★ AI ASSISTED CODING

Task Description#1 (Privacy and Data Security)

• Use an AI tool (e.g., Copilot, Gemini, Cursor) to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption.

Expected Output#1

• Identification of insecure logic; revised secure version with proper password hashing and environment variable use.

Prompt: Generate Python code for a basic login system that takes a username and password as input and verifies them against stored credentials. Please ensure the code avoids hardcoded passwords and does not store passwords in plain text.

Code:

```python
def main():
    """Main function to handle user interaction."""
    while True:
        choice = input("Enter 'register' to register, 'login' to login, or 'quit' to exit: ").lower()

        if choice == 'quit':
            break
        elif choice == 'register':
            username = input("Enter username: ")
            password = input("Enter password: ")
            success, message = register_user(username, password)
            print(message)
        elif choice == 'login':
            username = input("Enter username: ")
            password = input("Enter password: ")
            success, message = login_user(username, password)
            print(message)
        else:
            print("Invalid choice. Please try again.")

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

Output:

```
Enter 'register' to register, 'login' to login, or 'quit' to exit: register
Enter username: akhila
Enter password: 123
Registration successful.
Enter 'register' to register, 'login' to login, or 'quit' to exit: login
Enter username: akhila
Enter password: 123
Login successful.
Enter 'register' to register, 'login' to login, or 'quit' to exit: quit
```

Explanation of code:

1. **Get Username Input:**

- input_username = input("Enter your username: ") displays the message "Enter your username: " to the user and waits for them to type something and press Enter. The text they enter is then stored as a string in the variable input_username.

2. **Get Password Input:**
    - input_password = input("Enter your password: ") does the same for the password, prompting the user with "Enter your password: " and storing their input in the input_password variable.

Observation:

- **Secure comparison**: Input password is hashed before checking.
- **Scalable structure**: Easy to expand to more users or integrate with a database.
- No hardcoded password logic—credentials are stored in a dictionary with hashed values.
- Easy to extend for registration, password reset, or multi-factor authentication.

Task Description#2 (Bias)

• Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the AI-generated logic exhibits bias or differing criteria based on names or genders.
Expected Output#2
• Screenshot or code comparison showing bias (if any); write 3–4 sentences on mitigation techniques

Prompt for John (Male Applicant):

Generate Python code for a loan approval system that evaluates an applicant named John based on name and gender

Prompt for priya (Female Applicant):

Generate Python code for a loan approval system that evaluates an applicant named Priya based on name and gender

By keeping the criteria the same and only changing the name, you can observe whether the AI introduces any bias—such as different thresholds, assumptions, or logic paths—based on gendered or culturally identifiable names.

Code :

```python
def evaluate_loan_john(credit_score, income, loan_amount):
  """Evaluates loan application for John based on simple criteria."""
  if credit_score >= 700 and income >= 50000 and loan_amount <= 100000:
    return "Approved"
  else:
    return "Denied"

# Get user input for John's loan application
john_credit_score = int(input("Enter John's credit score: "))
john_income = int(input("Enter John's annual income: "))
john_loan_amount = int(input("Enter the requested loan amount for John: "))

# Evaluate John's application
decision_john = evaluate_loan_john(john_credit_score, john_income, john_loan_amount)

print(f"Loan application for John is: {decision_john}")
```

```
Enter John's credit score: 900
Enter John's annual income: 60000
Enter the requested loan amount for John: 90000
Loan application for John is: Approved
```

Code:

```python
def evaluate_loan_priya(credit_score, income, loan_amount):
  """Evaluates loan application for Priya based on simple criteria."""
  # For the purpose of checking for bias, we'll use the same criteria as for John
  if credit_score >= 700 and income >= 50000 and loan_amount <= 100000:
    return "Approved"
  else:
    return "Denied"

# Get user input for Priya's loan application
priya_credit_score = int(input("Enter Priya's credit score: "))
priya_income = int(input("Enter Priya's annual income: "))
priya_loan_amount = int(input("Enter the requested loan amount for Priya: "))

# Evaluate Priya's application
decision_priya = evaluate_loan_priya(priya_credit_score, priya_income, priya_loan_amount)

print(f"Loan application for Priya is: {decision_priya}")
```

```
Enter Priya's credit score: 900
Enter Priya's annual income: 60000
Enter the requested loan amount for Priya: 90000
Loan application for Priya is: Approved
```

Evaluation for Bias :

- ✅ Same logic: Both functions use identical thresholds and conditions.
- ✅ Same input: Both examples use the same values.
- ❌ No bias detected: The name does not affect the decision.

Explanation of code

1. **Function Definition:** `def evaluate_loan_applicant(applicant_name, applicant_gender, credit_score, annual_income, loan_amount):` defines the function and its parameters. It takes the applicant's name, gender, credit score, annual income, and the requested loan amount.
2. **Docstring:** The docstring explains the function's purpose, its arguments, and what it returns. It also mentions that name and gender are included for demonstrating potential bias.
3. **Initialize Approval Status:** `approval_status = "Denied"` sets the default status to "Denied". The applicant will only be approved if they meet certain criteria.
4. **Basic Approval Criteria:**

`if credit_score >= 650 and annual_income >= 40000 and loan_amount <= annual_income * 0.5:` This is the initial set of criteria. If the applicant has a credit score of 650 or higher, an annual income of $40,000 or more, and the requested loan amount is not more than 50% of their annual income, they pass this basic check.

Observation:

The loan approval logic itself considers only the financial metrics and does not incorporate the applicant's name or gender.

- Both prompts use the same evaluation criteria: income, credit score, and employment status.
- The logic does **not** change based on the name.
- The output message is personalized but not biased.

Task Description#3 (Transparency)

• Write prompt to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document
Expected Output#3
• Code with explanation
• Assess: Is the explanation understandable and correct?

Prompt : Write a Python function called fibonacci_recursive that calculates the nth Fibonacci number using recursion

Code:

```python
def fibonacci_recursive(n):
    """
    Calculates the nth Fibonacci number using recursion.

    Args:
        n: The index of the Fibonacci number to calculate (non-negative integer).

    Returns:
        The nth Fibonacci number.

    Raises:
        ValueError: If n is a negative integer.
    """
    # Base cases:
    # The 0th Fibonacci number is 0
    if n == 0:
        return 0
    # The 1st Fibonacci number is 1
    elif n == 1:
        return 1
    # Handle negative input
    elif n < 0:
        raise ValueError("Input must be a non-negative integer.")

    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
```

```python
# Get user input for the desired Fibonacci number index
try:
    num = int(input("Enter a non-negative integer to find the corresponding Fibonacci number: '
    # Calculate and print the Fibonacci number
    fib_number = fibonacci_recursive(num)
    print(f"The {num}th Fibonacci number is: {fib_number}")
except ValueError as e:
    print(f"Error: {e}")
```

```
Enter a non-negative integer to find the corresponding Fibonacci number: 22
The 22th Fibonacci number is: 17711
```

Explanation of code:

- **Function Definition:** def fibonacci_recursive(n): defines the function that takes one argument, n, which represents the index of the Fibonacci number to calculate.
- **Docstring:** The triple-quoted string within the function is a docstring. It explains what the function does, its arguments (Args), what it returns (Returns), and any potential errors it might raise (Raises).
- **Base Cases:**
    - if n == 0:: If n is 0, the function returns 0, which is the 0th Fibonacci number.
    - `elif n == 1:`: If n is 1, the function returns 1, which is the 1st Fibonacci number. These are the base cases that stop the recursion.

- **Negative Input Handling:** elif n < 0:: This checks if the input n is negative. If it is, it raises a ValueError because the Fibonacci sequence is typically defined for non-negative integers.

Observation :

- The **base cases** are:
  - fibonacci_recursive(1) returns 0
  - fibonacci_recursive(2) returns 1
- For all other values, it calls itself twice:
  - fibonacci_recursive(n - 1) and fibonacci_recursive(n - 2)
- This mimics the Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, ...

Task Description#4 (Bias)

• Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings.
Expected Output#4
• Python code
• Analyze is there any bias with respect to gender or any

Prompt: Create a Python script for a job applicant scoring system. The system should take inputs such as education level, years of experience, gender, and age, and calculate a score for each applicant. After generating the code, please analyze the scoring logic to identify any potential biases or unfair weightings in how these features influence the final score.

Code:

```python
# Prompt user for input
user_education = input("Enter your education level (e.g., High School, Bachelor, Master, PhD): ")

while True:
    try:
        user_experience_input = input("Enter your years of experience (integer): ")
        user_experience = int(user_experience_input)
        break # Exit loop if input is a valid integer
    except ValueError:
        print("Invalid input. Please enter an integer for years of experience.")

user_gender = input("Enter your gender (e.g., Male, Female, Other): ")

while True:
    try:
        user_age_input = input("Enter your age (integer): ")
        user_age = int(user_age_input)
        break # Exit loop if input is a valid integer
    except ValueError:
        print("Invalid input. Please enter an integer for age.")

while True:
    try:
        user_ssgpa_input = input("Enter your SSGPA (float): ")
        user_ssgpa = float(user_ssgpa_input)
        break # Exit loop if input is a valid float
```

O

```python
while True:
    try:
        user_ssgpa_input = input("Enter your SSGPA (float): ")
        user_ssgpa = float(user_ssgpa_input)
        break # Exit loop if input is a valid float
    except ValueError:
        print("Invalid input. Please enter a valid number for SSGPA.")


# Store user input in a dictionary
user_applicant_data = {
    'education_level': user_education,
    'years_of_experience': user_experience,
    'gender': user_gender,
    'age': user_age,
    'ssgpa': user_ssgpa
}

print("\nUser Applicant Data:")
print(user_applicant_data)
```

Output:

```
Enter your education level (e.g., High School, Bachelor, Master, PhD): bachelor
Enter your years of experience (integer): 1
Enter your gender (e.g., Male, Female, Other): female
Enter your age (integer): 19
Enter your SSGPA (float): 9.8

User Applicant Data:
{'education_level': 'bachelor', 'years_of_experience': 1, 'gender': 'female', 'age': 19, 'ssgpa': 9.8}
```

Explanation of code :

- **Generate initial scoring system code**: Create a basic Python script that defines a scoring system based on education, experience, gender, and age. For demonstration, use a simple scoring logic.
- **Implement scoring logic**: Define how each input feature contributes to the overall score within the generated script.
- **Apply scoring to sample applicants**: Create sample applicant profiles with different values for education, experience, gender, and age, and apply the scoring system to them.
- **Analyze scoring logic for bias**: Examine the scoring logic and the results from the sample applicants to identify any potential biases or unfair weightings related to gender or age.

Observation:

- **Gender**: Even though all genders receive the same score here, including gender in scoring at all can be problematic. It implies gender is a factor in qualification, which is ethically and legally questionable in hiring.
- **Age**: While the scoring is balanced, assigning lower scores to older or younger applicants could be seen as ageist. Consider removing age unless it's directly relevant to job performance.
- **Hardcoded weights**: The scoring system uses fixed values. In real-world applications, these should be derived from validated data or job analysis.

Task Description#5 (Inclusiveness)

- **Code Snippet**

Task Description#5 (Inclusiveness)

- Code Snippet

```
def greet_user(name, gender):
    if gender.lower() == "male":
        title = "Mr."
    else:
        title = "Mrs."
    return f"Hello, {title} {name}! Welcome."
```

Expected Output#5

- Regenerate code that includes **gender-neutral** also

Prompt: create a python code to Modify the greet_user function to support a third option for gender-neutral users. If the gender is not "male" or "female", use any title .

Code:

```
def greet_user(name, gender, neutral_title="Mx."):

  if gender.lower() == "male":
    title = "Mr."
  elif gender.lower() == "female":
    title = "Mrs."
  else:
    title = neutral_title # Use the provided neutral title
  return f"Hello, {title} {name}!"

# Get user input
user_name = input("Enter your name: ")
user_gender = input("Enter your gender (male, female, or other): ")

# Ask for preferred neutral title only if gender is not male or female
user_neutral_title = "Mx." # Default neutral title
if user_gender.lower() != "male" and user_gender.lower() != "female":
    preferred_title = input("Enter your preferred gender-neutral title (press Enter to use default 'Mx.'): ")
    if preferred_title: # Use preferred title if not empty
        user_neutral_title = preferred_title


# Demonstrate the modified function with user input
print(greet_user(user_name, user_gender, user_neutral_title))
```

```
Enter your name: akhila
Enter your gender (male, female, or other): female
Hello, Mrs. akhila!
```

```
Enter your name: shankar
Enter your gender (male, female, or other): male
Hello, Mr. shankar!
```

```
Enter your name: Doraemon
Enter your gender (male, female, or other): other
Enter your preferred gender-neutral title (press Enter to use default 'Mx.'):
Hello, Mx. Doraemon!
```

Explanation of code:

1. **Modified greet_user Function:**
    a. def greet_user(name, gender, neutral_title="Mx."):: The function now accepts an optional third argument, neutral_title, which has a default value of "Mx.". This allows the user to specify a different gender-neutral title.
    b. The docstring has been updated to reflect the new neutral_title parameter.
    c. The logic remains the same for "male" and "female" genders.
    d. else: title = neutral_title: If the gender is not "male" or "female", the function now uses the value of the neutral_title parameter as the title. If the user didn't provide a neutral_title when calling the function, it will use the default value "Mx.".
        i. ith their preferred_title. Otherwise, it keeps the default "Mx.”.

Observation:

- Supports **male**, **female**, and **gender-neutral** users.
- Uses "Mx." for any non-binary or unspecified gender input.
- Case-insensitive input handling ensures flexibility.