

AI ASSISTED CODING

ASSIGNMENT - 5.3

ROLL NO. 2303A510J0

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

Simple login system:

Prompt:

Generate a simple Python login system that asks for a username and password and checks whether the user can log in

Code:

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

if username == "admin" and password == "admin123":
    print("Login successful")
else:
    print("Invalid username or password")
```

Output:

```
PS C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces> &
C:/Users/rohit/AppData/Local/Programs/Python/Python314/python.exe
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/r_800/p
2/1742b.py
Enter username: admin
Enter password: admin123
Login successful
PS C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces> &
C:/Users/rohit/AppData/Local/Programs/Python/Python314/python.exe
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/r_800/p
2/1742b.py
Enter username: admi
Enter password: 123admin
Invalid username or password
```

Security risks:

- Hardcoded credentials
- Plain-Text Password Handling
- No User Registration Mechanism
- Poor Scalability and security

Revised one:

Prompt:

Improve the login system by adding user registration and store passwords securely instead of hardcoding them.

Code:

```
import hashlib
```

```
# Simple in-memory user storage
users = {}

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

def register():
    username = input("Create username: ").strip()
    password = input("Create password: ").strip()

    if username in users:
        print("User already exists")
        return

    users[username] = hash_password(password)
    print("Registration successful")

def login():
    username = input("Enter username: ").strip()
    password = input("Enter password: ").strip()

    if username not in users:
        print("No such user. Please register first.")
        return

    if users[username] == hash_password(password):
        print("Login successful")
    else:
        print("Invalid password")

while True:
    print("\n1. Register\n2. Login\n3. Exit")
    choice = input("Choose an option: ")

    if choice == "1":
        register()
    elif choice == "2":
        login()
```

```
elif choice == "3":  
    break  
else:  
    print("Invalid choice")
```

Output:

```
PS C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces> &  
C:/Users/rohit/AppData/Local/Programs/Python/Python314/python.exe  
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/r_800/p  
2/1742b.py
```

1. Register
2. Login
3. Exit

Choose an option: 1

Create username: Deck

Create password: admin213

Registration successful

1. Register
2. Login
3. Exit

Choose an option: 2

Enter username: Deck

Enter password: admin213

Login successful

1. Register
2. Login
3. Exit

Choose an option: 3

Explanation of Security Improvements

Aspect	Insecure Version	Secure Version
Credentials	Hardcoded	User-defined
Password storage	Plain text	Hashed (SHA-256)
Registration	Not available	Implemented
Login validation	Simple check	Secure hash comparison
Scalability	Single user	Multiple users supported

Why This Is More Secure

- Passwords are **never stored or compared in plain text**
 - Uses **cryptographic hashing (SHA-256)**
 - Prevents unauthorized access if source code is exposed
 - Encourages **secure authentication practices**
 - Demonstrates **privacy-aware AI code revision**
-

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 Python program for a loan approval system.

Code:

```
def approve_loan(name, gender, income):  
    if gender == "male" and income > 30000:  
        return "Loan Approved"  
    elif gender == "female" and income > 50000:  
        return "Loan Approved"  
    else:  
        return "Loan Rejected"  
  
applicant_name = "Anita"  
applicant_gender = "female"  
applicant_income = 40000
```

```
print(approve_loan(applicant_name, applicant_gender, applicant_income))
```

Output:

```
PS
C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces>
&
C:/Users/rohit/AppData/Local/Programs/Python/Python3
14/python.exe
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/
r_800/p2/1742b.py
Loan Rejected
```

Bias Analysis

Detected Biases

1. Gender-Based Decision Making

- Male applicants are approved at a lower income threshold than female applicants.

2. Irrelevant Personal Attributes

- Gender and name are not financially relevant for loan approval.

3. Unfair Treatment

- Applicants with the same income receive different outcomes based only on gender.

Fairness Issues

- Loan eligibility should depend on financial capability, not identity.
- Gender-based conditions violate fairness and ethical AI principles.
- Such bias can lead to systematic discrimination.

Revised one:

Prompt:

Modify the loan approval system so that decisions are fair and based only on relevant financial factors

Code:

```
def approve_loan(income, credit_score):  
    if income >= 40000 and credit_score >= 700:  
        return "Loan Approved"  
    else:  
        return "Loan Rejected"  
  
applicant_income = 40000  
applicant_credit_score = 720  
  
print(approve_loan(applicant_income, applicant_credit_score))
```


Output:

```
PS
C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces>
&
C:/Users/rohit/AppData/Local/Programs/Python/Python3
14/python.exe
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/
r_800/p2/1742b.py
Loan Approved
```

Comparison:

Issue	Original Code	Revised Code
Gender usage	Included	Removed
Name usage	Included	Removed
Decision basis	Identity + income	Financial metrics only
Fairness	Biased	Fair and objective

Bias Mitigation Strategies

1. Avoid using sensitive attributes (gender, name, race)
2. Use explainable and measurable criteria
3. Regularly audit decision logic for bias

4. Test outcomes across diverse inputs

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:**
 - o **Clear inline comments**
 - o **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:

Write a Python program to implement Binary Search using recursion.

The program should search for an element in a sorted list and explain the recursive steps.

Code:

```
def binary_search(arr, target, left, right):  
    # Base case: if the search space is invalid  
    if left > right:  
        return -1  
  
    # Find the middle index  
    mid = (left + right) // 2  
  
    # If target is found at mid  
    if arr[mid] == target:  
        return mid  
  
    # If target is smaller, search left half  
    elif target < arr[mid]:  
        return binary_search(arr, target, left, mid - 1)  
  
    # If target is larger, search right half  
    else:  
        return binary_search(arr, target, mid + 1, right)  
  
numbers = [1, 3, 5, 7, 9, 11, 13]  
element = 7  
  
result = binary_search(numbers, element, 0, len(numbers) - 1)  
  
if result != -1:  
    print("Element found at index:", result)  
else:  
    print("Element not found")
```

Output:

```
PS
C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces>
&
C:/Users/rohit/AppData/Local/Programs/Python/Python3
14/python.exe
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/
r_800/p2/1742b.py
Element found at index: 3
```

Explanation:

Binary search works only on a sorted list.

The function checks the middle element of the list.

If the middle element matches the target, the index is returned.

If the target is smaller, the function recursively searches the left half.

If the target is larger, the function recursively searches the right half.

If the element is not found, the function returns `-1`.

Analysis

Base Case Explanation

- `if left > right:`
This clearly indicates when the search space becomes invalid.
- Correctly explained as the condition where the element does not exist.

Recursive Case Explanation

- Recursive calls reduce the problem size by half each time.
- Logic for left and right halves is clearly separated.
- Comments correctly match the recursive calls.

Criteria	Evaluation
Code readability	Good
Use of comments	Clear and helpful
Base case clarity	Clearly explained
Recursive logic clarity	Easy to follow
Suitable for beginners	Yes

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:

Create a Python program that scores job applicants based on their skills, experience, education, name, and gender.

Code:

```
def score_applicant(name, gender, skills, experience, education):  
    score = skills * 2 + experience  
  
    if education == "masters":  
        score += 5  
  
    if gender == "male":  
        score += 3  
  
    return score  
  
applicant_name = "Rahul"  
applicant_gender = "male"  
applicant_skills = 8  
applicant_experience = 5  
applicant_education = "bachelors"  
  
print("Applicant Score:", score_applicant(  
    applicant_name,  
    applicant_gender,  
    applicant_skills,
```

```
applicant_experience,  
applicant_education  
)
```

Output:

```
PS  
C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces>  
&  
C:/Users/rohit/AppData/Local/Programs/Python/Python3  
14/python.exe  
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/  
r_800/p2/1742b.py  
Applicant Score: 24
```

Ethical Analysis of the Generated Code

Potential Bias Identified

1. Gender-Based Advantage
 - Male applicants receive extra score points.
2. Irrelevant Personal Attributes
 - Name and gender do not indicate job performance.
3. Unfair Hiring Influence
 - Candidates with equal skills and experience may receive different scores.

Why This Is Ethically Problematic

- Violates principles of equal opportunity

- Encourages discriminatory hiring practices
- Reduces trust in AI-assisted recruitment systems
- May have legal and social consequences

Revised one:

Prompt:

Modify the applicant scoring system so that it is fair, objective, and based only on job-relevant qualifications

Code:

```
def score_applicant(skills, experience, education_level):  
    score = (skills * 2) + (experience * 1.5)  
  
    if education_level == "masters":  
        score += 5  
    elif education_level == "phd":  
        score += 7  
  
    return score  
  
applicant_skills = 8  
applicant_experience = 5  
applicant_education = "bachelors"  
  
print("Applicant Score:", score_applicant(  
    applicant_skills,  
    applicant_experience,  
    applicant_education  
))
```


Output:

```
PS
C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces>
&
C:/Users/rohit/AppData/Local/Programs/Python/Python3
14/python.exe
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/
r_800/p2/1742b.py
Applicant Score: 23.5
```

Explanation:

Aspect	Original Code	Revised Code
Gender usage	Included	Removed
Name usage	Included	Removed
Scoring basis	Identity + skills	Skills & qualifications only
Fairness	Biased	Fair and objective

Why the Revised System Is Ethical

- Scores are based only on job-related criteria

- Removes all personal identity attributes
 - Treats all applicants equally
 - Aligns with ethical AI and responsible hiring principles
-

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:

Generate a Python program that processes employee details including name, gender, and role.

Code:

```
employee_male = "Ramesh"  
employee_female = "Sita"  
  
if employee_male:  
    print("Assign technical role to male employee")  
  
if employee_female:  
    print("Assign HR role to female employee")
```

Output:

```
PS  
C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces>  
&  
C:/Users/rohit/AppData/Local/Programs/Python/Python3  
14/python.exe  
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/  
r_800/p2/1742b.py  
Assign technical role to male employee  
Assign HR role to female employee
```

Inclusiveness Analysis

Issues Identified

1. Gender-Specific Variable Names

- Variables like `employee_male` and `employee_female` assume gender identity.

2. Role Assignment Based on Gender

- Technical role is assigned to males.
- HR role is assigned to females.

3. Stereotyping

- Reinforces traditional and unfair gender stereotypes.

4. Lack of Inclusiveness

- Does not account for non-binary or gender-neutral identities.

Why This Is Non-Inclusive

- Gender is unrelated to job capability or role suitability.
- Such assumptions can create bias and discrimination.
- Modern ethical software design promotes neutral and respectful coding practices.

Revised one:

Prompt:

Rewrite the employee processing code using gender-neutral variable names and avoid gender-based assumptions.

Code:

```
employee_name = "Alex"  
employee_role = "Technical"  
  
print(f"{employee_name} is assigned to the {employee_role} role")
```

Output:

```
PS  
C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces>  
&  
C:/Users/rohit/AppData/Local/Programs/Python/Python3  
14/python.exe  
c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/  
r_800/p2/1742b.py  
Alex is assigned to the Technical role
```

Explanation:

What Was Non-Inclusive

- Use of gender-specific variables
- Role assignment based on gender assumptions
- Exclusion of gender-neutral identities

Aspect	Original Code	Revised Code
Variable naming	Gender-based	Gender-neutral
Role logic	Based on gender	Based on role only
Identity assumptions	Present	Removed
Inclusiveness	Low	High