

## ASSIGNMENT-5.3

**NAME:T.SASYA**

**BATCH:20**

**ROLLNO:2303A51346**

### **TASK-1\_\_ AI-Generated (Insecure) Login Code Example**

**PROMPT:create a insecure login system in python**

```
# Insecure Login System

username = "admin"
password = "12345"

input_user = input("Enter username: ")
input_pass = input("Enter password: ")

if input_user == username and input_pass == password:
    print("Login Successful!")
else:
    print("Invalid Credentials")

Enter username: admin
Enter password: 12345
Login Successful!
```

#### **2. Identification of Security Risks**

##### **(a) Hardcoded Credentials**

```
username = "admin"
```

```
password = "12345"
```

- Username and password are written directly in the source code.
- Anyone with access to the file can see them.

##### **(b) Plain Text Password Storage**

```
password = "12345"
```

- Password is stored as plain text.
- If the file is leaked, all credentials are exposed.

### (c) Plain Text Comparison

```
if input_pass == password:
```

- Passwords are compared directly.
- No encryption or hashing is used.

### (d) No Input Validation

- User input is accepted without validation.
- Vulnerable to malformed or malicious input.

### (e) No Protection Against Attacks

- No protection from:
  - Brute-force attacks
  - Credential stuffing
  - Timing attacks

## Revised Secure Version of the Code

**prompt: create a revised and secure version of code where the username and password should be hidden**

```
import hashlib
import getpass

# Simulated database (in real systems, use an actual database)
users_db = {
    "admin": "5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8"
    # This is SHA-256 hash of "password"
}

def hash_password(password):
    """Hash a password using SHA-256"""
    return hashlib.sha256(password.encode()).hexdigest()

def validate_input(username, password):
    """Basic input validation"""
    if not username or not password:
        return False

    if len(username) < 3 or len(password) < 6:
        return False

    return True

def login():
    print("== Secure Login System ==")

    username = input("Enter username: ").strip()
    password = getpass.getpass("Enter password: ")

    # Validate input
    if not validate_input(username, password):
        print("Invalid input format.")
        return

    # Check if user exists
    if username not in users_db:
        print("Invalid credentials.")
        return

    # Hash entered password
    hashed_input = hash_password(password)

    # Compare hashed passwords
    if hashed_input == users_db[username]:
        print("Login Successful!")
    else:
        print("Invalid credentials.")
```

```
if __name__ == "__main__":
    login()

...
*** === Secure Login System ===
Enter username: admin
Enter password: .....
Login Successful!
```

### Brief Explanation of Improvements

#### (a) Removed Hardcoded Plain Passwords

Before:

```
password = "12345"
```

After:

```
users_db = {
    "admin": "hashed_value"
}
```

- ✓ Passwords are now stored as hashes.

#### (b) Password Hashing

```
hashlib.sha256(password.encode()).hexdigest()
```

- ✓ Converts passwords into irreversible hashes.
- ✓ Even if data is leaked, real passwords cannot be easily recovered.

#### (c) Secure Password Input

```
import getpass

password = getpass.getpass()

✓ Hides password while typing.
```

#### (d) Input Validation

```
if not validate_input(username, password):
```

✓ Prevents:

- Empty input
- Very weak credentials
- Invalid formats

#### (e) No Plain Text Comparison

Before:

```
if input_pass == password:
```

After:

```
if hashed_input == users_db[username]:
```

✓ Only hashed values are compared.

#### (f) Better Error Handling

```
print("Invalid credentials.")
```

✓ Does not reveal whether username or password is wrong (prevents user enumeration).

## TASK-2\_\_Bias Detection in AI-Generated Decision Systems

**Prompt:** Generate a python code on Bias detection in AI-Generated decision systems

```

# Biased Loan Approval System

def approve_loan(name, gender, income, credit_score):

    # Bias: Favoring male applicants
    if gender.lower() == "male":
        credit_score += 20

    # Bias: Certain names are favored
    if name.lower() in ["rahul", "arjun", "rohit"]:
        credit_score += 15

    if income >= 30000 and credit_score >= 650:
        return "Loan Approved"
    else:
        return "Loan Rejected"

# Test cases
print(approve_loan("Rahul", "Male", 40000, 640))
print(approve_loan("Priya", "Female", 40000, 640))

```

```

Loan Approved
Loan Rejected

```

### Problem:

- Male applicants get extra points.
- Female applicants are disadvantaged

```

# Bias: Certain names are favored
if name.lower() in ["rahul", "arjun", "rohit"]:
    credit_score += 15

```

- Certain names get preference.
- Indicates cultural/social bias.
- Discriminates against others.

### Revised Version

**Prompt:** Give me a revised version of the code where there is no gender bias and ensure fairness

```

# Fair Loan Approval System

def approve_loan(income, credit_score, employment_years):

    # Only financial factors are used
    if income < 25000:
        return "Loan Rejected"

    if credit_score < 650:
        return "Loan Rejected"

    if employment_years < 1:
        return "Loan Rejected"

    return "Loan Approved"

# Test cases
print(approve_loan(40000, 640, 2))
print(approve_loan(40000, 700, 2))

```

Loan Rejected  
Loan Approved

## Bias Mitigation Strategies

### 1. Feature Selection Control

Only use relevant attributes:

- ✓ Income
- ✓ Credit score
- ✓ Employment history
- ✓ Existing liabilities

#### **✗** Avoid:

- Gender
- Religion
- Caste
- Name
- Region

### 2. Bias Auditing

Regularly test system with:

# Same profile, different genders

```
approve_loan(40000, 700, 2) # Should always match
```

✓ Check consistency.

### 3. Fairness Testing

Use fairness metrics:

Metric	Meaning
Demographic Parity	Equal approval rates
Equal Opportunity	Equal true positive rates
Disparate Impact	$\leq 80\%$ rule

### 4. Diverse Training Data

- Use balanced datasets.
- Avoid historical discrimination patterns.

### 5. Explainable AI (XAI)

- Log why decisions are made.
- Provide users explanations.

Example:

Rejected due to credit score < 650

### 6. Human Review Layer

- Borderline cases reviewed manually.
- Prevents automated unfair rejection.

### 7. Comparison Summary

Aspect	Biased System	Fair System
--------	---------------	-------------

Gender Used	Yes <input checked="" type="checkbox"/>	No <input checked="" type="checkbox"/>
Name Used	Yes <input checked="" type="checkbox"/>	No <input checked="" type="checkbox"/>
Transparency	Low <input checked="" type="checkbox"/>	High <input checked="" type="checkbox"/>
Legal Safety	Low <input checked="" type="checkbox"/>	High <input checked="" type="checkbox"/>
Trust Level	Weak <input checked="" type="checkbox"/>	Strong <input checked="" type="checkbox"/>

## 8. Conclusion

AI-generated systems may unintentionally embed social bias.

By removing sensitive attributes, applying objective criteria, and auditing decisions regularly, fairness can be ensured.

- ✓ Ethical
- ✓ Legal
- ✓ Trustworthy
- ✓ Transparent

## Task-3 Transparency and Explainability in AI-Generated Code (Recursive Binary Search)

Prompt:#Recursive Binary search Program

```
# Recursive Binary Search Program
# This program searches for a target element in a sorted list using recursion.

def binary_search(arr, left, right, target):
    """
    Performs recursive binary search on a sorted list.

    Parameters:
    arr    : Sorted list of elements
    left   : Starting index of the search range
    right  : Ending index of the search range
    target : Element to be searched

    Returns:
    Index of target if found, otherwise -1
    """

    # ----- Base Case -----
    # If left index becomes greater than right,
    # it means the element is not present.
    if left > right:
        return -1

    # ----- Recursive Step -----
    # Find the middle index of the current range
    mid = (left + right) // 2
```

```

# If the middle element is the target, return its index
if arr[mid] == target:
    return mid

# If target is smaller, search in the left half
elif target < arr[mid]:
    return binary_search(arr, left, mid - 1, target)

# If target is larger, search in the right half
else:
    return binary_search(arr, mid + 1, right, target)

----- Main Program -----
if __name__ == "__main__":
    numbers = [2, 5, 8, 12, 16, 23, 38, 45, 56, 72]
    target = 23

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

    if result != -1:
        print("Element found at index:", result)
    else:
        print("Element not found")

```

• Element found at index: 5

## AI-Generated Step-by-Step Explanation of Recursive Logic

### Step 1: Start with Full List

The search begins with the entire list:

`left = 0`

`right = len(arr) - 1`

### Step 2: Base Case (Stopping Condition)

`if left > right:`

`return -1`

- ✓ This condition stops recursion.
- ✓ It means the search range is empty.
- ✓ So, the element is not present.

### **Step 3: Find the Middle Element**

```
mid = (left + right) // 2
```

- ✓ Divides the list into two halves.

### **Step 4: Compare Middle with Target**

```
if arr[mid] == target:
```

- ✓ If match is found → return index.

### **Step 5: Recursive Case (Left Sub-Array)**

```
elif target < arr[mid]:
```

```
    return binary_search(arr, left, mid - 1, target)
```

- ✓ Searches the left half.

### **Step 6: Recursive Case (Right Sub-Array)**

```
else:
```

```
    return binary_search(arr, mid + 1, right, target)
```

- ✓ Searches the right half.

### **Step 7: Repeat Until Base Case**

Steps 2–6 repeat until:

- Element is found, or
- Search space becomes empty.

## **3. Analysis of Explanation and Comments**

### **(a) Base Case Explanation**

**Code:**

```
if left > right:
```

```
    return -1
```

**Explanation Quality:**  Clear

- Explains when recursion stops.
- Easy to understand.
- Properly linked to logic.

### (b) Recursive Case Explanation

**Code:**

```
return binary_search(arr, left, mid - 1, target)
return binary_search(arr, mid + 1, right, target)
```

**Explanation Quality:**  Accurate

- Explains left and right recursion.
- Matches code behavior.
- No ambiguity.

### (c) Comment-Code Consistency

Part	Comment Matches Code?
------	-----------------------

Base case  Yes

Mid calculation  Yes

Left recursion  Yes

Right recursion  Yes

Return values  Yes

✓ No misleading comments found

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

## PROMPT:#Job Application Scoring System

```
# Job Applicant Scoring System (Biased Version)

def calculate_score(name, gender, skills, experience, education):

    score = 0

    # Bias: Gender preference
    if gender.lower() == "male":
        score += 10

    # Bias: Certain names are favored
    if name.lower() in ["rahul", "arjun", "rohit"]:
        score += 5

    # Skill evaluation
    score += len(skills) * 5

    # Experience evaluation
    score += experience * 3

    # Education evaluation
    if education.lower() == "phd":
        score += 20
    elif education.lower() == "masters":
        score += 15

    elif education.lower() == "bachelors":
        score += 10

    return score

# Test case
applicant_score = calculate_score(
    "Priya",
    "Female",
    ["Python", "ML", "SQL"],
    3,
    "Masters"
)

print("Applicant Score:", applicant_score)
```

Applicant Score: 39

### 2. Identification of Potential Bias

#### (a) Gender-Based Bias

```
if gender.lower() == "male":
```

```
    score += 10
```

✗ Problem:

- Male candidates automatically get higher scores.
- Female and non-binary candidates are disadvantaged.

### (b) Name-Based Bias

```
if name.lower() in ["rahul", "arjun", "rohit"]:
```

```
    score += 5
```

**✗ Problem:**

- Certain cultural names are favored.
- Discriminates against other applicants.

### (c) Use of Irrelevant Features

Feature	Used in Scoring	Relevant?
---------	-----------------	-----------

Name	Yes	<b>✗</b> No
------	-----	-------------

Gender	Yes	<b>✗</b> No
--------	-----	-------------

Skills	Yes	<input checked="" type="checkbox"/> Yes
--------	-----	---

Experience	Yes	<input checked="" type="checkbox"/> Yes
------------	-----	---

Education	Yes	<input checked="" type="checkbox"/> Yes
-----------	-----	---

**✓** Only professional attributes should be used.

## 3. Ethical Analysis of the Scoring Logic

### 1. Violation of Equal Opportunity

- Candidates with equal qualifications receive different scores.
- Breaks fairness principles.

### 2. Reinforcement of Social Bias

- Encourages stereotypes.
- Marginalizes groups.

### 3. Legal and Compliance Risks

- Violates employment equality laws.
- May conflict with HR and labor regulations.

### 4. Lack of Transparency

- Applicants cannot understand why they scored lower.
- Reduces system trust.

### 5. Moral Responsibility

- Hiring systems affect livelihoods.
- Biased AI may cause long-term harm.

## Revised Version

**Prompt:** Give me a revised version of the code

```

# Fair Job Applicant Scoring System

def calculate_score.skills, experience, education, certifications=0):

    score = 0

    # Skill evaluation (max 40 points)
    score += min(len.skills) * 5, 40)

    # Experience evaluation (max 30 points)
    score += min(experience * 3, 30)

    # Education evaluation (max 20 points)
    education_scores = {
        "phd": 20,
        "masters": 15,
        "bachelors": 10,
        "diploma": 5
    }

    score += education_scores.get(education.lower(), 0)

    # Certification bonus (max 10 points)
    score += min(certifications * 5, 10)

    return score

# Test cases (same profile, different names/genders)
print(calculate_score(["Python", "ML", "SQL"], 3, "Masters", 1))
print(calculate_score(["Python", "ML", "SQL"], 3, "Masters", 1))

...
44
44

```

### Explanation of Improvements

#### (a) Removed Personal Identifiers

Before:

name, gender

After:

(skills, experience, education, certifications)

✓ No sensitive attributes are used.

#### (b) Objective Scoring Rules

- ✓ Points based on measurable qualifications.
- ✓ Upper limits prevent unfair advantage.

### (c) Transparency

Scoring is predictable and explainable:

Factor      Points

Skills       $\leq 40$

Experience  $\leq 30$

Education  $\leq 20$

Certificates  $\leq 10$

### (d) Consistency

Same profile  $\rightarrow$  Same score

✓ Ensures fairness.

## 6. Ethical Mitigation Strategies

### 1. Bias Auditing

Regularly test system:

```
# Same qualifications, different identities  
calculate_score(["Python"], 5, "Masters")
```

✓ Results must match.

### 2. Feature Governance

- Approve only job-related features.
- Document why each feature is used.

### Human Oversight

- Final decisions reviewed by HR.
- Prevents blind automation.

### 4. Explainability Reports

Provide feedback:

Example:

Score: 72

+ Skills: 25

+ Experience: 9

+ Education: 15

+ Certifications: 5

## 5. Dataset Review (If ML-Based)

- Remove historical discrimination.
- Balance training data.

## 7. Summary Comparison

Aspect	Biased System	Fair System
Gender Used	Yes <input checked="" type="checkbox"/>	No <input checked="" type="checkbox"/>
Name Used	Yes <input checked="" type="checkbox"/>	No <input checked="" type="checkbox"/>
Transparency	Low <input checked="" type="checkbox"/>	High <input checked="" type="checkbox"/>
Legal Safety	Weak <input checked="" type="checkbox"/>	Strong <input checked="" type="checkbox"/>
Ethical Level	Poor <input checked="" type="checkbox"/>	High <input checked="" type="checkbox"/>

## 8. Conclusion

The original AI-generated scoring system contained unethical biases based on gender and name.

After removing irrelevant personal attributes and applying objective, job-related criteria, the revised system becomes:

✓ Fair

✓ Transparent

- ✓ Legally safer
- ✓ Ethically responsible

## Task 5: Inclusiveness and Ethical Variable Design Scenario

Prompt: #Employee Information System

```
# Employee Information System (Non-Inclusive Version)

def process_employee(name, gender, age, salary):

    print("Employee Name:", name)

    # Gender-based role assumption
    if gender.lower() == "male":
        role = "Team Leader"
    elif gender.lower() == "female":
        role = "Assistant"
    else:
        role = "Support Staff"

    # Salary bonus based on gender
    if gender.lower() == "male":
        bonus = salary * 0.10
    else:
        bonus = salary * 0.05

    print("Gender:", gender)
    print("Assigned Role:", role)
    print("Bonus:", bonus)
```

```
print("Gender:", gender)
print("Assigned Role:", role)
print("Bonus:", bonus)

# Example
process_employee("Anita", "Female", 28, 40000)
```

- Employee Name: Anita  
Gender: Female  
Assigned Role: Assistant  
Bonus: 2000.0

### Identification of Non-Inclusive Elements

#### (a) Gender-Specific Variable

gender

✗ Problem:

- Forces binary classification.

- Excludes non-binary and gender-diverse identities.

### **(b) Role Assignment Based on Gender**

```
if gender.lower() == "male":  
    role = "Team Leader"
```

✗ Problem:

- Assumes leadership = male.
- Reinforces stereotypes.

### **(c) Unequal Bonus Logic**

```
if gender.lower() == "male":  
    bonus = salary * 0.10
```

✗ Problem:

- Pay discrimination.
- Unethical and often illegal.

### **(d) Non-Inclusive Output**

```
print("Gender:", gender)
```

✗ Problem:

- Highlights personal identity unnecessarily.

```

# Employee Information System (Inclusive Version)

def process_employee(name, position, age, salary, performance_rating):

    print("Employee Name:", name)
    print("Position:", position)

    # Role is based on job title, not identity
    role = position

    # Bonus based on performance
    if performance_rating >= 4.5:
        bonus = salary * 0.12
    elif performance_rating >= 3.5:
        bonus = salary * 0.08
    else:
        bonus = salary * 0.05

    print("Assigned Role:", role)
    print("Performance Rating:", performance_rating)
    print("Bonus:", bonus)

# Example
process_employee("Anita", "Software Engineer", 28, 40000, 4.2)

```

\*\*\* Employee Name: Anita  
 Position: Software Engineer  
 Assigned Role: Software Engineer  
 Performance Rating: 4.2  
 Bonus: 3200.0

Below is a complete, structured solution for **Task 5: Inclusiveness and Ethical Variable Design**.

## Task 5: Inclusiveness and Ethical Variable Design

### 1. Original AI-Generated Code (Non-Inclusive Example)

PROMPT:# Employee Information System (Non-Inclusive Version)

```
# Employee Information System (Non-Inclusive Version)

def process_employee(name, gender, age, salary):

    print("Employee Name:", name)

    # Gender-based role assumption
    if gender.lower() == "male":
        role = "Team Leader"
    elif gender.lower() == "female":
        role = "Assistant"
    else:
        role = "Support Staff"

    # Salary bonus based on gender
    if gender.lower() == "male":
        bonus = salary * 0.10
    else:
        bonus = salary * 0.05

    print("Gender:", gender)
    print("Assigned Role:", role)
    print("Bonus:", bonus)
```

```
# Example
process_employee("Anita", "Female", 28, 40000)
```

```
Employee Name: Anita
Gender: Female
Assigned Role: Assistant
Bonus: 2000.0
```

```
# Example
```

```
process_employee("Anita", "Female", 28, 40000)
```

## 2. Identification of Non-Inclusive Elements

### (a) Gender-Specific Variable

gender

✗ Problem:

- Forces binary classification.
- Excludes non-binary and gender-diverse identities.

### (b) Role Assignment Based on Gender

```
if gender.lower() == "male":  
    role = "Team Leader"
```

#### ✗ Problem:

- Assumes leadership = male.
- Reinforces stereotypes.

### (c) Unequal Bonus Logic

```
if gender.lower() == "male":  
    bonus = salary * 0.10
```

#### ✗ Problem:

- Pay discrimination.
- Unethical and often illegal.

### (d) Non-Inclusive Output

```
print("Gender:", gender)
```

#### ✗ Problem:

- Highlights personal identity unnecessarily.

---

## Revised Inclusive and Gender-Neutral Code

```

# Employee Information System (Inclusive Version)

def process_employee(name, position, age, salary, performance_rating):

    print("Employee Name:", name)
    print("Position:", position)

    # Role is based on job title, not identity
    role = position

    # Bonus based on performance
    if performance_rating >= 4.5:
        bonus = salary * 0.12
    elif performance_rating >= 3.5:
        bonus = salary * 0.08
    else:
        bonus = salary * 0.05

    print("Assigned Role:", role)
    print("Performance Rating:", performance_rating)
    print("Bonus:", bonus)

```

```

# Example
process_employee("Anita", "Software Engineer", 28, 40000, 4.2)

```

```

...
Employee Name: Anita
Position: Software Engineer
Assigned Role: Software Engineer
Performance Rating: 4.2
Bonus: 3200.0

```

PROMPT:# Employee Information System (Inclusive Version)

### Explanation of Improvements

#### (a) Removal of Gender Variable

**Before:**

gender

**After:**

position, performance\_rating

- ✓ No identity-related variable required.

#### (b) Neutral Role Assignment

**Before:**

```
if gender == "male": role = "Team Leader"
```

**After:**

```
role = position
```

- ✓ Role depends on job designation, not gender.

### (c) Fair Bonus Calculation

**Before:**

```
bonus = salary * 0.10 (for males)
```

**After:**

```
bonus = salary * 0.12 (based on performance)
```

- ✓ Rewards merit, not identity.

### (d) Respectful Data Handling

- ✓ No unnecessary personal attributes displayed.
- ✓ Protects privacy.

## 5. Summary Table

Aspect	Original Code	Revised Code
Gender Variable	Yes <input checked="" type="checkbox"/>	No <input checked="" type="checkbox"/>
Stereotyping	Present <input checked="" type="checkbox"/>	Removed <input checked="" type="checkbox"/>
Bonus Bias	Present <input checked="" type="checkbox"/>	Fair <input checked="" type="checkbox"/>
Inclusiveness	Low <input checked="" type="checkbox"/>	High <input checked="" type="checkbox"/>
Ethical Design	Weak <input checked="" type="checkbox"/>	Strong <input checked="" type="checkbox"/>

## 6. Conclusion

The original AI-generated code made unfair assumptions based on gender and reinforced stereotypes in role and pay allocation.

The revised version follows inclusive design principles by:

- ✓ Using neutral variables
- ✓ Applying merit-based logic
- ✓ Respecting diversity
- ✓ Promoting fairness

This results in more ethical, professional, and inclusive software.