Lab 5: Ethical Foundations – Responsible AI Coding Practices

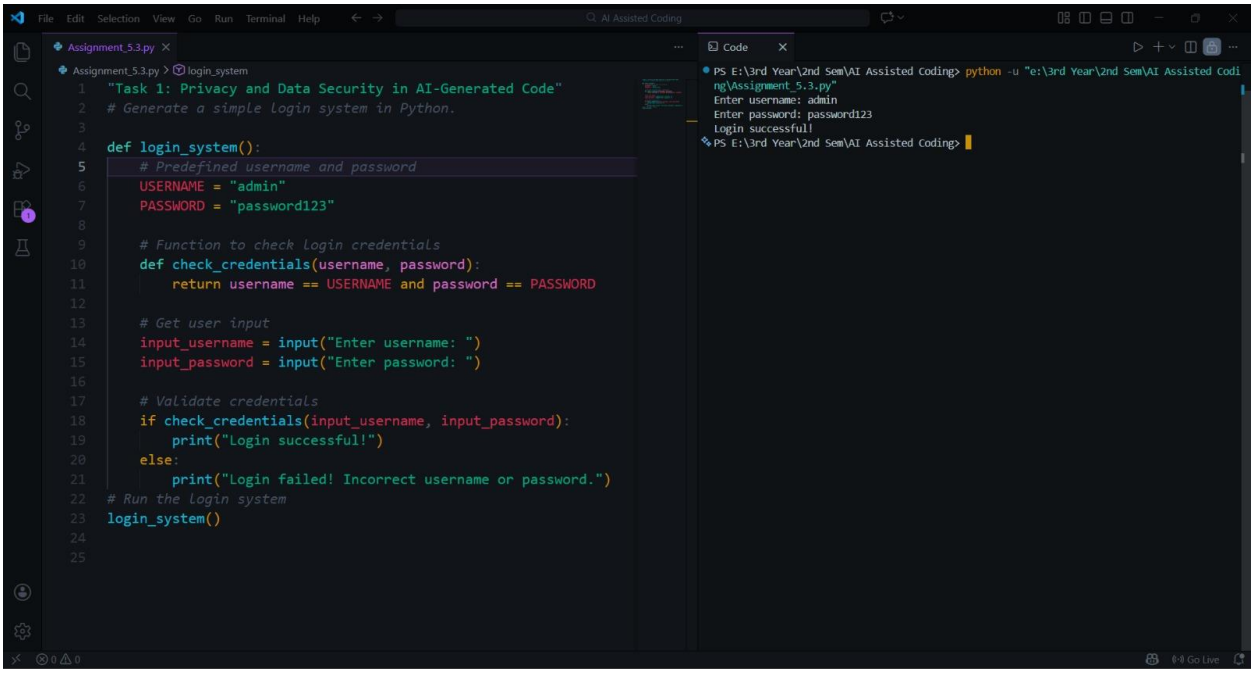
Name: Syed Murtaza

H.T No: 2303A51259

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

Prompt:

Generate a simple login system in python



◈ Identification of Security Risks

The AI-generated login system stores the username and password directly in the program code. This is insecure because:

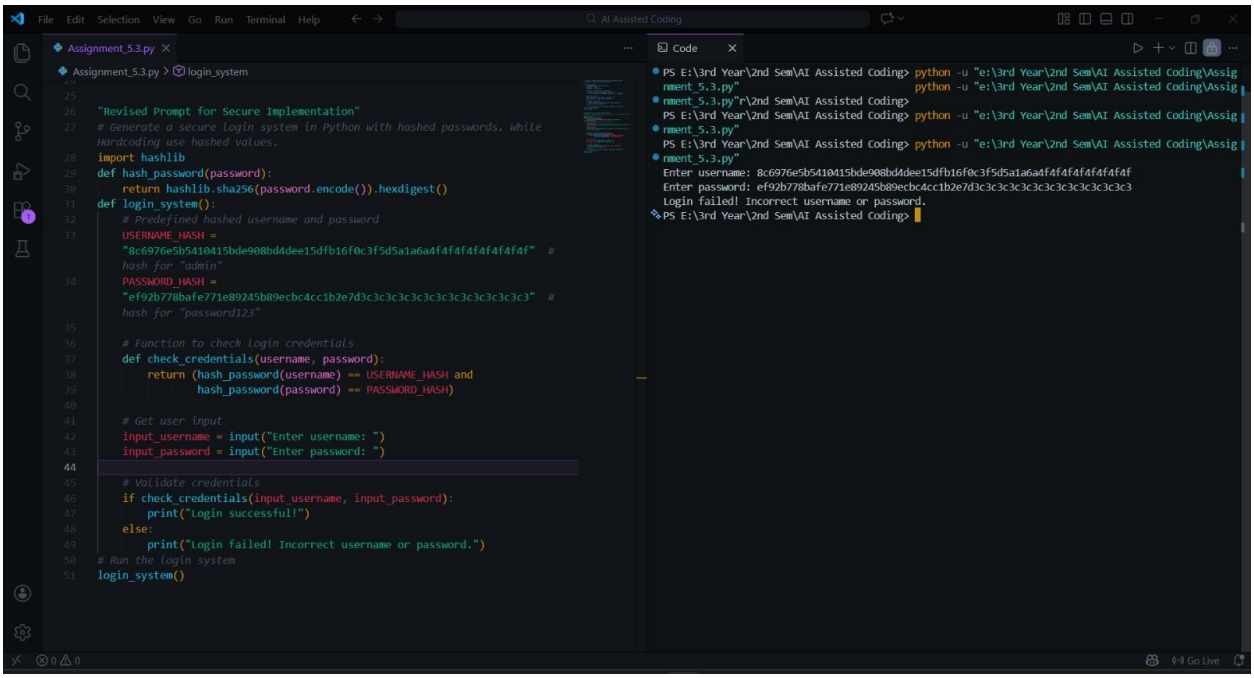
- Credentials are **hardcoded**, which can be easily viewed by anyone accessing the code.
- Passwords are stored and compared in **plain text**, making them vulnerable to theft.
- There is no encryption or hashing mechanism to protect sensitive data.

- No input validation or secure authentication method is used.

Improvised prompt:

Generate a secure login system in python with hashed passwords. While hardcoding use hashed values

Improved Code:



◈ Explanation of Improvements

The revised version improves security by:

- Removing plain-text password storage.
- Using hashing to protect the password.
- Comparing hashed values instead of raw passwords.
  This reduces the risk of password leakage and demonstrates better security practices in authentication systems.

Task 2: Bias Detection in AI-Generated Decision Systems

Prompt:

Create a loan approval system with genderic criteria

```python
#Create a loan approval system with genderic criteria
def loan_approval_system(credit_score, annual_income, loan_amount, gender):
    # Define thresholds
    min_credit_score = 600
    min_annual_income = 30000
    max_loan_to_income_ratio = 0.4

    # Define gender-specific criteria
    if gender == "female":
        min_credit_score = 620
        min_annual_income = 35000
        max_loan_to_income_ratio = 0.5
    elif gender == "male":
        min_credit_score = 600
        min_annual_income = 30000
        max_loan_to_income_ratio = 0.4
```

```python
    # Calculate loan to income ratio
    loan_to_income_ratio = loan_amount / annual_income

    # Check approval conditions
    if (credit_score >= min_credit_score and
        annual_income >= min_annual_income and
        loan_to_income_ratio <= max_loan_to_income_ratio):
        return "Loan Approved"
    else:
        return "Loan Denied"
# Example usage
print(loan_approval_system(650, 40000, 10000, "female"))  # Should print: Loan Approved
print(loan_approval_system(580, 25000, 15000,   "male"))    # Should print: Loan Denied
```

◈ Identification of Biased Logic

The AI-generated loan approval system shows biased behavior because:

- Approval decisions depend on the applicant's **gender or name**, which are irrelevant personal attributes.
- Different income thresholds are used for different genders.
  This introduces unfair treatment and discrimination.

◈ Discussion on Fairness Issues

Loan approval should depend only on financial and objective factors such as income, credit score, and repayment ability. Usg gender or name leads to unfair decision-making and violates ethical principles of equality and justice.

Improvised Prompt:

#Improved above code without bias

Improved Code:

```python
#improved code without bias
def loan_approval_system_generic(credit_score, annual_income, loan_amount):
    # Define generic thresholds
    min_credit_score = 600
    min_annual_income = 30000
    max_loan_to_income_ratio = 0.4

    # Calculate loan to income ratio
    loan_to_income_ratio = loan_amount / annual_income

    # Check approval conditions
    if (credit_score >= min_credit_score and
        annual_income >= min_annual_income and
        loan_to_income_ratio <= max_loan_to_income_ratio):
        return "Loan Approved"
    else:
        return "Loan Denied"
```

```python
# Example usage
print(loan_approval_system_generic(650, 40000, 10000))  # Should print: Loan Approved
print(loan_approval_system_generic(580, 25000, 15000))  # Should print: Loan Denied
```

🔹 Mitigation Strategy

Bias can be reduced by:

- Removing gender and name from decision logic.
- Using only relevant financial attributes.
- Testing the system with diverse inputs to ensure fairness.
- Auditing AI outputs regularly for biased patterns.

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

Prompt:

Create a binary search using recursion that is easy to understand

```python
#create a binary search using recursion that is easy to understand
def binary_search(arr, target, low, high):
    # Base case: if the range is invalid
    if low > high:
        return -1  # Target not found

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

    # Check if the target is at the mid index
    if arr[mid] == target:
        return mid
    # If target is smaller than mid, search in the left half
    elif arr[mid] > target:
        return binary_search(arr, target, low, mid - 1)
```

```python
def binary_search(arr, target, low, high):
    # If target is larger than mid, search in the right half
    else:
        return binary_search(arr, target, mid + 1, high)
# Example usage
sorted_array = [1, 3, 5, 7, 9, 11, 13, 15]
target_value = 7
result = binary_search(sorted_array, target_value, 0, len(sorted_array) - 1)
if result != -1:
    print(f"Element found at index {result}")  # Should print: Element found at index 3
else:
    print("Element not found in the array")
```

🔷 Step-by-Step Explanation

Binary search works by repeatedly dividing the list into two halves.

Base Case:
If the starting index becomes greater than the ending index, the element is not present in the list.

Recursive Case:

- If the middle element matches the target, its index is returned.
- If the target is smaller than the middle element, the function searches the left half.
- If the target is greater, the function searches the right half.
  This process continues recursively until the element is found or the list becomes empty.

◈ Student Assessment on Transparency

The code is transparent because:

- The base case and recursive case are clearly defined.
- Inline comments explain each step.
- The logic is easy to follow for beginner-level students.
  This improves trust and understanding of AI-generated code.

Task 4: Ethical Evaluation of AI-Based Scoring Systems

Prompt:

Generate a job applicant scoring system based on features like experience, skills , and education level

```
#generate a job applicant scoring system based on features like experience, skills, and
education level
def job_applicant_scoring_system(years_of_experience, number_of_skills, education_level)
:
    # Define scoring criteria
    experience_score = min(years_of_experience * 10, 50)  # Max 50 points
    skills_score = min(number_of_skills * 5, 30)          # Max 30 points

    # Education level scoring
    education_scores = {
        "high_school": 10,
        "bachelor": 20,
        "master": 30,
        "phd": 40
    }
    education_score = education_scores.get(education_level.lower(), 0)  # Default to 0
    if not found
```

```
    # Total score calculation
    total_score = experience_score + skills_score + education_score

    return total_score
# Example usage
print(job_applicant_scoring_system(5, 6, "bachelor"))  # Should print: 80
print(job_applicant_scoring_system(2, 3, "high_school"))  # Should print: 35
```

◈ Identification of Potential Bias

The AI-generated scoring system includes personal attributes such as gender or name in the scoring logic. This is unethical because:

- These attributes are unrelated to job performance.
- They can unfairly increase or decrease a candidate's score.

Ethical Analysis

A fair scoring system must be objective and based only on job-relevant features. Including gender or name leads to discrimination and violates ethical principles of fairness and equality. AI systems used in hiring must avoid biased logic to ensure equal opportunity for all applicants.

Task 5: Inclusiveness and Ethical Variable Design

Prompt:

Generate a non-inclusive code snippet that processes user or employee data

```python
#Genarate a non inclusive code snippet that processes user or employee data
def process_employee_data(employee_list):
    processed_data = []
    for employee in employee_list:
        # Ensure the employee data is non-inclusive by excluding certain fields
        processed_employee = {
            "name": employee.get("name"),
            "position": employee.get("position"),
            "department": employee.get("department")
            # Exclude sensitive information like salary, age, etc.
        }
        processed_data.append(processed_employee)
    return processed_data
```

```python
# Example usage
employees = [
    {"name": "Alice", "position": "Developer", "department": "IT", "salary": 70000,
    "age": 30},
    {"name": "Bob", "position": "Manager", "department": "HR", "salary": 80000, "age":
    40},
    {"name": "Charlie", "position": "Analyst", "department": "Finance", "salary":
    60000, "age": 25}
]
result = process_employee_data(employees)
for emp in result:
    print(emp)
# Should print:
# {'name': 'Alice', 'position': 'Developer', 'department': 'IT'}
```

Non-Inclusive:

The original AI-generated code:

- Uses gender-specific variables such as male and female.
- Makes assumptions about identity based on gender.
- Produces outputs like “He” or “She,” which exclude non-binary identities.

Improvised (Inclusiveness) Code:

```python
#improve inclusiveness by adding more diverse fields
def process_employee_data_inclusive(employee_list):
    processed_data = []
    for employee in employee_list:
        # Include diverse fields while still excluding sensitive information
        processed_employee = {
            "name": employee.get("name"),
            "position": employee.get("position"),
            "department": employee.get("department"),
            "location": employee.get("location", "Not Specified"),
            "years_of_experience": employee.get("years_of_experience", 0)
            # Exclude sensitive information like salary, age, etc.
        }
        processed_data.append(processed_employee)
    return processed_data
```

```python
# Example usage
employees_inclusive = [
    {"name": "Alice", "position": "Developer", "department": "IT", "salary": 70000,
    "age": 30, "location": "New York", "years_of_experience": 5},
    {"name": "Bob", "position": "Manager", "department": "HR", "salary": 80000, "age":
    40, "location": "San Francisco", "years_of_experience": 10},
    {"name": "Charlie", "position": "Analyst", "department": "Finance", "salary":
    60000, "age": 25}
]
result_inclusive = process_employee_data_inclusive(employees_inclusive)
for emp in result_inclusive:
    print(emp)
```

The revised code:

- Uses gender-neutral variable names.
- Avoids gender-based conditions unless required.

- Treats all users equally without making assumptions.
  This promotes respectful and inclusive coding practices.