AI-ASSISSTANT-CODING-LAB-5.3

Name: G. Kruthik Roshan

Batch:41

Roll-No:2303A52339

TASK 1: Privacy and Data Security in AI-Generated Code
PROMPT: This task demonstrates the importance of secure password storage. Compare insecure (plain text) vs secure (hashed) password systems.Learn how proper password hashing protects user data from security breaches.
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).

Code:

```
# ============================================================================
# TASK 1: Privacy and Data Security in AI-Generated Code
# ============================================================================
# PROMPT: This task demonstrates the importance of secure password storage.
# Compare insecure (plain text) vs secure (hashed) password systems.
# Learn how proper password hashing protects user data from security breaches.
# ============================================================================

import hashlib

# Storage for registered users (in-memory for demo purposes)
insecure_users = {}  # Plain text passwords - BAD
secure_users = {}    # Hashed passwords - GOOD

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

```python
def is_password_strong(password):
    """
    Check if password meets strong password criteria
    Requirements:
    - At least 8 characters long
    - Contains at least one uppercase letter
    - Contains at least one lowercase letter
    - Contains at least one digit
    - Contains at least one special character
    """
    if len(password) < 8:
        return False, "Password must be at least 8 characters long"

    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(c in "!@#$%^&*()_+-=[]{}|;:,.<>?" for c in password)
```

```python
    if not has_upper:
        return False, "Password must contain at least one uppercase letter"
    if not has_lower:
        return False, "Password must contain at least one lowercase letter"
    if not has_digit:
        return False, "Password must contain at least one digit"
    if not has_special:
        return False, "Password must contain at least one special character (!@#$%^&* etc

    return True, "Password is strong"

def insecure_register():
    """Insecure registration - stores plain text passwords"""
    print("\n--- INSECURE REGISTRATION ---")
    print("Issues: Stores passwords in plain text\n")

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

    if not username or not password:
        print("✗ Username and password cannot be empty!")
        return
    if username in insecure_users:
        print("✗ Username already exists!")
        return

    # SECURITY RISK: Storing password in plain text
    insecure_users[username] = password
    print(f"✓ Registration successful! (Password stored as: {password}) ⚠ INSECURE!"

def insecure_login():
    """Insecure login system"""
    print("\n--- INSECURE LOGIN SYSTEM ---")
    print("Issues: Plain text password comparison\n")

    if not insecure_users:
        print("✗ No users registered! Please register first.")
        return

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

```
    # Plain text comparison - SECURITY RISK
    if username in insecure_users and insecure_users[username] == password:
        print("✓ Login successful!")
        print(f"⚠️ Your password '{password}' is visible in database!")
    else:
        print("✗ Login failed! Username or password incorrect.")

def secure_register():
    """Secure registration - stores hashed passwords"""
    print("\n--- SECURE REGISTRATION ---")
    print("Improvements: Password hashing, strong password validation\n")

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

    # Input validation
    if not username or not password:
        print("✗ Username and password cannot be empty!")
        return
```

```
    # Check password strength
    is_strong, message = is_password_strong(password)
    if not is_strong:
        print(f"✗ Password is not strong enough! {message}")
        print("\nPassword Requirements:")
        print("  • At least 8 characters long")
        print("  • At least one uppercase letter (A-Z)")
        print("  • At least one lowercase letter (a-z)")
        print("  • At least one digit (0-9)")
        print("  • At least one special character (!@#$%^&* etc.)")
        return

    if username in secure_users:
        print("✗ Username already exists!")
        return

    # SECURE: Store hashed password
    secure_users[username] = hash_password(password)
    print("✓ Registration successful! Password stored securely (hashed).")

def secure_login():
    """Secure login system with hashed passwords"""
    print("\n--- SECURE LOGIN SYSTEM ---")
    print("Improvements: Password hashing, input validation\n")
```

```
print("\nSecurity Explanation:")
print("- Insecure: Plain text passwords (visible in database)")
print("- Secure: Hashed passwords (SHA-256), input validation")
```

OUTPUT:

```
======================================================================
TASK 1: Privacy and Data Security
======================================================================

[1] Insecure System (Plain Text)
[2] Secure System (Hashed)

Select system (1/2): 1

[A] Register
[B] Login

Select action (A/B): A

--- INSECURE REGISTRATION ---
Issues: Stores passwords in plain text

Enter username: Kruthik18R
Enter password: Kruthik18R@
✓ Registration successful! (Password stored as: Kruthik18R@) ⚠INSECURE!

Security Explanation:
- Insecure: Plain text passwords (visible in database)
- Secure: Hashed passwords (SHA-256), input validation
```

Justification:
This task demonstrates the importance of protecting user passwords by comparing plain text storage with secure password hashing. It highlights ethical data handling and privacy protection.

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.

PROMPT: This task illustrates how AI systems can contain hidden biases. Compare a biased loan approval system (using gender/name) vs a fair system.Learn to identify and eliminate discriminatory factors in decision-making.

CODE:

```
# ============================================================================
# TASK 2: Bias Detection in AI-Generated Decision Systems
# ============================================================================
# PROMPT: This task illustrates how AI systems can contain hidden biases.
# Compare a biased loan approval system (using gender/name) vs a fair system.
# Learn to identify and eliminate discriminatory factors in decision-making.
# ============================================================================

def biased_loan_approval(name, gender, income, credit_score):
    """Biased loan system - CONTAINS BIAS"""
    approved = False

    # BIAS: Gender-based approval
    if gender == "male" and income > 30000 and credit_score > 650:
        approved = True
    elif gender == "female" and income > 35000 and credit_score > 700:
        # Higher requirements for females - UNFAIR BIAS
        approved = True

    return approved

def fair_loan_approval(income, credit_score, employment_years):
    """Fair loan system based only on relevant financial factors"""
    # Gender-neutral, name-neutral evaluation
    if income > 30000 and credit_score > 650 and employment_years >= 2:
        return True
    return False
```

```
def task2_demo():
    print("\n" + "=" * 70)
    print("TASK 2: Bias Detection in Loan Approval System")
    print("=" * 70)

    choice = input("\nTest [1] Biased System or [2] Fair System? (1/2): ")

    if choice == "1":
        print("\n--- BIASED LOAN APPROVAL SYSTEM ---")
        name = input("Enter applicant name: ")
        gender = input("Enter gender (male/female): ").lower()
        income = int(input("Enter annual income: "))
        credit_score = int(input("Enter credit score: "))

        result = biased_loan_approval(name, gender, income, credit_score)
        print(f"\nLoan Approval: {'✓ APPROVED' if result else '✗ DENIED'}")
        print("\nBIAS WARNING: This system uses different criteria for different genders!")

    elif choice == "2":
        print("\n--- FAIR LOAN APPROVAL SYSTEM ---")
        income = int(input("Enter annual income: "))
        credit_score = int(input("Enter credit score: "))
        employment_years = int(input("Enter years of employment: "))

        result = fair_loan_approval(income, credit_score, employment_years)
        print(f"\nLoan Approval: {'✓ APPROVED' if result else '✗ DENIED'}")
        print("\nThis system uses only relevant financial criteria (no bias)")
    else:
        print("Invalid choice!")

    print("\nMitigation: Use only relevant financial criteria, no gender/name bias")
```

OUTPUT:

```
======================================================================
TASK 2: Bias Detection in Loan Approval System
======================================================================

Test [1] Biased System or [2] Fair System? (1/2): 1

--- BIASED LOAN APPROVAL SYSTEM ---
Enter applicant name: Kruthikroshan
Enter gender (male/female): male
Enter annual income: 800000
Enter credit score: 763

Loan Approval: ✓ APPROVED

BIAS WARNING: This system uses different criteria for different genders!

Mitigation: Use only relevant financial criteria, no gender/name bias
```

Justification:
This task shows how using gender or name in AI decisions can cause unfair bias and explains the need for bias-free, fair decision-making systems.

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

PROMPT: This task emphasizes the importance of transparent, explainable code.See how a binary search algorithm can be made clear with step-by-step output.Learn to write code that is easy to understand and debug for all developers.

CODE:

```
# ============================================================================
# TASK 3: Transparency and Explainability - Recursive Binary Search
# ============================================================================
# PROMPT: This task emphasizes the importance of transparent, explainable code.
# See how a binary search algorithm can be made clear with step-by-step output.
# Learn to write code that is easy to understand and debug for all developers.
# ============================================================================

def binary_search_recursive(arr, target, left, right, depth=0):
    """
    Recursive Binary Search with detailed explanations

    Parameters:
    - arr: sorted list of numbers
    - target: element to search for
    - left: left boundary index
    - right: right boundary index
    - depth: recursion depth for visualization

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

    # STEP 1: BASE CASE - Search space exhausted
    if left > right:
        print(f"{'  ' * depth}Base case: left > right, element not found")
        return -1

    # STEP 2: Calculate middle index
    mid = (left + right) // 2
    print(f"{'  ' * depth}Depth {depth}: Checking middle index {mid}, value = {arr[mid]}")
```

```
    # STEP 3: BASE CASE - Element found
    if arr[mid] == target:
        print(f"{'  ' * depth}Found! Target {target} at index {mid}")
        return mid

    # STEP 4: RECURSIVE CASE - Search left half
    elif arr[mid] > target:
        print(f"{'  ' * depth}Target {target} < {arr[mid]}, search LEFT half")
        return binary_search_recursive(arr, target, left, mid - 1, depth + 1)

    # STEP 5: RECURSIVE CASE - Search right half
    else:
        print(f"{'  ' * depth}Target {target} > {arr[mid]}, search RIGHT half")
        return binary_search_recursive(arr, target, mid + 1, right, depth + 1)

def task3_demo():
    print("\n" + "=" * 70)
    print("TASK 3: Transparent Recursive Binary Search")
    print("=" * 70)

    sorted_list = [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]
    print(f"\nSorted list: {sorted_list}")

    target = int(input("Enter number to search: "))

    print("\nStep-by-step execution:")
    result = binary_search_recursive(sorted_list, target, 0, len(sorted_list) - 1)

    if result != -1:
        print(f"\n✓ Result: Element {target} found at index {result}")
    else:
        print(f"\n✗ Result: Element {target} not found in list")
```

```
    print("\nTransparency Assessment:")
    print("✓ Clear comments explaining each step")
    print("✓ Base cases and recursive cases explained")
    print("✓ Visualization of recursion depth")
    print("✓ Easy to understand for beginners")
```

OUTPUT:

```
======================================================================
TASK 3: Transparent Recursive Binary Search
======================================================================

Sorted list: [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]
Enter number to search: 56

Step-by-step execution:
Depth 0: Checking middle index 5, value = 23
Target 56 > 23, search RIGHT half
  Depth 1: Checking middle index 8, value = 56
  Found! Target 56 at index 8

✓ Result: Element 56 found at index 8

Transparency Assessment:
✓ Clear comments explaining each step
✓ Base cases and recursive cases explained
✓ Visualization of recursion depth
✓ Easy to understand for beginners
```

Justification:
This task explains how clear, well-documented algorithms improve transparency and make AI systems easy to understand and trust.

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

PROMPT: This task explores fairness in automated scoring/ranking systems.Compare biased scoring (using name/gender) vs fair scoring (merit-based only).Learn to design evaluation systems that judge people on relevant criteria only.

CODE:

```
# ============================================================================
# TASK 4: Ethical Evaluation of AI-Based Scoring Systems
# ============================================================================
# PROMPT: This task explores fairness in automated scoring/ranking systems.
# Compare biased scoring (using name/gender) vs fair scoring (merit-based only).
# Learn to design evaluation systems that judge people on relevant criteria only.
# ============================================================================

def biased_applicant_score(name, gender, skills, experience, education):
    """Biased scoring - UNETHICAL"""
    score = 0

    # Skill scoring
    score += skills * 20
    score += experience * 10
    score += education * 15

    # BIAS: Gender-based bonus - UNETHICAL
    if gender == "male":
        score += 10  # Unfair advantage

    # BIAS: Name-based assumption - UNETHICAL
    if name in ["John", "Michael", "David"]:
        score += 5  # Cultural/name bias

    return score

def fair_applicant_score(skills_rating, years_experience, education_level,
                         certifications, portfolio_quality):
    """
    Fair scoring based only on relevant professional factors
    All personal attributes (name, gender, age, etc.) are excluded
    """
    score = 0
```

```
    # Skills (0-10 scale)
    score += skills_rating * 20  # Max: 200 points

    # Experience (years)
    score += min(years_experience * 10, 100)  # Max: 100 points

    # Education level (1=High School, 2=Bachelor, 3=Master, 4=PhD)
    score += education_level * 15  # Max: 60 points

    # Certifications count
    score += certifications * 10  # Max: varies

    # Portfolio quality (0-10 scale)
    score += portfolio_quality * 15  # Max: 150 points

    return score

def task4_demo():
    print("\n" + "=" * 70)
    print("TASK 4: Ethical Job Applicant Scoring System")
    print("=" * 70)

    choice = input("\nTest [1] Biased Scoring or [2] Fair Scoring? (1/2): ")

    if choice == "1":
        print("\n--- BIASED SCORING SYSTEM ---")
        name = input("Enter applicant name: ")
        gender = input("Enter gender (male/female): ").lower()
        skills = int(input("Enter skills rating (0-10): "))
        experience = int(input("Enter years of experience: "))
        education = int(input("Enter education level (1-4): "))

        score = biased_applicant_score(name, gender, skills, experience, education)
        print(f"\n✗ Total Score: {score}")
        print("BIAS DETECTED: Gender and name influence score!")
```

```
    elif choice == "2":
        print("\n--- FAIR SCORING SYSTEM ---")
        skills = int(input("Enter skills rating (0-10): "))
        experience = int(input("Enter years of experience: "))
        education = int(input("Enter education level (1=HS, 2=Bach, 3=Mast, 4=PhD): "))
        certs = int(input("Enter number of certifications: "))
        portfolio = int(input("Enter portfolio quality (0-10): "))

        score = fair_applicant_score(skills, experience, education, certs, portfolio)
        print(f"\n✓ Total Score: {score}")
        print("\nEthical Analysis:")
        print("✓ No gender, name, or identity factors")
        print("✓ Only relevant professional criteria")
    else:
        print("Invalid choice!")
```

OUTPUT:

```
======================================================================
TASK 4: Ethical Job Applicant Scoring System
======================================================================

Test [1] Biased Scoring or [2] Fair Scoring? (1/2): 1

--- BIASED SCORING SYSTEM ---
Enter applicant name: Kruthikroshan
Enter gender (male/female): male
Enter skills rating (0-10): 10
Enter years of experience: 8
Enter education level (1-4): 4

✗ Total Score: 350
BIAS DETECTED: Gender and name influence score!
```

Justification:
This task compares biased and fair scoring models to show that ethical AI must evaluate individuals based only on relevant qualifications.

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

PROMPT: This task demonstrates inclusive and respectful software design.Learn how to create gender-neutral systems that respect all users equally.See how fair compensation should be based on role and experience, not identity.

CODE:

```
# TASK 5: Inclusiveness and Ethical Variable Design
# ================================================================================
# PROMPT: This task demonstrates inclusive and respectful software design.
# Learn how to create gender-neutral systems that respect all users equally.
# See how fair compensation should be based on role and experience, not identity.
# ================================================================================

def calculate_salary(position, experience):
    """Fair salary calculation based on role and experience"""
    base_salaries = {
        "Developer": 45000,
        "Senior Developer": 65000,
        "Lead Developer": 85000
    }
    base = base_salaries.get(position, 40000)
    experience_bonus = experience * 2000
    return base + experience_bonus

def task5_demo():
    print("\n" + "=" * 70)
    print("TASK 5: Inclusive and Ethical Variable Design")
    print("=" * 70)

    print("\nNON-INCLUSIVE VERSION:")
    print("❌ Separates employees by gender (male_employees, female_employees)")
    print("❌ Different salaries based on gender")
    print("❌ Assumes titles (Mr./Ms.)")

    print("\nINCLUSIVE VERSION:")
    print("✓ Gender-neutral variables")
    print("✓ Salary based on position and experience only")
    print("✓ Optional user-specified titles")

    print("\n--- ADD EMPLOYEE (INCLUSIVE SYSTEM) ---")
    name = input("Enter employee name: ")
    position = input("Enter position (Developer/Senior Developer/Lead Developer): ")
    experience = int(input("Enter years of experience: "))
    title = input("Enter preferred title (Mr./Ms./Mx./Dr. or leave blank): ").strip()
```

```
print("\n--- ADD EMPLOYEE (INCLUSIVE SYSTEM) ---")
name = input("Enter employee name: ")
position = input("Enter position (Developer/Senior Developer/Lead Developer): ")
experience = int(input("Enter years of experience: "))
title = input("Enter preferred title (Mr./Ms./Mx./Dr. or leave blank): ").strip()

salary = calculate_salary(position, experience)

print("\n--- EMPLOYEE INFORMATION ---")
if title:
    print(f"Name: {title} {name}")
else:
    print(f"Name: {name}")
print(f"Position: {position}")
print(f"Experience: {experience} years")
print(f"Salary: ${salary:,}")

print("\n✓ This system is inclusive and fair!")
print("✓ No gender assumptions or bias")
```

OUTPUT:

```
======================================================================
TASK 5: Inclusive and Ethical Variable Design
======================================================================

NON-INCLUSIVE VERSION:
❌ Separates employees by gender (male_employees, female_employees)
❌ Different salaries based on gender
❌ Assumes titles (Mr./Ms.)

INCLUSIVE VERSION:
✓ Gender-neutral variables
✓ Salary based on position and experience only
✓ Optional user-specified titles

Enter years of experience: 5
Enter preferred title (Mr./Ms./Mx./Dr. or leave blank): Mr

--- EMPLOYEE INFORMATION ---
Name: Mr kruthikroshan
Position: developer
Experience: 5 years
Salary: $50,000

✓ This system is inclusive and fair!
✓ No gender assumptions or bias
```

Justification:
This task highlights inclusive and gender-neutral design, ensuring fairness and respect for all users in AI-based systems.