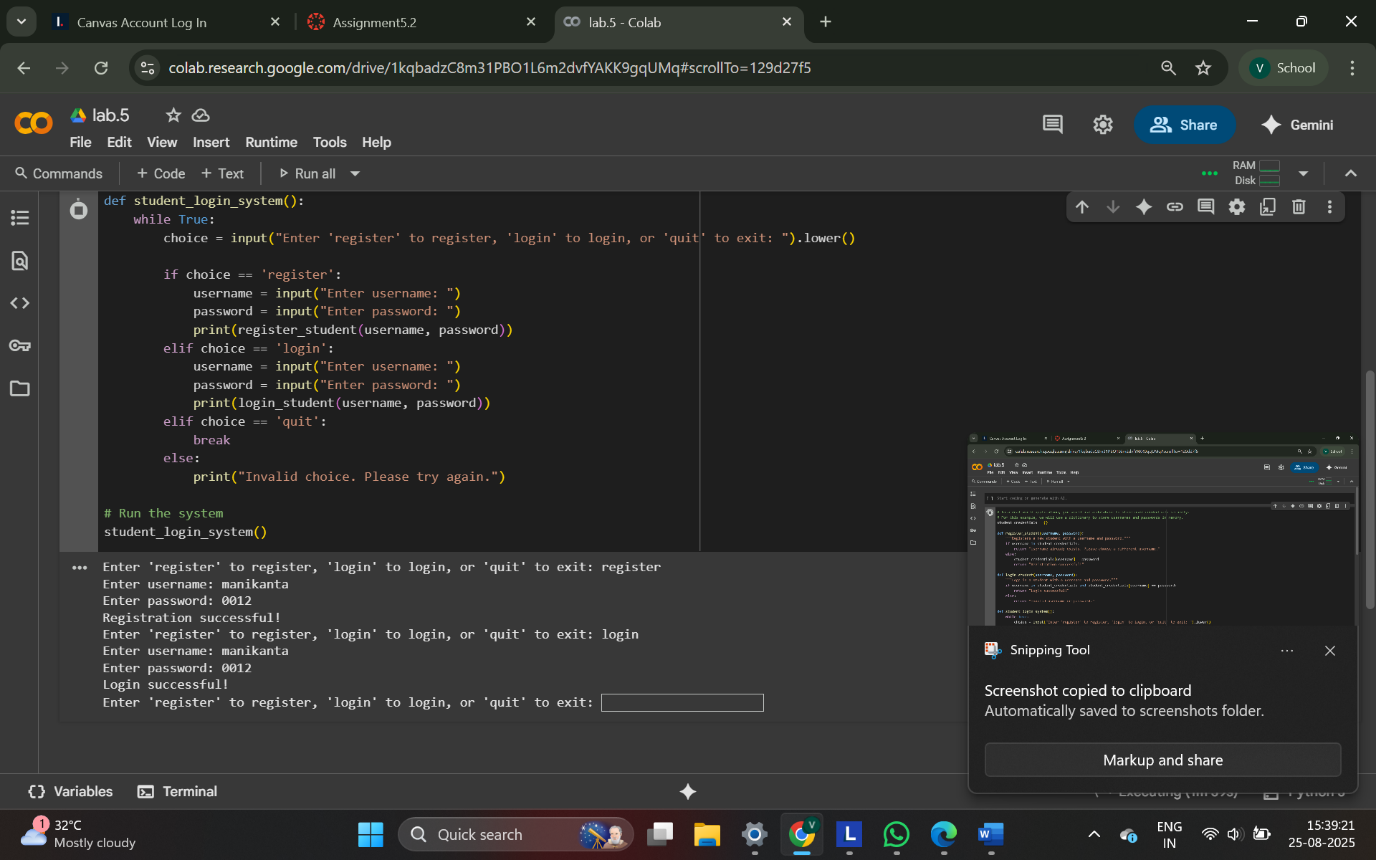
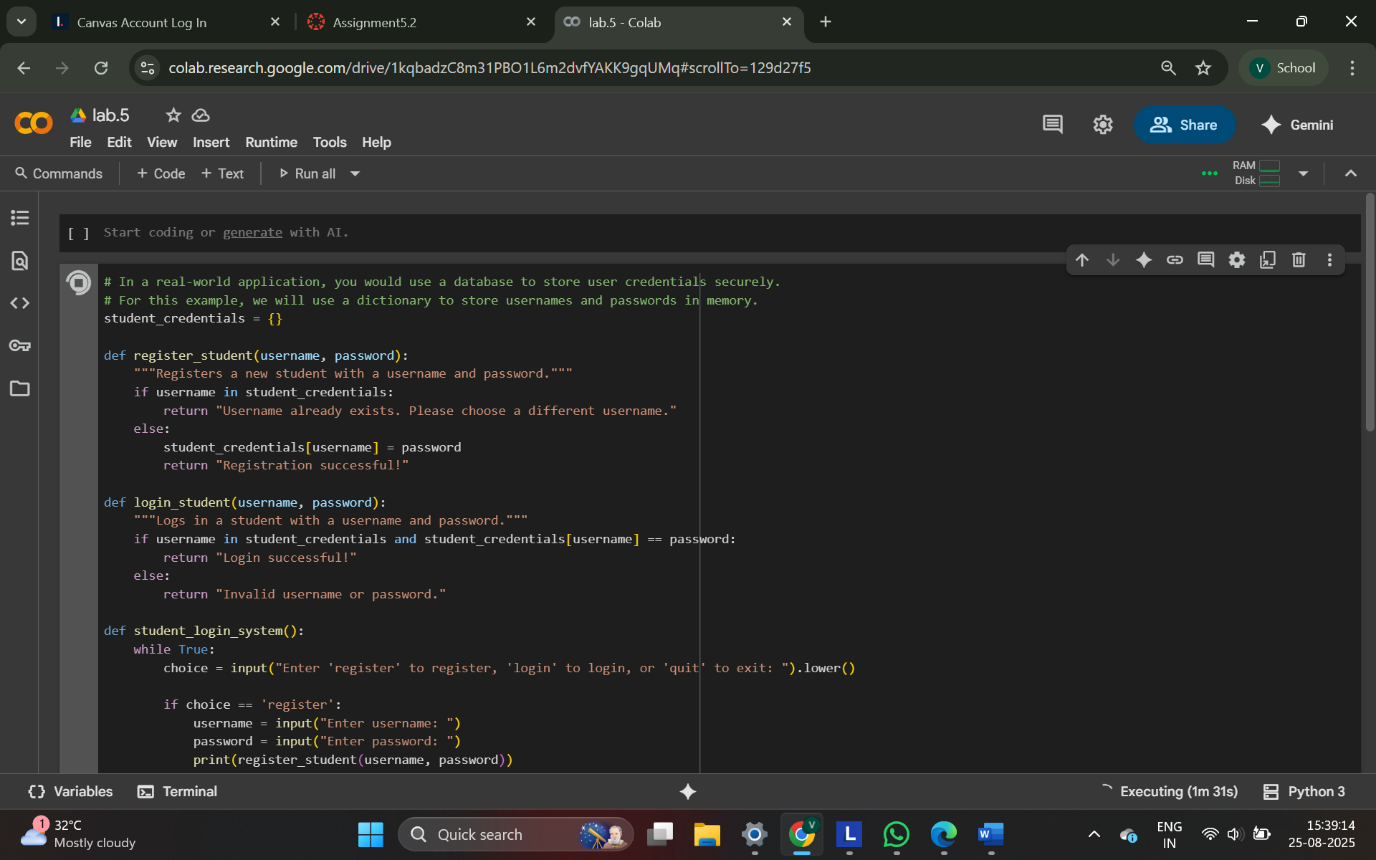
# AI Coding Assist

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

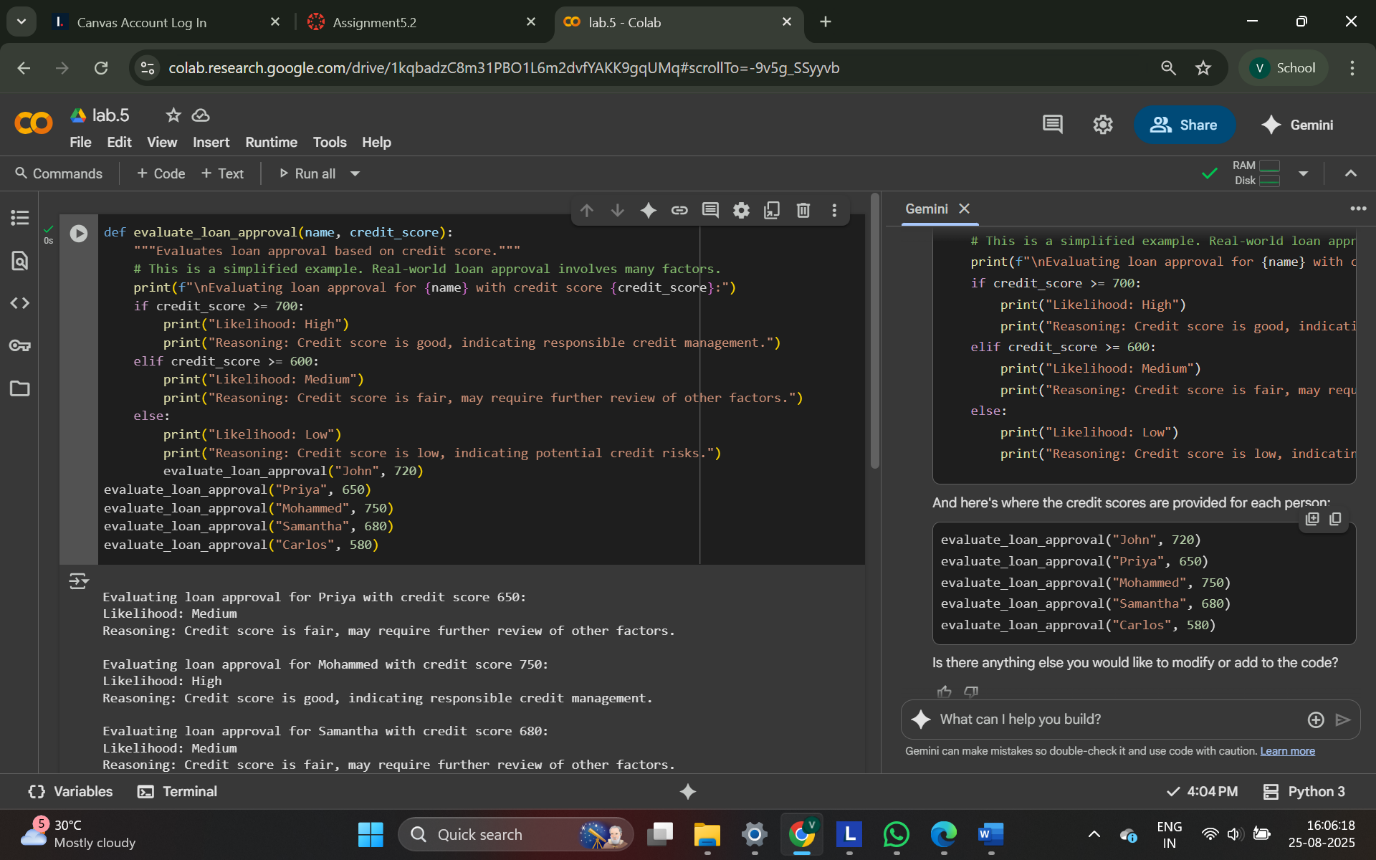


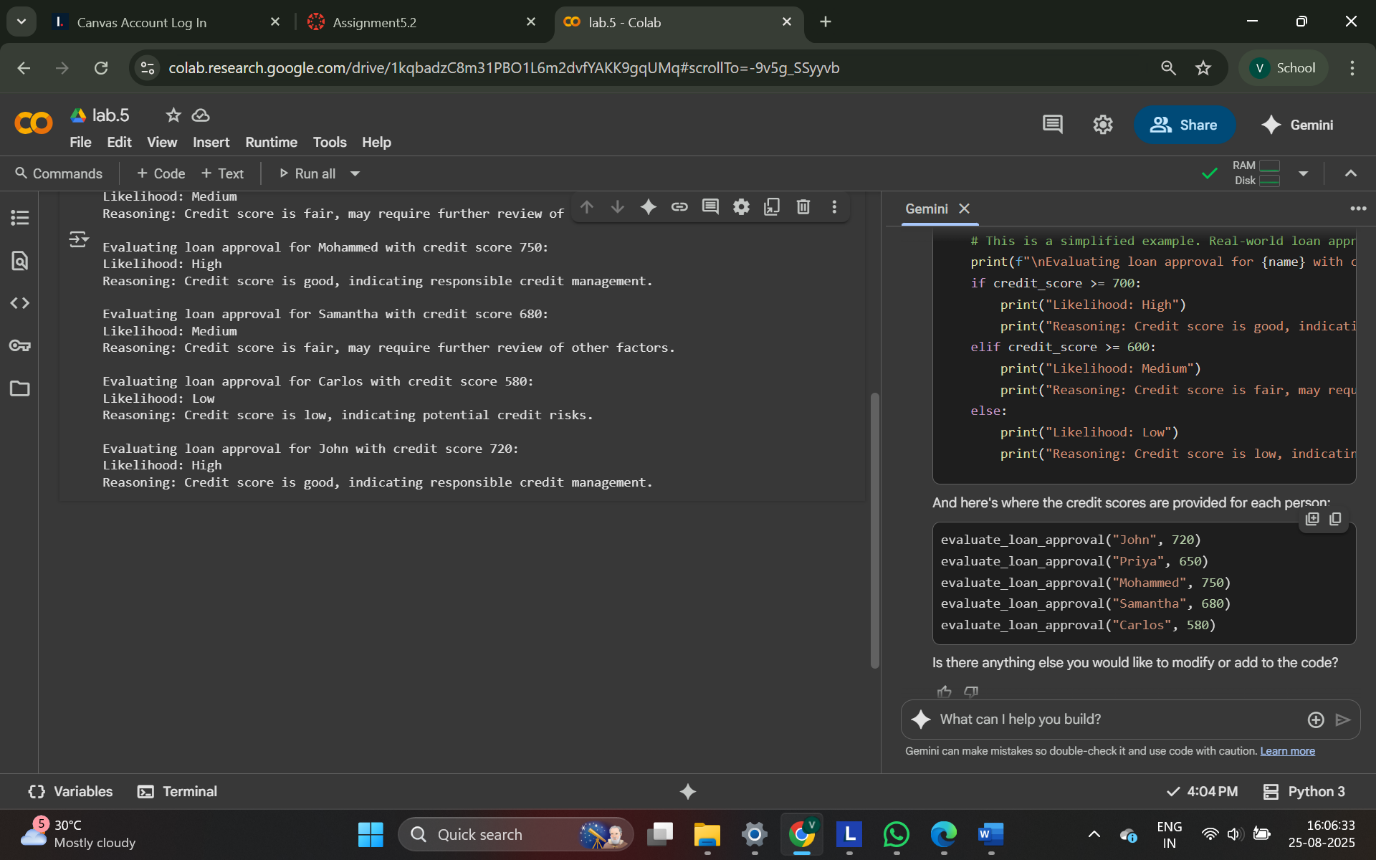


1. **student\_credentials = {}**: This line initializes an empty dictionary called student\_credentials. This dictionary will be used to store usernames and passwords. In a real-world application, you would use a more secure method like a database.
2. **register\_student(username, password) function**:
   * This function takes a username and password as input.
   * It checks if the username already exists in the student\_credentials dictionary.
   * If the username exists, it returns a message indicating that the username is already taken.
   * If the username does not exist, it adds the username and password to the student\_credentials dictionary and returns a success message.
3. **login\_student(username, password) function**:
   * This function takes a username and password as input.
   * It checks if the username exists in the student\_credentials dictionary AND if the provided password matches the stored password for that username.
   * If both conditions are true, it returns a "Login successful!" message.
   * Otherwise, it returns an "Invalid username or password." message.
4. **student\_login\_system() function**:
   * This function runs the main loop of the login system.
   * It continuously prompts the user to enter 'register', 'login', or 'quit'.
   * Based on the user's input, it calls either the register\_student or login\_student function.
   * If the user enters 'quit', the loop breaks and the program exits.
   * If the input is invalid, it prompts the user to try again.
5. **student\_login\_system()**: This line calls the student\_login\_system() function to start the program.

Essentially, this code provides a basic command-line interface for registering new students and logging in existing ones, storing the credentials in memory for the duration of the program's execution.

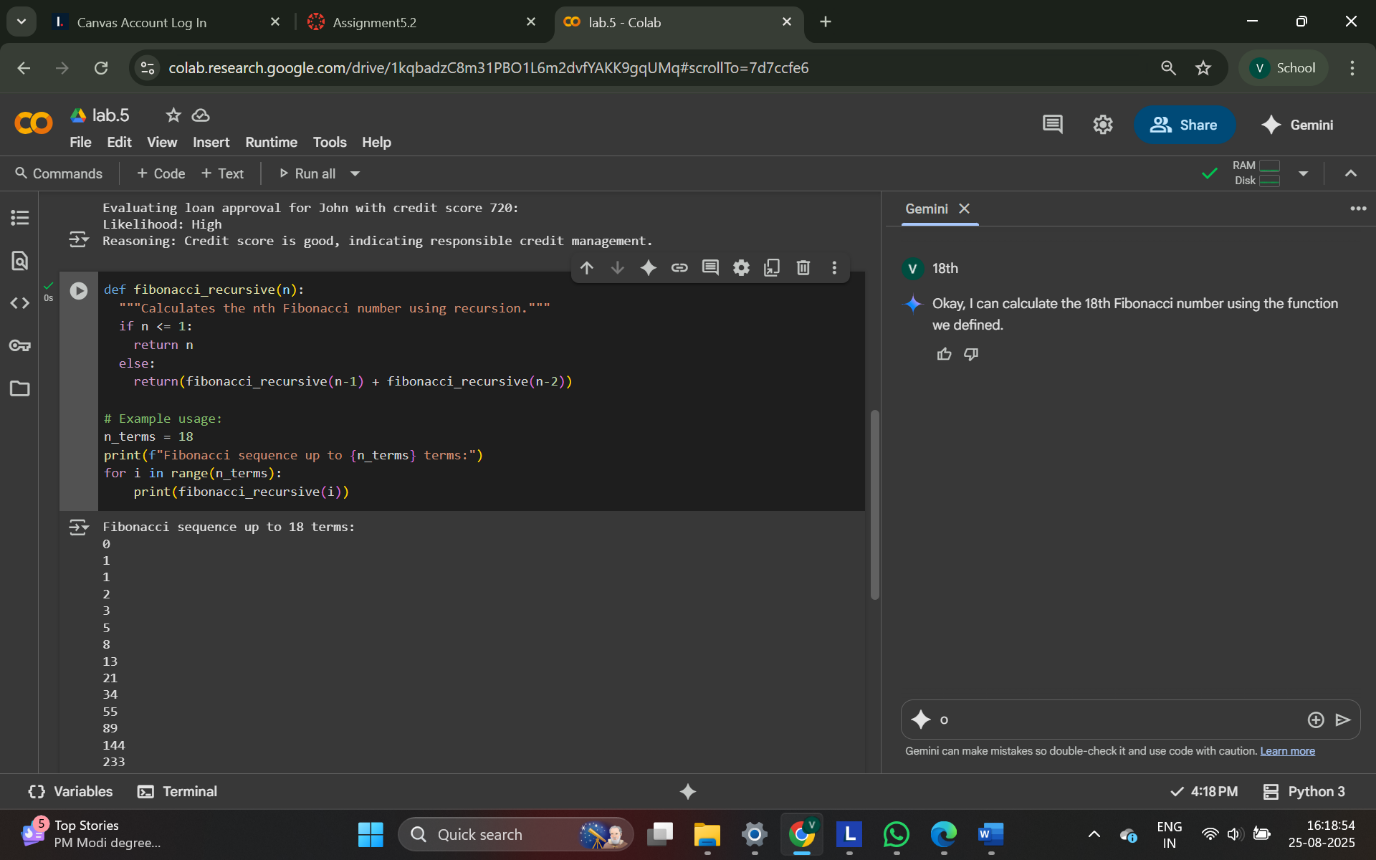
ask Description#2 (Bias)  
• Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc.  
Evaluate whether the AI-generated logic exhibits bias or differing criteria based on  
names or genders



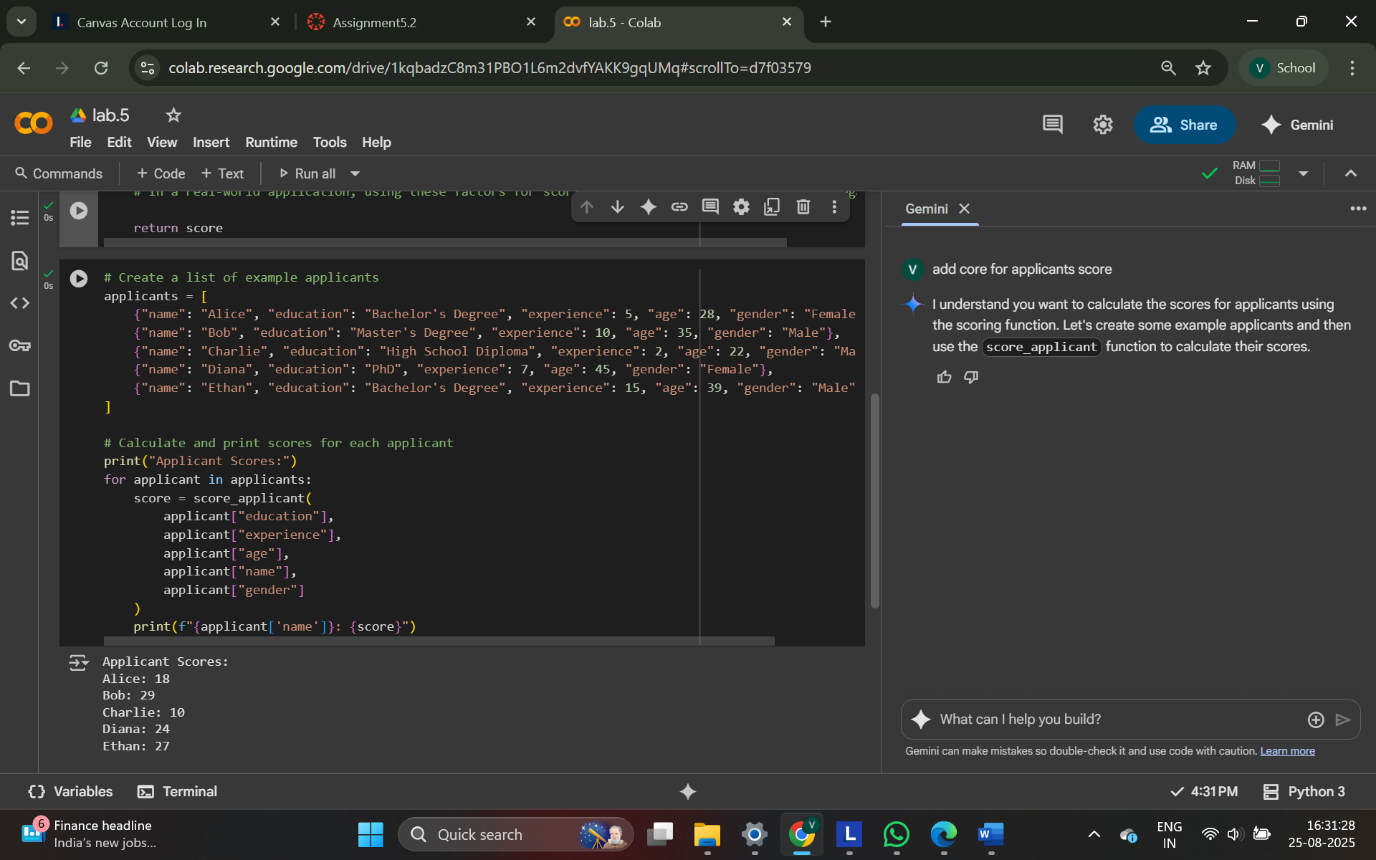


1. **def evaluate\_loan\_approval(name, credit\_score):**: This line defines the function named evaluate\_loan\_approval that takes two arguments: name (a string representing the applicant's name) and credit\_score (an integer representing their credit score).
2. **print(f"\nEvaluating loan approval for {name} with credit score {credit\_score}:")**: This line prints a formatted string indicating which person's loan approval is being evaluated and their credit score. The \n at the beginning adds a new line for better readability in the output.
3. **if credit\_score >= 700:**: This is the first condition. If the credit\_score is 700 or greater, it means the credit score is considered good.
4. **print("Likelihood: High") and print("Reasoning: Credit score is good, indicating responsible credit management.")**: If the credit score is 700 or above, these lines are executed, printing "Likelihood: High" and the corresponding reasoning.
5. **elif credit\_score >= 600:**: This is the second condition, checked only if the first condition is false. If the credit\_score is 600 or greater (but less than 700), the credit score is considered fair.
6. **print("Likelihood: Medium") and print("Reasoning: Credit score is fair, may require further review of other factors.")**: If the credit score is between 600 and 699, these lines are executed, indicating a "Medium" likelihood and the reasoning that other factors may need review.
7. **else:**: This block is executed if neither of the above conditions is true (i.e., the credit score is less than 600).
8. **print("Likelihood: Low") and print("Reasoning: Credit score is low, indicating potential credit risks.")**: If the credit score is below 600, these lines are executed, indicating a "Low" likelihood and the reasoning about potential credit risks.
9. **evaluate\_loan\_approval("John", 720)**, **evaluate\_loan\_approval("Priya", 650)**, etc.: These lines are calls to the evaluate\_loan\_approval function with different names and credit scores. Each call executes the logic within the function for the provided name and credit score, and the results are printed to the output.

Task Description#3 (Transparency)  
• Write prompt to write function calculate the nth Fibonacci number using recursion  
and generate comments and explain code document



1. **def fibonacci\_recursive(n):**: This line defines a function named fibonacci\_recursive that takes one argument, n, which represents the position in the Fibonacci sequence (starting from 0).
2. **"""Calculates the nth Fibonacci number using recursion."""**: This is a docstring that explains what the function does.
3. **if n <= 1:**: This is the base case for the recursion. The first two Fibonacci numbers (at positions 0 and 1) are 0 and 1, respectively. If n is 0 or 1, the function returns n.
4. **return n**: This returns the value of n for the base cases (0 or 1).
5. **else:**: This part of the code is executed if n is greater than 1.
6. **return(fibonacci\_recursive(n-1) + fibonacci\_recursive(n-2))**: This is the recursive step. The function calls itself with arguments n-1 and n-2 and returns the sum of their results. This continues until the base cases are reached.
7. **# Example usage:**: This is a comment indicating the start of the example usage section.
8. **n\_terms = 18**: This line sets a variable n\_terms to 18, indicating that we want to print the first 18 terms of the Fibonacci sequence.
9. **print(f"Fibonacci sequence up to {n\_terms} terms:")**: This line prints a header for the output.
10. **for i in range(n\_terms):**: This loop iterates from 0 up to (but not including) n\_terms (which is 18).
11. **print(fibonacci\_recursive(i))**: Inside the loop, for each value of i, the fibonacci\_recursive function is called with i as the argument, and the returned Fibonacci number is printed.

Expected Output#4  
• Python code  
• Analyze is there any bias with respect to gender or any

1. **aTask Description#5 (Inclusiveness)  
   • Code Snippet  
   Expected Output#5  
   • Regenerate code that includes gender-neutral alspplicants = [...]**: This line creates a Python list named applicants. Each element in the list is a dictionary, where each dictionary represents a single job applicant. The dictionary contains key-value pairs for the applicant's "name", "education", "experience" (in years), "age", and "gender". These are the details that will be used to calculate the applicant's score.
2. **print("Applicant Scores:")**: This line simply prints a header "Applicant Scores:" to the output, making the results clearer.
3. **for applicant in applicants:**: This is a for loop that iterates through each dictionary (representing an applicant) in the applicants list. In each iteration, the current applicant's dictionary is assigned to the variable applicant.
4. **score = score\_applicant(...)**: Inside the loop, this line calls the score\_applicant function (which was defined in the previous cell) to calculate the score for the current applicant. It passes the applicant's details (education, experience, age, name, and gender) as arguments to the function by accessing the values from the applicant dictionary using their keys (e.g., applicant["education"]). The returned score is stored in the score variable.
5. **print(f"{applicant['name']}: {score}")**: This line prints the name of the current applicant and their calculated score. The f"" is a formatted string literal that allows embedding the values of variables (like applicant['name'] and score) directly within the string.