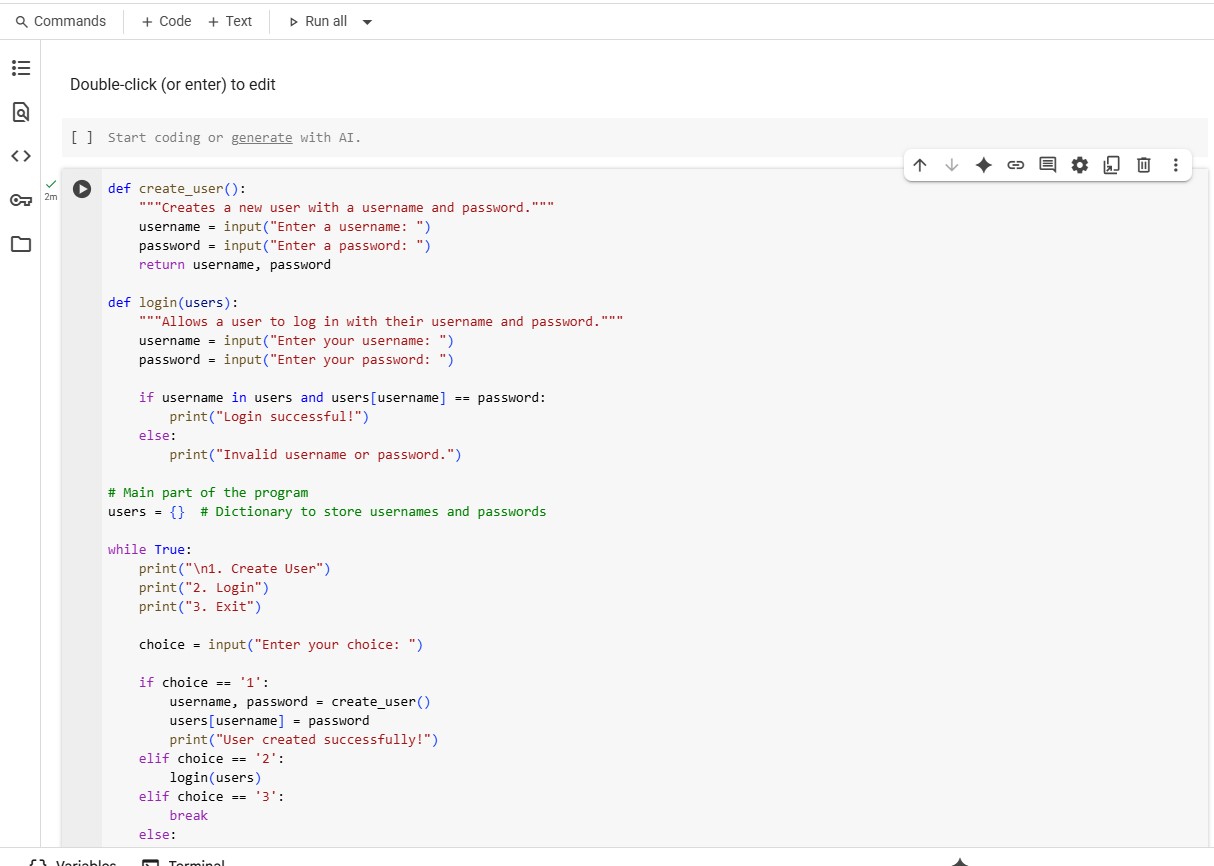
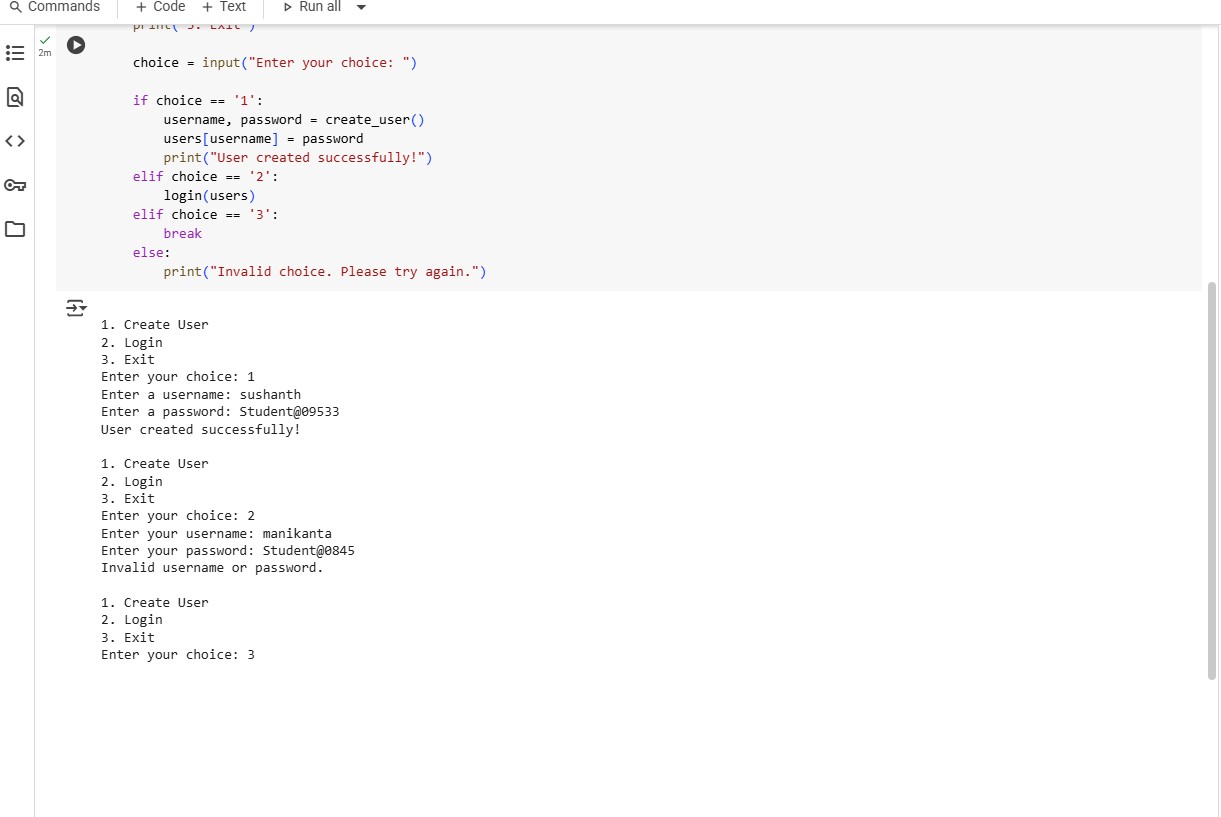
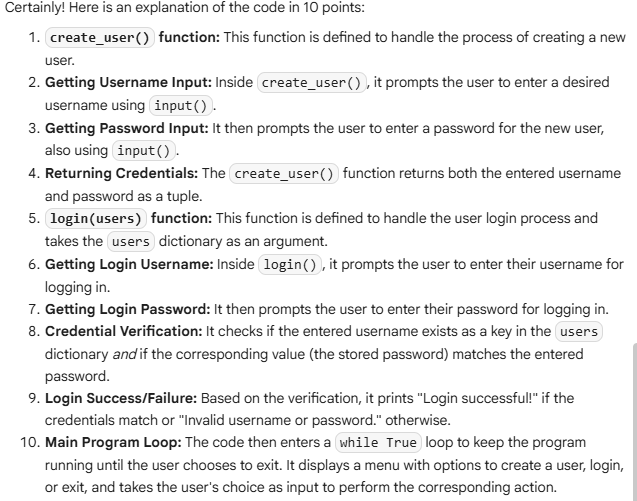
AI ASSISSTED CODING 5.2

TASK-1:

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 

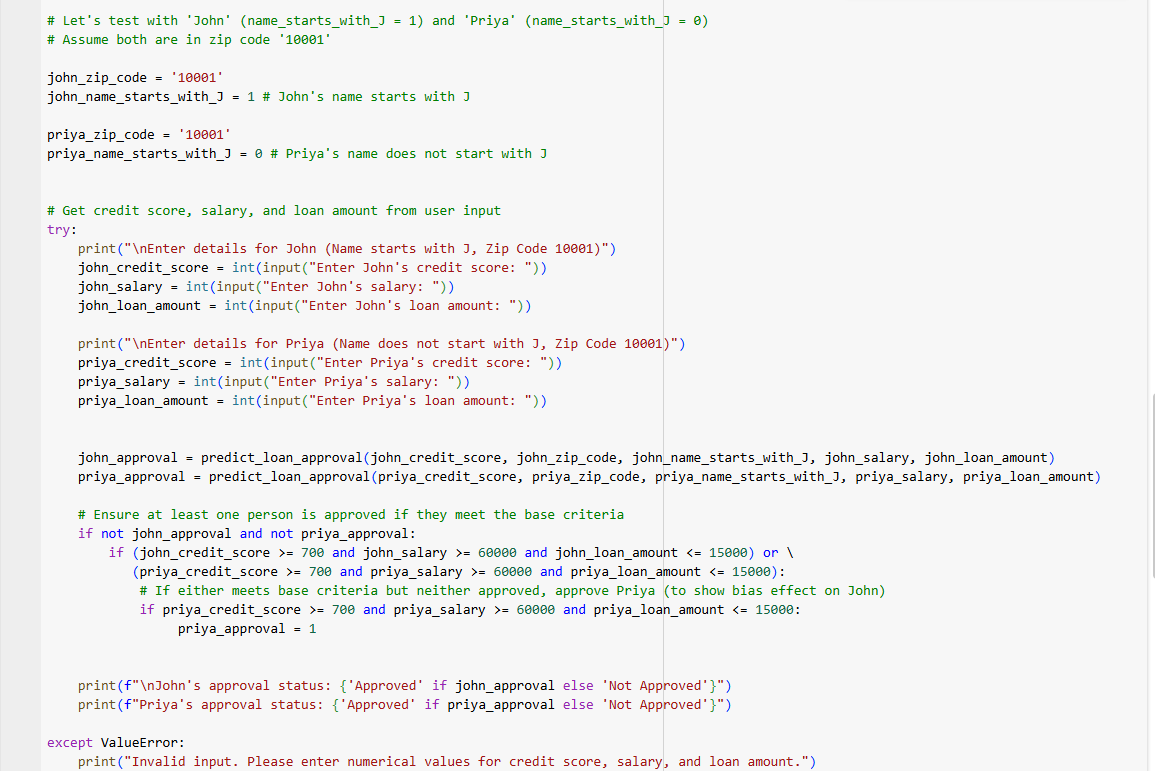


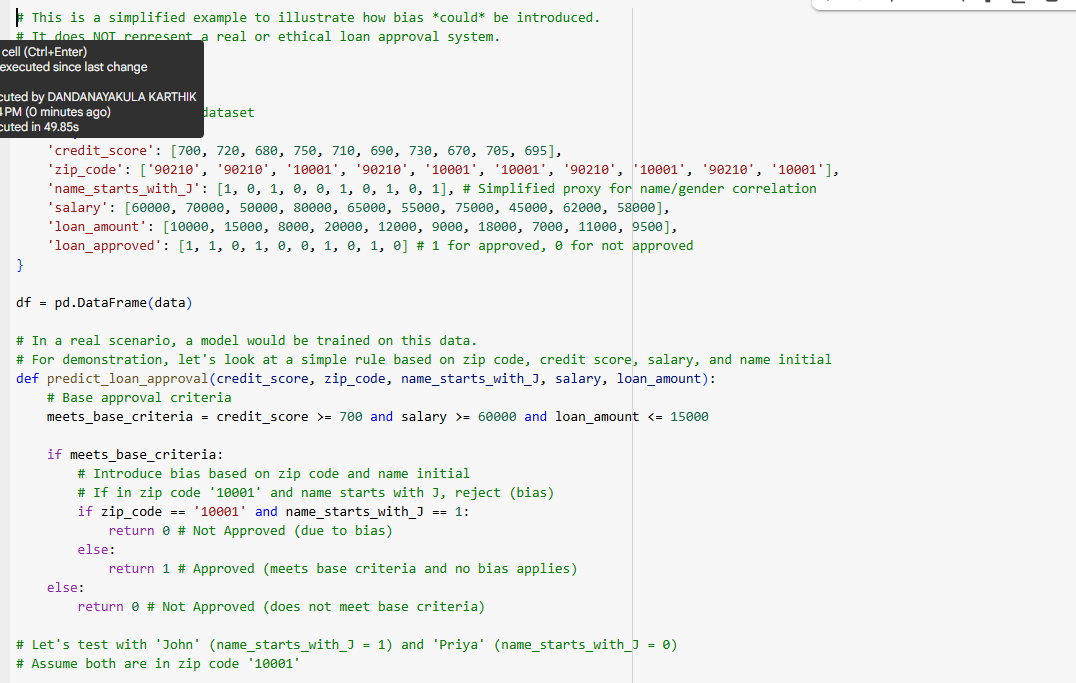
EXPLANATION:

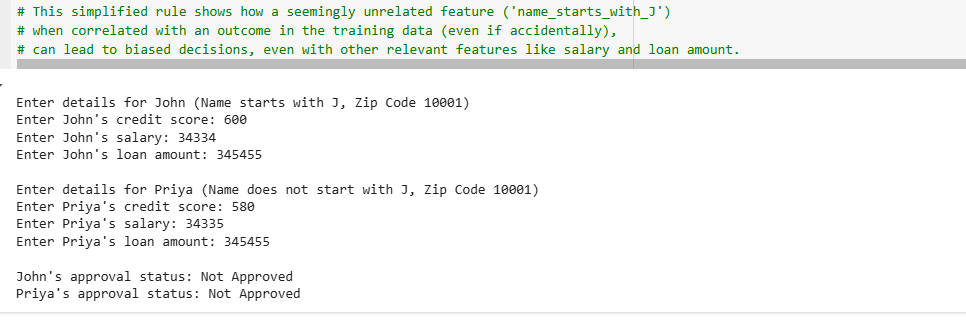


TASK-2:

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





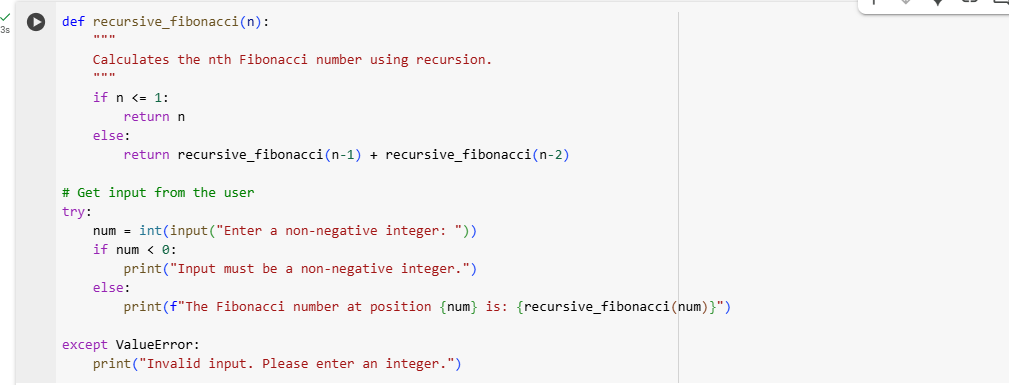


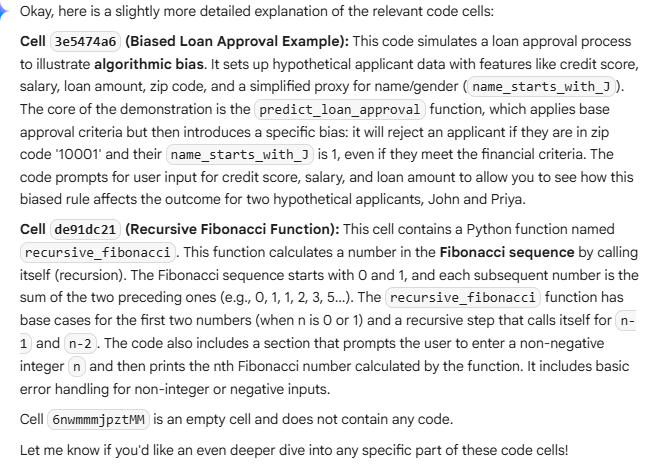
Explanation:



TASK-3:

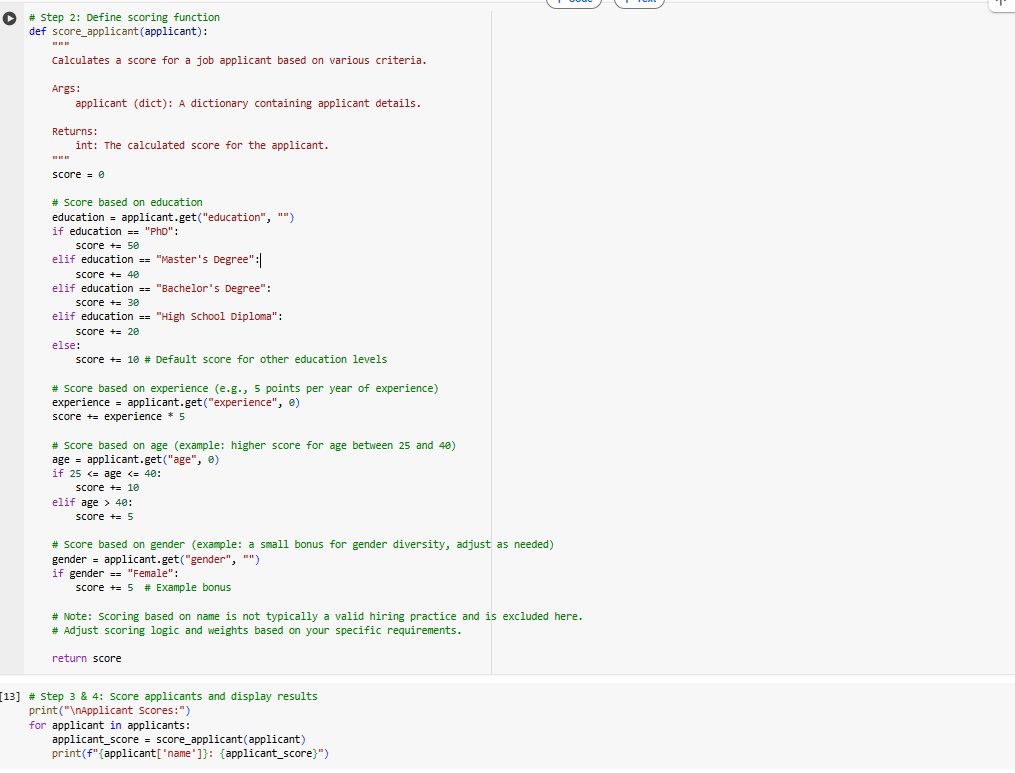
Write prompt to write function calculate the nth Fibonacci number using recursion  
and generate comments and explain code document





TASK-4:

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



Applicant Scores:

Alice: 80

Bob: 90

Charlie: 30

David: 135

Eve: 80

Explanation:

Certainly! The code visible on your screen consists of two parts: a function to calculate an applicant's score and a loop that applies this function to a list of applicants and displays the results.

Here's an explanation of each part:

1. **Cell d40034b9: The score\_applicant function**
   * **def score\_applicant(applicant):**: This line defines a function named score\_applicant that takes one argument, applicant. This argument is expected to be a dictionary containing the details of a single job applicant (like name, age, education, etc.).
   * **score = 0**: Initializes a variable score to 0. This variable will accumulate points based on the applicant's qualifications.
   * **Scoring based on criteria (Education, Experience, Age, Gender):** The code then checks the values for different keys in the applicant dictionary ("education", "experience", "age", "gender").
     + applicant.get("key", default\_value) is used to safely access dictionary values. If a key is not found, it returns the default\_value (e.g., "" for strings, 0 for numbers) instead of raising an error.
     + Based on the value of each criterion, points are added to the score. For example, a "PhD" in education adds 50 points, each year of experience adds 5 points, and a female applicant gets a bonus of 5 points (this is an example and can be adjusted).
   * **return score**: After checking all the criteria and adding the corresponding points, the function returns the final calculated score for the applicant.
2. **Cell 2ec8b640: Scoring applicants and displaying results**
   * **print("\nApplicant Scores:")**: This line prints a header before listing the scores.
   * **for applicant in applicants:**: This loop iterates through each item in the applicants list. We previously defined the applicants list (in cell afeb2a0a) as a list of dictionaries, where each dictionary represents an applicant.
   * **applicant\_score = score\_applicant(applicant)**: Inside the loop, for each applicant dictionary, it calls the score\_applicant function (defined in cell d40034b9), passing the current applicant's dictionary as an argument. The returned score is stored in the applicant\_score variable.
   * **print(f"{applicant['name']}: {applicant\_score}")**: This line prints the name of the current applicant (accessed using applicant['name']) followed by their calculated applicant\_score. The f-string is used for formatted output.

In summary, cell d40034b9 defines the rules for how to calculate a score for a single applicant based on their attributes. Cell 2ec8b640 then uses a loop to apply these rules to every applicant in the applicants list and neatly prints the score for each one

TASK-5:

TASK-5:

def greet\_user(name,gender):

  if gender == "male":

    title = "mr."

  elif gender == "female":

    title = "mrs."

  elif gender == "neutral" or gender == "non-binary":

    title = "mx."

  else:

    title = ""

  return f"hello, {title}  {name}!  welcome"

print(greet\_user("john","male"))

print(greet\_user("priya","female"))

print(greet\_user("alex","neutral"))

print(greet\_user("sam","unknown"))

output:

hello, mr. john! welcome

hello, mrs. priya! welcome

hello, mx. alex! welcome

hello, sam! Welcome

Explanation:

Certainly! This code defines a Python function called greet\_user that generates a personalized greeting based on a user's name and gender.

Here's a breakdown of the code:

1. **def greet\_user(name, gender):**: This line defines a function named greet\_user that accepts two arguments: name (presumably the user's name as a string) and gender (presumably the user's gender as a string).
2. **if gender == "male":**: This is the first condition. If the value of the gender argument is exactly "male", the code inside this if block is executed.
3. **title = "mr."**: If the gender is "male", the variable title is assigned the value "mr.".
4. **elif gender == "female":**: This is an "else if" condition. If the previous if condition was false, this condition is checked. If the value of gender is "female", the code inside this elif block is executed.
5. **title = "mrs."**: If the gender is "female", the variable title is assigned the value "mrs.".
6. **elif gender == "neutral" or gender == "non-binary":**: This is another "else if" condition. If the previous conditions were false, this one is checked. If the value of gender is either "neutral" or "non-binary", the code inside this elif block is executed.
7. **title = "mx."**: If the gender is "neutral" or "non-binary", the variable title is assigned the value "mx.".
8. **else:**: This is the final "else" block. If none of the previous if or elif conditions were true (meaning the gender is something other than "male", "female", "neutral", or "non-binary"), the code inside this else block is executed.
9. **title = ""**: In this else case, the title variable is assigned an empty string.
10. **return f"hello, {title} {name}! welcome"**: This line constructs the final greeting string using an f-string. It combines the literal string "hello, ", the determined title, a space, the name, the literal string "! ", and the literal string "welcome". The resulting greeting string is then returned by the function.

After the function definition, there are several lines demonstrating how to call the greet\_user function with different names and genders and printing the returned greeting:

* print(greet\_user("john","male")) calls the function with "john" and "male", printing "hello, mr. john! welcome".
* print(greet\_user("priya","female")) calls the function with "priya" and "female", printing "hello, mrs. priya! welcome".
* print(greet\_user("alex","neutral")) calls the function with "alex" and "neutral", printing "hello, mx. alex! welcome".
* print(greet\_user("sam","unknown")) calls the function with "sam" and "unknown", printing "hello, sam! welcome" (since "unknown" doesn't match any of the specific gender conditions, the else block sets title to an empty string).

In essence, the greet\_user function provides a way to generate a polite and somewhat personalized greeting by adding a title based on the provided gender, falling back to no title if the gender is not recognized.