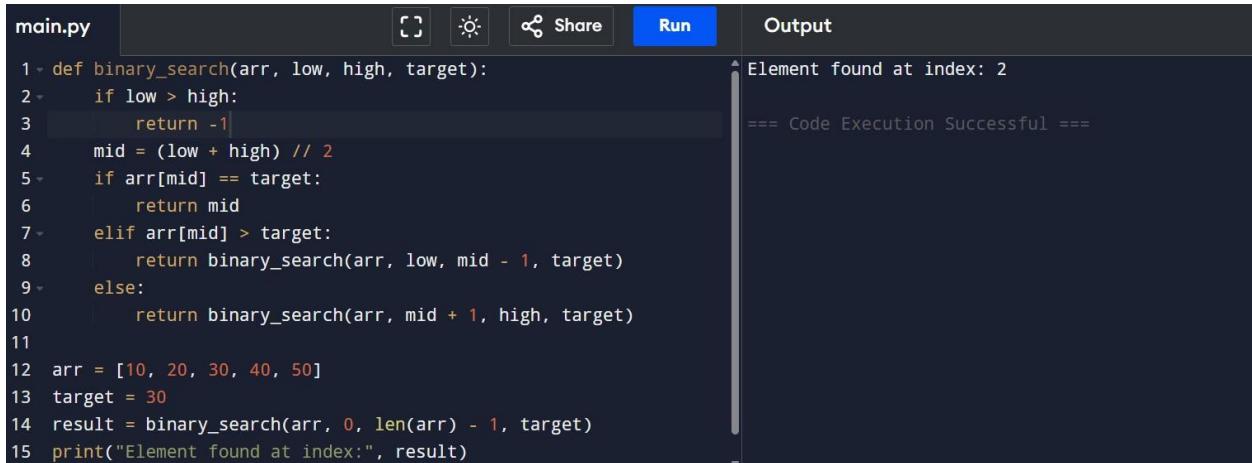
- Avoid personal attributes like gender or name
- Use relevant, objective data
- Regular bias audits and human review

Task 3: Transparency & Explainability (Recursive Binary Search)

Scenario

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

AI-Generated Recursive Binary Search Code



The screenshot shows a Jupyter Notebook interface with the following details:

- Code Cell:** The code is named `main.py`. It contains a recursive function `binary_search` that takes an array `arr`, a low index `low`, a high index `high`, and a target value `target`. The function returns the index of the target if found, or -1 if it's not in the array. It uses integer division (`//`) to find the middle element.
- Execution:** The code is run, and the output is displayed in the "Output" pane.
- Output:** The output shows the message "Element found at index: 2" followed by "==== Code Execution Successful ===".

```

main.py
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)

```

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

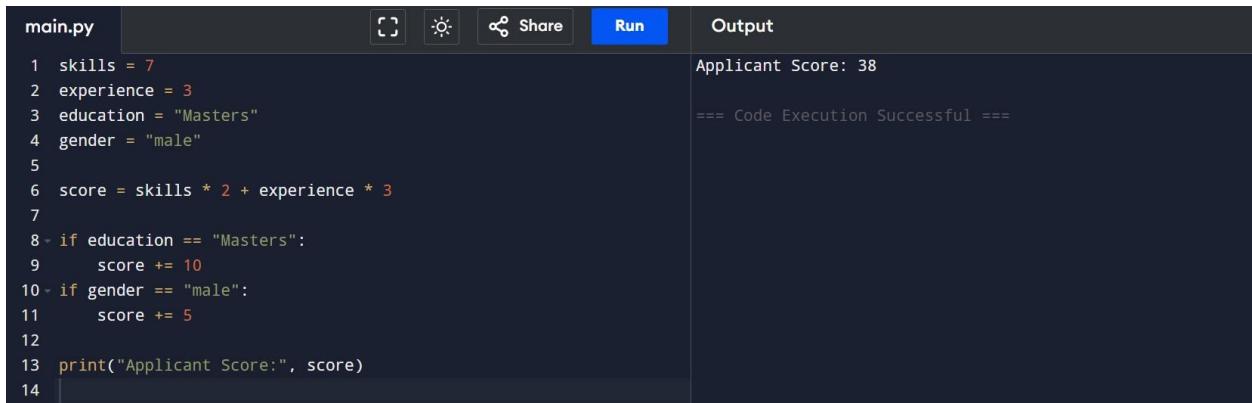
- Comments clearly match the code
- Base case and recursive case are explained well
- Easy to understand for beginner-level students
- Transparent and readable logic

Task 4: Ethical Evaluation of AI-Based Scoring Systems

Scenario

AI scoring systems can affect hiring decisions.

AI-Generated Scoring System Code



The screenshot shows a code editor interface with a dark theme. On the left, a file named 'main.py' contains the following Python code:

```
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
```

On the right, there is an 'Output' panel showing the results of running the code. The output is:

```
Applicant Score: 38
== Code Execution Successful ==
```

Ethical Issues Identified

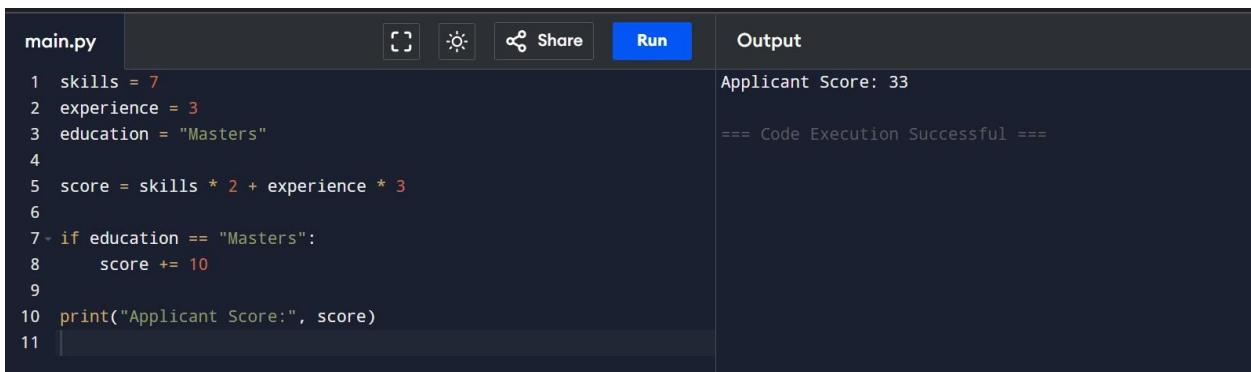
- Gender directly affects the score

- Gender is **irrelevant** for job performance
- Leads to biased hiring decisions

Ethical Analysis

- Violates fairness and equal opportunity
- Can disadvantage qualified candidates

Ethical Version



The screenshot shows a Jupyter Notebook interface with a code cell titled "main.py" containing Python code. The code calculates a score based on skills and experience, with an additional 10 points if the education level is "Masters". The output cell shows the result "Applicant Score: 33" and a message "Code Execution Successful".

```
main.py
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 
```

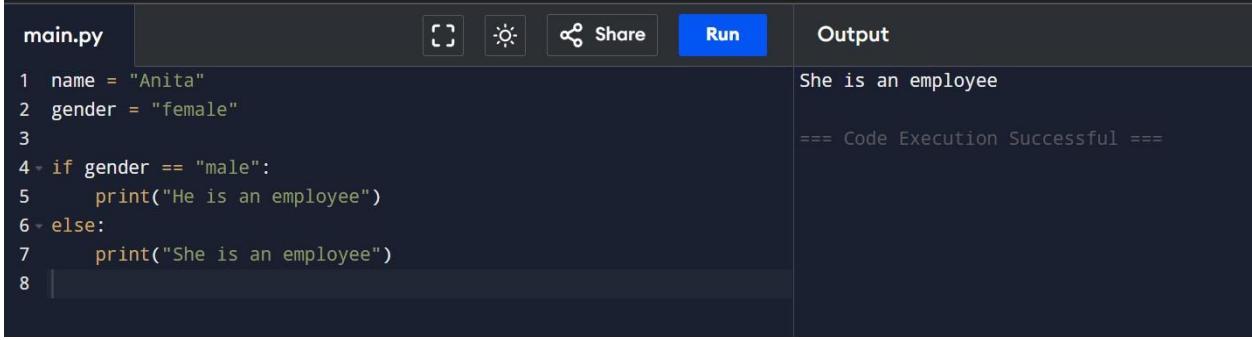
Applicant Score: 33
== Code Execution Successful ==

Task 5: Inclusiveness & Ethical Variable Design

Scenario

Inclusive coding avoids gender assumptions.

AI-Generated Code



The screenshot shows a code editor interface with a dark theme. On the left, a file named "main.py" contains the following Python code:

```
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
```

On the right, the "Output" panel displays the results of running the code:

```
She is an employee
== Code Execution Successful ==
```

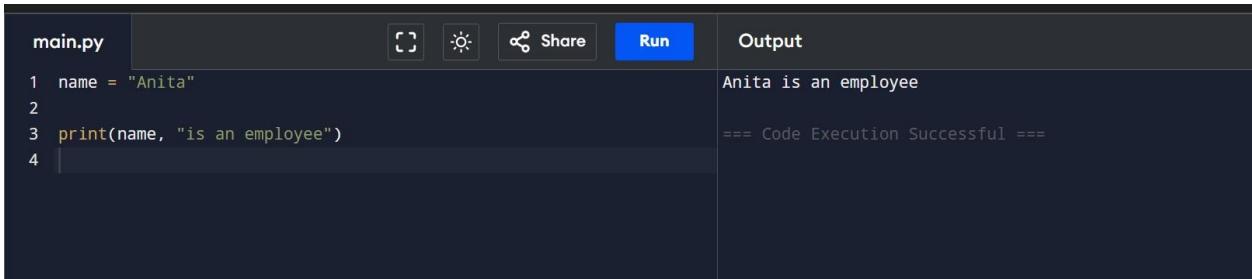
Issues Identified

- 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



The screenshot shows a code editor interface with a dark theme. On the left, a file named "main.py" contains the following Python code:

```
1 name = "Anita"
2
3 print(name, "is an employee")
4
```

On the right, the "Output" panel displays the results of running the code:

```
Anita is an employee
== Code Execution Successful ==
```