

AI Assisted Coding Lab 5.3

Chithra

2303A52137

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

Prompt:

“Generate a simple login system in Python that asks for a username and password and verifies the user.”

CODE:

```
s
def login():
    username = input("Enter username: ")
    password = input("Enter password: ")

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

login()
```

Output:

```
• Enter username: admin
  Enter password: admin123
  Login successful!
```

Prompt: revised version

“Generate a secure Python program for a multi-user authentication system. The program should allow users to register and log in, store usernames and hashed passwords in a dictionary, use SHA-256 for password hashing, validate inputs (non-empty username, minimum password length), and provide a menu-driven interface for registration, login, and exit.”

Code:

```
import hashlib

# Dictionary to store multiple users
users_db = {}

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

def register():
    print("\n--- User Registration ---")
    username = input("Enter username: ").strip()
    password = input("Enter password: ").strip()

    if not username or not password:
        print("Username and password cannot be empty.")
        return

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

    if len(password) < 6:
        print("Password must be at least 6 characters long.")
        return

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

def login():
    print("\n--- User Login ---")
    username = input("Enter username: ").strip()
    password = input("Enter password: ").strip()

    if username in users_db and users_db[username] == hash_password(password):
        print("Login successful!")
    else:
        print("Invalid username or password.")

def main():
    while True:
        print("\n1. Register")
        print("2. Login")
        choice = input("Enter your choice: ")

        if choice == "1":
            register()
        elif choice == "2":
            login()
        else:
            print("Invalid choice. Please enter 1 or 2.")
```

```
def main():
    while True:
        print("\n1. Register")
        print("2. Login")
        print("3. Exit")

        choice = input("Choose an option: ")

        if choice == "1":
            register()
        elif choice == "2":
            login()
        elif choice == "3":
            print("Exiting program.")
            break
        else:
            print("Invalid choice.")

main()
```

Output:

```
...
1. Register
2. Login
3. Exit
Choose an option: 1

--- User Registration ---
Enter username: chithra
Enter password: chithra@123
Registration successful!

1. Register
2. Login
3. Exit
Choose an option: 2

--- User Login ---
Enter username: chithra
Enter password: chithra@123
Login successful!

1. Register
2. Login
3. Exit
Choose an option: 3
Exiting program.
```

EXPLANATION:

Non-Secure Login System

A non-secure login system does not protect user information properly.

- Usernames and passwords are written directly in the code
- Passwords are stored and checked in plain text
- Anyone who sees the code can read all passwords
- There is no checking for empty or weak passwords
- User data can be easily stolen or misused

Result: User privacy is not safe and hackers can easily access accounts.

✓ Secure Login

A secure login system protects user information.

- Usernames and passwords are not written in the code
- Passwords are converted into a hidden form (hashed)
- Even if data is stolen, passwords cannot be read
- The system checks for valid input and strong passwords
- Many users can register and log in safely

Result: User data is safe and the system is reliable.

Task 2: Bias Detection in AI-Generated Decision Systems

Prompt 1 (Basic Loan Approval System)

“Create a simple Python loan approval system that decides whether a loan should be approved based on applicant details.”

Code:

```
def loan_approval(name, gender, income, credit_score):
    if gender == "Male" and income >= 30000 and credit_score >=
        return "Loan Approved"
    elif gender == "Female" and income >= 50000 and credit_score >=
        return "Loan Approved"
    else:
        return "Loan Rejected"

# Test cases
print(loan_approval("Rahul", "Male", 35000, 650))
print(loan_approval("Anita", "Female", 35000, 650))
```

Output:

```
•• Loan Approved  
    Loan Rejected
```

Prompt 2 (Varying Name and Gender)

“Create a loan approval system in Python where applicants have different names, genders, income, and credit score.”

Code:

```
def loan_approval(name, gender, income, credit_score):  
    if gender == "Male":  
        if income > 30000 and credit_score > 600:  
            return f"{name}: Loan Approved"  
    elif gender == "Female":  
        if income > 50000 and credit_score > 700:  
            return f"{name}: Loan Approved"  
    return f"{name}: Loan Rejected"  
  
# Same financial details, different gender  
print(loan_approval("Amit", "Male", 40000, 650))  
print(loan_approval("Priya", "Female", 40000, 650))
```

Output:

```
Amit: Loan Approved  
Priya: Loan Rejected
```

Explanation:

Code 1 Explanation

The loan decision depends on gender, income, and credit score.

- **Male and female applicants have different rules.**
- **This is unfair because gender should not affect loan approval.**

Code 2 Explanation

The program checks gender first before financial details.

- Applicants with the same income and credit score get different results.
- This causes bias and discrimination.
-

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

Prompt

“Generate a Python program that implements Binary Search using recursion to find an element in a sorted list. Include clear inline comments and explain the base case and recursive case step by step for beginners.”

Code:

```

def binary_search(arr, low, high, target):
    # Base case: if the search range is invalid
    if low > high:
        return -1 # Element not found

    # Find the middle index
    mid = (low + high) // 2

    # If the target is found at the middle
    if arr[mid] == target:
        return mid

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

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

# Sorted list
numbers = [2, 4, 6, 8, 10, 12, 14]

# Element to search
result = binary_search(numbers, 0, len(numbers) - 1, 10)

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

```

Output:

```
# Element to search
result = binary_search(numbers, 0, len(numbers))

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

...
*** Element found at index: 4
```

Explanation

- Binary Search works only on a sorted list
- The list is divided into two halves
- The middle element is checked:
 - If it matches, search stops
 - If the target is smaller, search the left half
 - If the target is larger, search the right half
- This process repeats using recursion
 - ◆ Base Case:
 - When low > high, the element is not present
 - ◆ Recursive Case:
 - The function calls itself with a smaller part of the list

Task 4: Ethical Evaluation of AI-Based Scoring Systems

Prompt

“Create a Python program that scores job applicants based on skills, work experience, and education level.”

Code:

```

def calculate_score.skills, experience_years, education_level):
    score = 0

    # Skill scoring (max 50 points)
    score += len.skills) * 10

    # Experience scoring (max 30 points)
    score += experience_years * 5

    # Education scoring
    if education_level == "PhD":
        score += 20
    elif education_level == "Masters":
        score += 15
    elif education_level == "Bachelors":
        score += 10
    else:
        score += 5

    return score

# Example applicant
applicant_skills = ["Python", "SQL", "Machine Learning"]
experience = 3 # years
education = "Bachelors"

final_score = calculate_score(applicant_skills, experience, education)

print("Applicant Score:", final_score)

```

Output:

```

final_score = calculate_score(applicant_skills, experience, education)

print("Applicant Score:", final_score)

...
Applicant Score: 55

```

Explanation:

- The function `calculate_score` computes a total score for a job applicant.
- It takes three inputs: `skills` (list), `experience_years` (int), and `education_level` (string).

- score starts at 0 and accumulates points from skills, experience, and education.
- Each skill adds 10 points to the score (max 50 points).
- Each year of experience adds 5 points (max 30 points).
- Education scoring is tiered: PhD = 20, Masters = 15, Bachelors = 10, others = 5.
- The final score is the sum of skill points, experience points, and education points.
- Example applicant has 3 skills, 3 years of experience, and a Bachelor's degree.
- calculate_score is called with these values and returns the total score.
- The score is printed with print("Applicant Score:", final_score) to show eligibility.

Task 5: Inclusiveness and Ethical Variable Design

Prompt:

Write a Python program that collects and processes employee details.

The program should ask for the employee's name, gender, and role,
and then print a personalized welcome message including a title based on gender.

Use simple input and print statements.

Code: Original AI-generated code snippet (non-inclusive)

```
# Original AI-generated code snippet
employee_name = input("Enter employee name: ")
employee_gender = input("Enter gender (male/female): ")
employee_role = input("Enter role (manager/assistant): ")

if employee_gender == "male":
    print(f"Welcome Mr. {employee_name}, your role is {employee_role}.")
elif employee_gender == "female":
    print(f"Welcome Ms. {employee_name}, your role is {employee_role}.")
else:
    print(f"Welcome {employee_name}, your role is {employee_role}.")
```

Output:

```
• Enter employee name: chithra  
Enter gender (male/female): female  
Enter role (manager/assistant): assistant  
Welcome Ms. chithra, your role is assistant.
```

Revised Inclusive and Gender-Neutral Code

```
# Inclusive and gender-neutral version  
employee_name = input("Enter employee name: ")  
employee_role = input("Enter role: ")  
  
print(f"Welcome {employee_name}, your role is {employee_role}.")
```

Output:

```
Enter employee name: chithra  
Enter role: manager  
Welcome chithra, your role is manager.
```

Explanation:

What was non-inclusive

1. **Gender-specific variables** – The code asked for `employee_gender` and expected “male” or “female” inputs.
2. **Gender-based assumptions** – The program applied titles “Mr.” or “Ms.” depending on the gender, assuming only two genders exist.
3. **Non-inclusive naming and logic** – Non-binary or gender-nonconforming employees would not be properly addressed.

How inclusiveness was improved

1. **Removed gender variable and conditions** – No assumptions about gender were made.
2. **Used gender-neutral greetings** – The code now addresses employees by name only.
3. **Simplified, inclusive design** – Focuses on universally relevant information like name and role, making it respectful and applicable to everyone.