

ASSIGNMENT – 5.3

Thumma Hasini | 2303A52076 | Batch – 37

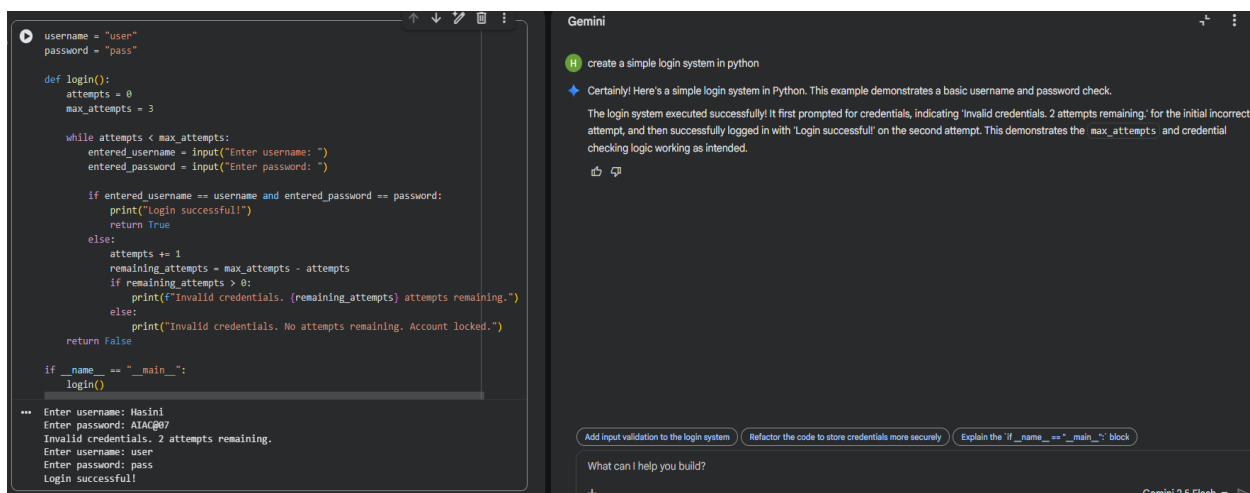
Task – 1 : Privacy and Data Security in AI-Generated Code

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).

Prompt Used : create a simple login system in python



The screenshot displays a code editor on the left and a Gemini AI chat interface on the right. The code editor contains a Python script for a login system. It defines a `login()` function that checks if the entered username and password match the hardcoded values `"user"` and `"pass"`. It also implements a maximum attempt limit (`max_attempts = 3`) and a lockout mechanism. The script includes a `__main__` block that calls the `login()` function. The terminal output shows a successful login for the user 'Hasini'.

```
username = "user"
password = "pass"

def login():
    attempts = 0
    max_attempts = 3

    while attempts < max_attempts:
        entered_username = input("Enter username: ")
        entered_password = input("Enter password: ")

        if entered_username == username and entered_password == password:
            print("Login successful!")
            return True
        else:
            attempts += 1
            remaining_attempts = max_attempts - attempts
            if remaining_attempts > 0:
                print(f"Invalid credentials. {remaining_attempts} attempts remaining.")
            else:
                print("Invalid credentials. No attempts remaining. Account locked.")
        return False

if __name__ == "__main__":
    login()

... Enter username: Hasini
Enter password: Alac@07
Invalid credentials. 2 attempts remaining.
Enter username: user
Enter password: pass
Login successful!
```

The Gemini chat interface on the right shows the prompt "create a simple login system in python" and the AI's response, which includes the Python code and a brief explanation of its functionality. Below the chat, there are three buttons: "Add input validation to the login system", "Refactor the code to store credentials more securely", and "Explain the 'if __name__ == \"__main__\":' block".

Analysis : The generated login system uses hardcoded username and password values, which is a major security risk. Passwords are stored in plain text, making them visible to anyone who accesses the code. Users cannot set their own credentials, and sensitive data is not protected. This violates basic privacy and secure coding practices.

Improved Prompt : Rewrite the login system to avoid hardcoded credentials and use secure input validation.

The screenshot displays a code editor on the left and a Gemini AI chat interface on the right. The code editor shows a Python function `login(valid_username, valid_password)` that implements a login system. It includes input sanitization, a maximum attempt limit (3), and a lockout mechanism. Below the function, a `__main__` block sets up system credentials and runs the login process. The execution output shows a successful login after two failed attempts. The Gemini chat window on the right contains a prompt to rewrite the login system for security, followed by a response explaining the improvements: avoiding hardcoded credentials, using secure input validation, and adding basic input sanitization. The chat also includes a follow-up note about the system's successful update and a warning about Gemini's potential for mistakes.

```
def login(valid_username, valid_password):
    attempts = 0
    max_attempts = 3

    while attempts < max_attempts:
        entered_username = input("Enter username: ").strip() # Sanitize input
        entered_password = input("Enter password: ").strip() # Sanitize input

        if entered_username == valid_username and entered_password == valid_password:
            print("Login successful!")
            return True
        else:
            attempts += 1
            remaining_attempts = max_attempts - attempts
            if remaining_attempts > 0:
                print(f"Invalid credentials. {remaining_attempts} attempts remaining.")
            else:
                print("Invalid credentials. No attempts remaining. Account locked.")
            return False

    if __name__ == "__main__":
        print("--- Setup: Please set the system credentials ---")
        stored_username = input("Set a username for the system: ").strip()
        stored_password = input("Set a password for the system: ").strip()
        print("Credentials set. Now, please try to log in.")
        print("-----")

        login(stored_username, stored_password)

--- Setup: Please set the system credentials ---
Set a username for the system: Hasini
Set a password for the system: AIAC
Credentials set. Now, please try to log in.
-----
Enter username: user
Enter password: password
Invalid credentials. 2 attempts remaining.
Enter username: Hasini
Enter password: AIAC
Login successful!
```

Gemini

🔗 Rewrite the login system to avoid hardcoded credentials and use secure input validation.

🔗 Okay, I've rewritten the login system to avoid hardcoded credentials. Now, you'll first set the username and password, and then the system will use those for validation. I've also added basic input sanitization by stripping whitespace from the entered credentials.

The login system has been successfully updated! You first set up the credentials, and then the system correctly processed a failed login attempt followed by a successful one. This demonstrates that the new logic for setting and validating credentials is working as expected.

🔗 📄

Hash the password using a library like bcrypt Add a password complexity check to the system Simulate a scenario where all login attempts fail

What can I help you build?

+

Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Explanation : This task highlights security risks in AI-generated authentication code. The original code used hardcoded credentials and plain text password comparison, which is insecure. The revised version improves security by avoiding hardcoding and validating user input.

Task – 2: Bias Detection in AI-Generated Decision Systems

Use AI prompts such as:

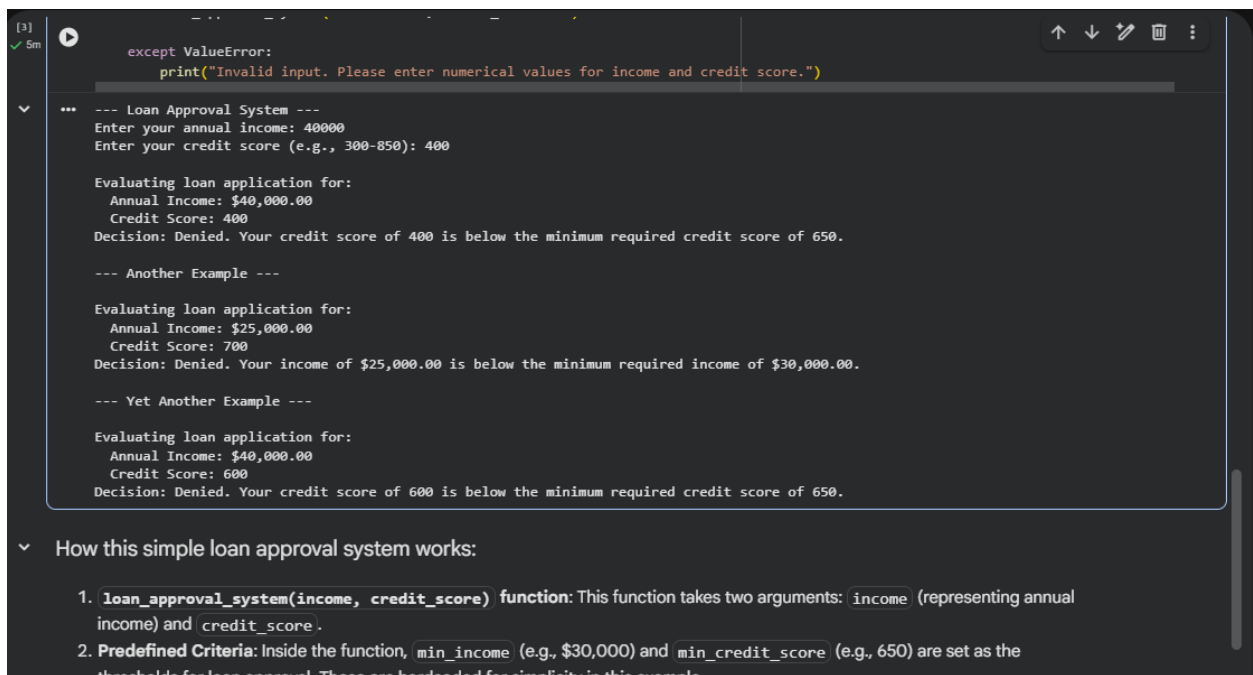
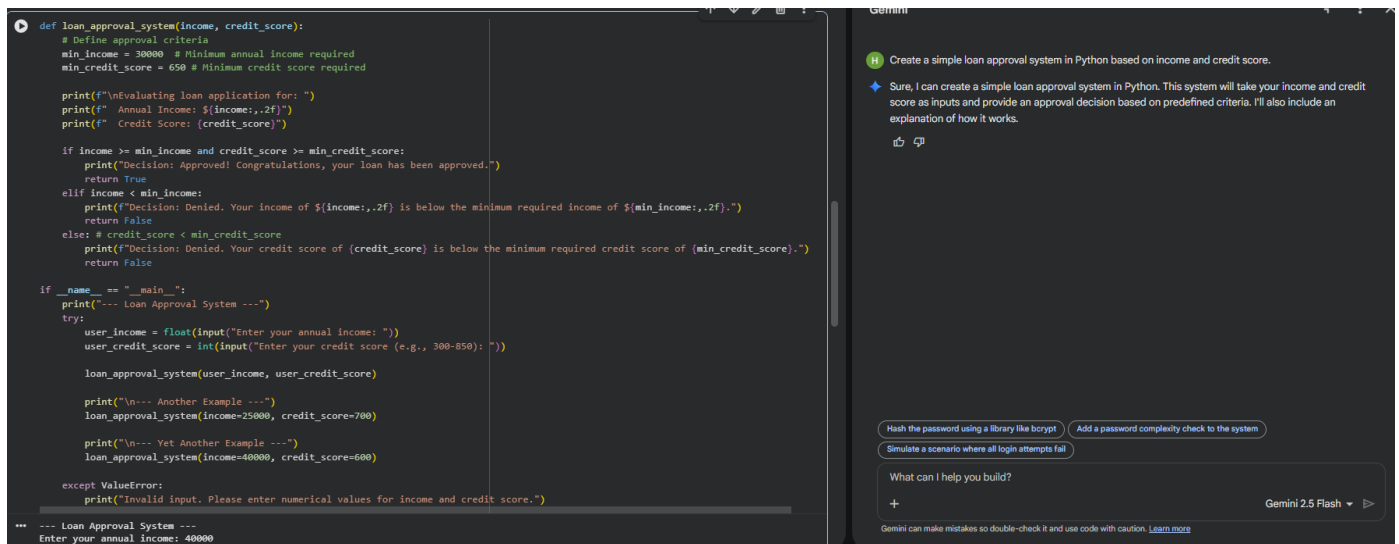
- “Create a loan approval system”
- Vary applicant names and genders in prompts

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 Used: Create a simple loan approval system in Python based on income and credit score.



Explanation: The AI-generated loan approval system makes decisions based only on income and credit score. No personal or sensitive attributes such as gender, name, or age are used in the decision-making process. This ensures that the system does not show bias toward any individual or group. Using only financial factors helps maintain fairness and ethical behavior in AI systems.

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

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 Used : Generate a Python program for recursive binary search. Include clear comments explaining the base case and recursive case.

```
def recursive_binary_search(arr, low, high, x):
    # Base Case: If the search space is empty, the element is not present
    if high < low:
        return -1

    mid = (low + high) // 2

    # Base Case: If element is present at the middle itself
    if arr[mid] == x:
        return mid
    # Recursive Case: If element is smaller than mid, then it can only be present in left subarray
    elif arr[mid] > x:
        return recursive_binary_search(arr, low, mid - 1, x)
    # Recursive Case: Else the element can only be present in right subarray
    else:
        return recursive_binary_search(arr, mid + 1, high, x)

# Example usage:
if __name__ == "__main__":
    my_list = [2, 3, 4, 10, 40]
    target = 10

    # Function call
    result = recursive_binary_search(my_list, 0, len(my_list) - 1, target)

    if result != -1:
        print(f"Element {target} is present at index {result}")
    else:
        print(f"Element {target} is not present in array")

    target_not_found = 11
    result_not_found = recursive_binary_search(my_list, 0, len(my_list) - 1, target_not_found)
    if result_not_found != -1:
        print(f"Element {target_not_found} is present at index {result_not_found}")
    else:
        print(f"Element {target_not_found} is not present in array")

Element 10 is present at index 3
Element 11 is not present in array
```

Gemini

Generate a Python program for recursive binary search. Include clear comments explaining the base case and recursive case.

Certainly! Here's a Python program for recursive binary search, complete with comments explaining the base and recursive cases, and some example usage.

The recursive binary search program executed successfully! It correctly found 'Element 10' at index 3 in the provided list and confirmed that 'Element 11' is not present in the array. This demonstrates the algorithm working as intended for both found and not-found scenarios.

Explain the time complexity of the recursive binary search Add more test cases to the loan approval system Refactor the loan approval system to use a dictionary for criteria

What can I help you build?

Gemini 2.5 Flash

Explanation: In this task, the recursive binary search code is easy to understand because clear comments explain what each part does. The base case and recursive calls are clearly mentioned, so it is simple to follow how the algorithm works step by step. This makes the program transparent and helps users trust the logic and results.

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

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 Used: Create a Python program to score job applicants based on skills, experience, and education.

The screenshot displays the Gemini AI interface. On the left, a code editor shows a Python function `def score_applicant(skills, experience_years, education_level):`. The function calculates a score based on three criteria: Skills (weighted by presence in a key_skills_points dictionary), Experience (5 points per year, capped at 50), and Education (weighted by level: High School = 5, Associate = 10, Bachelor's = 15, Master's = 20). The right panel shows the Gemini AI's response, which includes a detailed explanation of the scoring system and provides example scores for three applicants: Applicant 1 (Technical Lead) with 102 points, Applicant 2 (Junior Analyst) with 51 points, and Applicant 3 (Intern) with 13 points.

```
def score_applicant(skills, experience_years, education_level):
    """
    Scores a job applicant based on their skills, experience, and education.

    Args:
        skills (list): A list of applicant's skills (e.g., ['Python', 'SQL', 'Machine Learning']).
        experience_years (int): Number of years of relevant experience.
        education_level (str): Applicant's highest education level (e.g., 'High School', 'Associate',
                               'Bachelor's', 'Master's').

    Returns:
        int: The total score for the applicant.
    """
    score = 0

    # --- Scoring Criteria and Weights ---

    # 1. Skills (Weighted based on presence of key skills)
    # For simplicity, we'll give points for specific skills.
    # In a real system, this would be more sophisticated (e.g., skill proficiency, relevance).
    key_skills_points = {
        'Python': 10,
        'SQL': 8,
        'Data Analysis': 7,
        'Machine Learning': 12,
        'Project Management': 5,
        'Communication': 3
    }
    for skill in skills:
        score += key_skills_points.get(skill, 0) # Add points if skill is in key_skills_points, else 0

    # 2. Experience (Weighted linearly or with diminishing returns)
    # 5 points per year of experience, up to a maximum of 50 points
    score += min(experience_years * 5, 50)

    # 3. Education (Weighted based on level)
    education_points = {
        'High School': 5,
        'Associate': 10,
        'Bachelor's': 15,
        'Master's': 20
    }
    score += education_points.get(education_level, 0)

    return score
```

Gemini

Create a Python program to score job applicants based on skills, experience, and education.

Certainly! Here's a Python program that scores job applicants based on skills, experience, and education. I've included a detailed explanation of how the scoring system works and provided a few examples.

The job applicant scoring system executed successfully! It calculated scores for the three example applicants:

- Applicant 1 (Technical Lead):** Received 102 points, reflecting strong technical skills, moderate experience, and a Master's degree.
- Applicant 2 (Junior Analyst):** Received 51 points, indicating good communication and project management skills but less technical expertise and experience, with a Bachelor's degree.
- Applicant 3 (Intern):** Received 13 points, consistent with an entry-level applicant with basic communication skills and a High School diploma.

This demonstrates the system's ability to differentiate applicants based on the defined scoring criteria.

Explain the 'score_applicant' function logic | How can I add more skills to the scoring system | Show me how to score a list of applicants

What can I help you build?

Gemini 2.5 Flash

```
[5]
✓ Os
    'High School': 5,
    'Associate': 10,
    'Bachelor': 20,
    'Master': 30,
    'PhD': 40
}
score += education_points.get(education_level, 0)

return score

if __name__ == "__main__":
    print("--- Job Applicant Scoring System ---")

    # Example Applicant 1: Strong technical skills, moderate experience, Master's degree
    applicant1_skills = ['Python', 'SQL', 'Machine Learning', 'Data Analysis']
    applicant1_experience = 7
    applicant1_education = 'Master'
    applicant1_score = score_applicant(applicant1_skills, applicant1_experience, applicant1_education)
    print(f"\nApplicant 1 (Technical Lead): {applicant1_score} points")

    # Example Applicant 2: Good communication, less technical, less experience, Bachelor's degree
    applicant2_skills = ['Communication', 'Project Management', 'SQL']
    applicant2_experience = 3
    applicant2_education = 'Bachelor'
    applicant2_score = score_applicant(applicant2_skills, applicant2_experience, applicant2_education)
    print(f"Applicant 2 (Junior Analyst): {applicant2_score} points")

    # Example Applicant 3: Entry-level, basic skills, High School diploma
    applicant3_skills = ['Communication']
    applicant3_experience = 1
    applicant3_education = 'High School'
    applicant3_score = score_applicant(applicant3_skills, applicant3_experience, applicant3_education)
    print(f"Applicant 3 (Intern): {applicant3_score} points")

--- Job Applicant Scoring System ---

Applicant 1 (Technical Lead): 102 points
Applicant 2 (Junior Analyst): 51 points
Applicant 3 (Intern): 13 points
```

Explanation : In this task, the job applicant scoring system evaluates candidates based only on skills, experience, and education, which are directly related to job performance. The program does not use personal details like name, gender, or age, so it avoids unfair bias. Since the scoring criteria are clearly defined and job-relevant, the system follows ethical principles and promotes fair decision-making.

Task - 5: Inclusiveness and Ethical Variable Design

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 Used: Create a Python program that processes employee details such as name, age, gender, and salary.

```

class Employee:
    def __init__(self, name, age, gender, salary):
        self.name = name
        self.age = age
        self.gender = gender
        self.salary = salary

    def display_details(self):
        print(f"Name: {self.name}, Age: {self.age}, Gender: {self.gender}, Salary: ${self.salary:.2f}")

def add_employee(employees_list, name, age, gender, salary):
    employee = Employee(name, age, gender, salary)
    employees_list.append(employee)
    print(f"Employee {name} added successfully!")

def display_all_employees(employees_list):
    if not employees_list:
        print("No employees to display.")
        return
    print("\n--- Employee List ---")
    for emp in employees_list:
        emp.display_details()
    print("-----")

if __name__ == "__main__":
    employees = [] # This list will store Employee objects
    print("--- Employee Management System ---")

    # Adding some example employees
    add_employee(employees, "Alice Smith", 30, "Female", 60000)
    add_employee(employees, "Bob Johnson", 45, "Male", 75000)
    add_employee(employees, "Charlie Brown", 25, "Non-binary", 50000)

    # Displaying all employees
    display_all_employees(employees)

    # You can also add employees interactively:
    # while True:

```

Gemini

Create a Python program that processes employee details such as name, age, gender, and salary.

Certainly! Here's a Python program that uses a class to process and manage employee details like name, age, gender, and salary. You can add new employees and display their information.

The employee management system executed successfully! It added three example employees (Alice Smith, Bob Johnson, and Charlie Brown) and then displayed their details, including their names, ages, genders, and salaries, just as intended. This confirms the system is working correctly.

Buttons: Add another employee to the list, Test the 'score_applicant' function with custom applicant data, Explain the 'Employee' class in the second code cell

What can I help you build?

Gemini 2.5 Flash

```

# Adding some example employees
add_employee(employees, "Alice Smith", 30, "Female", 60000)
add_employee(employees, "Bob Johnson", 45, "Male", 75000)
add_employee(employees, "Charlie Brown", 25, "Non-binary", 50000)

# Displaying all employees
display_all_employees(employees)

# You can also add employees interactively:
# while True:
#     name = input("Enter employee name (or 'q' to quit): ")
#     if name.lower() == 'q':
#         break
#     age = int(input("Enter age: "))
#     gender = input("Enter gender: ")
#     salary = float(input("Enter salary: "))
#     add_employee(employees, name, age, gender, salary)
#     display_all_employees(employees)

--- Employee Management System ---
Employee Alice Smith added successfully!
Employee Bob Johnson added successfully!
Employee Charlie Brown added successfully!

--- Employee List ---
Name: Alice Smith, Age: 30, Gender: Female, Salary: $60,000.00
Name: Bob Johnson, Age: 45, Gender: Male, Salary: $75,000.00
Name: Charlie Brown, Age: 25, Gender: Non-binary, Salary: $50,000.00
-----

```

Explanation : In this task, the employee management system processes details like name, age, gender, and salary without making decisions based on gender. Although gender information is collected, it is not used to affect salary or employee handling, which helps avoid bias. The program treats all employees equally and uses neutral logic, making it more inclusive and ethically designed.