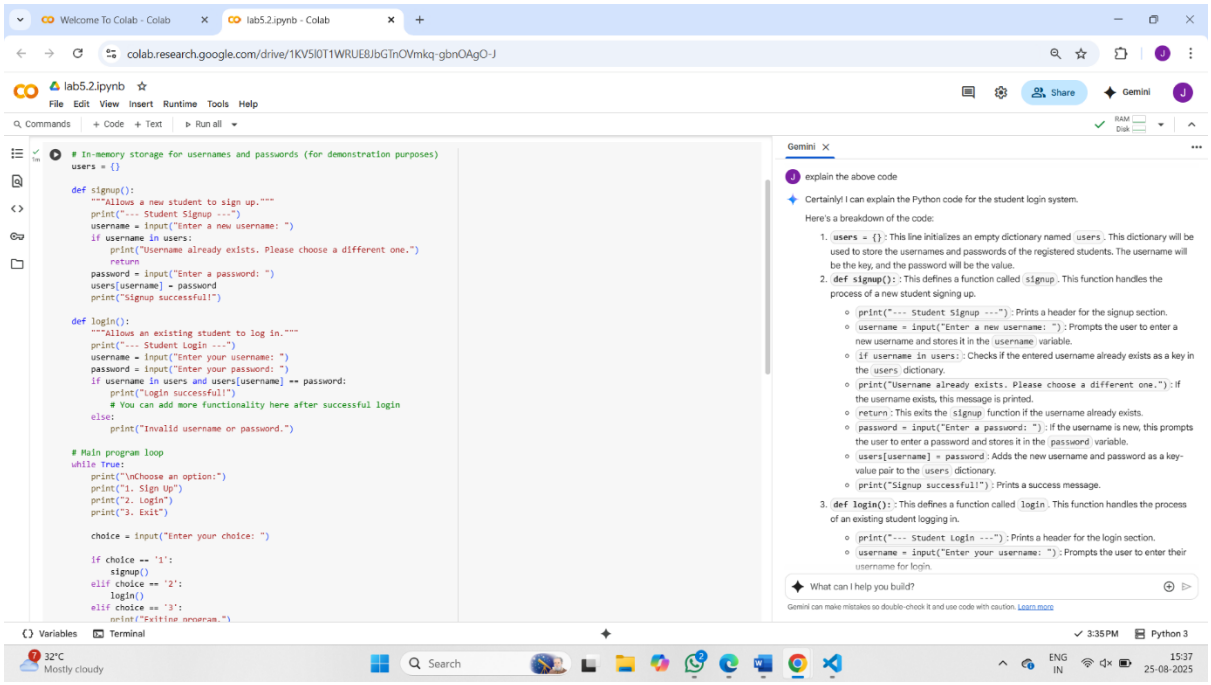
AI ASSISTED CODING

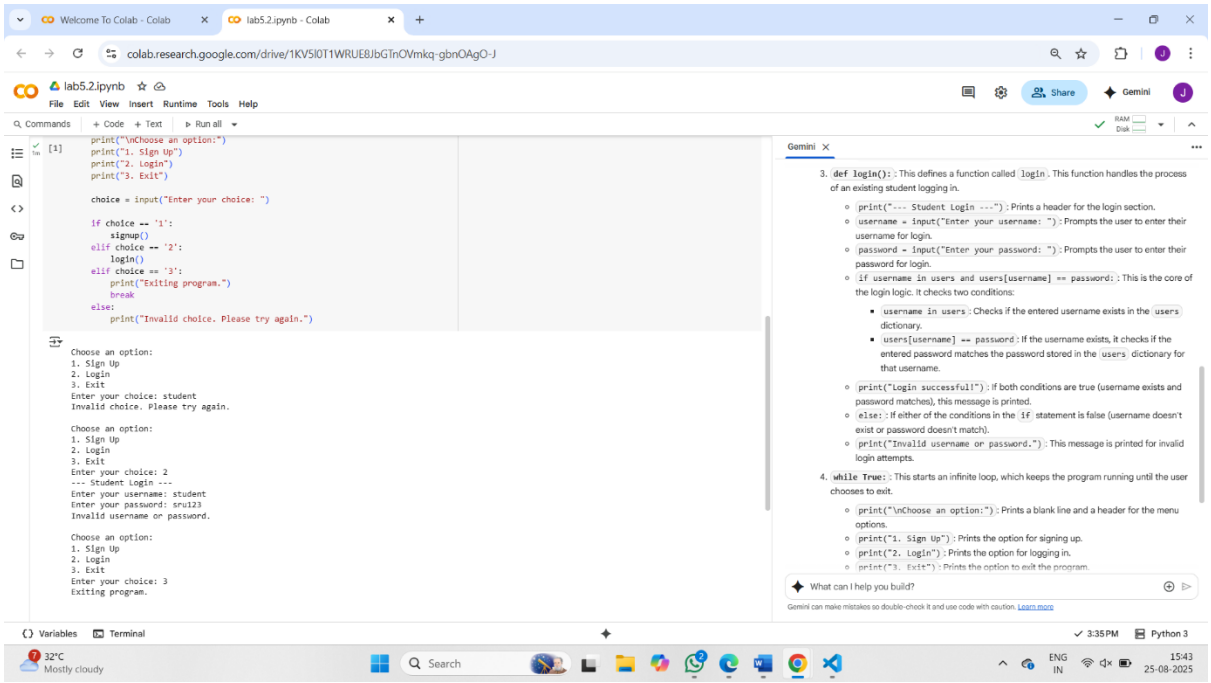
LAB– 5.2

Name: N.Jyoshna Sri

HALL NO:2403A52106

TASK-1:





EXPLANATION:

Disk

Gemini ×

explain the above code

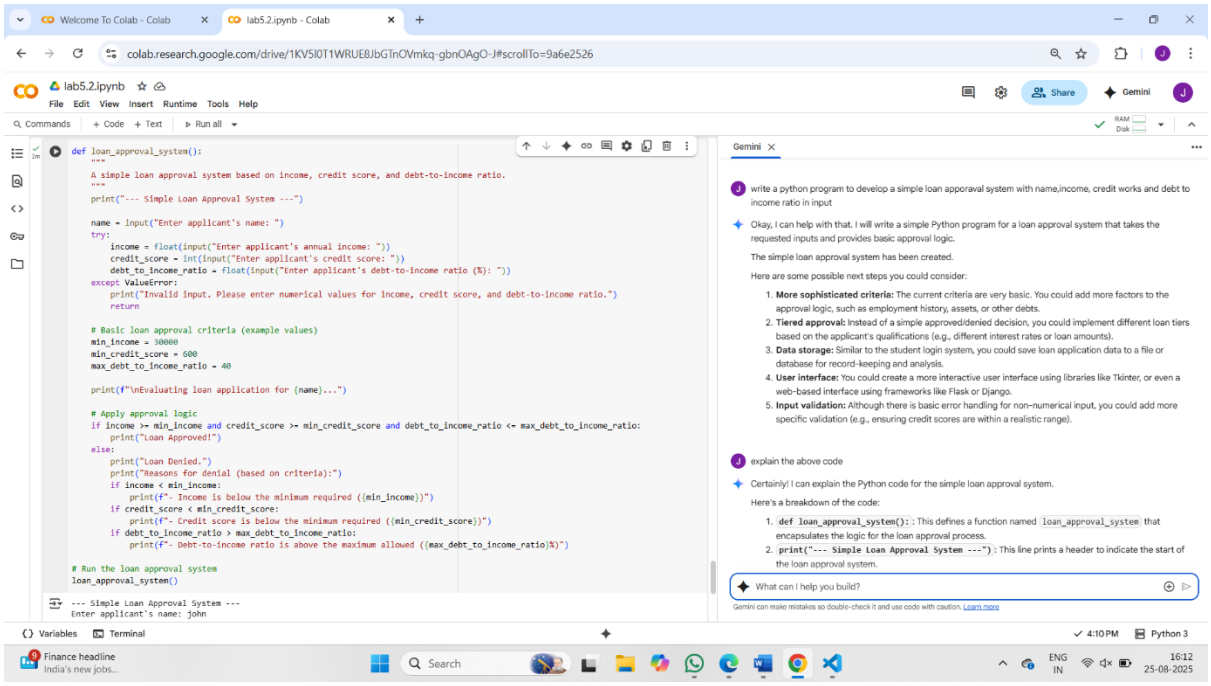
Certainly! I can explain the Python code for the student login system.

Here's a breakdown of the code:

1. `users = {}`: This line initializes an empty dictionary named `users`. This dictionary will be used to store the usernames and passwords of the registered students. The username will be the key, and the password will be the value.
2. `def signup():`: This defines a function called `signup`. This function handles the process of a new student signing up.
    - `print("--- Student Signup ---")`: Prints a header for the signup section.
    - `username = input("Enter a new username: ")`: Prompts the user to enter a new username and stores it in the `username` variable.
    - `if username in users:`: Checks if the entered username already exists as a key in the `users` dictionary.
    - `print("Username already exists. Please choose a different one.")`: If the username exists, this message is printed.
    - `return`: This exits the `signup` function if the username already exists.
    - `password = input("Enter a password: ")`: If the username is new, this prompts the user to enter a password and stores it in the `password` variable.
    - `users[username] = password`: Adds the new username and password as a key-value pair to the `users` dictionary.
    - `print("Signup successful!")`: Prints a success message.
3. `def login():`: This defines a function called `login`. This function handles the process of an existing student logging in.
    - `print("--- Student Login ---")`: Prints a header for the login section.
    - `username = input("Enter your username: ")`: Prompts the user to enter their username for login.

TASK-2:

CODE:



Output:

```
# Run the loan approval system
loan_approval_system()
```

```
--- Simple Loan Approval System ---
Enter applicant's name: john
Enter applicant's annual income: 4000000
Enter applicant's credit score: 650
Enter applicant's debt-to-income ratio (%): 0.4

Evaluating loan application for john...
Loan Approved!
```

EXPLANATION:

Share Gemini J

RAM Disk

Gemini

explain the above code

Certainly! I can explain the Python code for the simple loan approval system.

Here's a breakdown of the code:

1. `def loan_approval_system():`: This defines a function named `loan_approval_system` that encapsulates the logic for the loan approval process.
2. `print("--- Simple Loan Approval System ---")`: This line prints a header to indicate the start of the loan approval system.
3. **Input Collection**:
    - `name = input("Enter applicant's name: ")`: Prompts the user to enter the applicant's name and stores it in the `name` variable.
    - `try...except ValueError:`: This block attempts to collect numerical input for income, credit score, and debt-to-income ratio. It uses a `try...except` block to catch `ValueError` if the user enters non-numerical input.
    - `income = float(input("Enter applicant's annual income: "))`: Prompts for annual income and converts the input to a floating-point number.
    - `credit_score = int(input("Enter applicant's credit score: "))`: Prompts for credit score and converts the input to an integer.
    - `debt_to_income_ratio = float(input("Enter applicant's debt-to-income ratio (%): "))`: Prompts for debt-to-income ratio and converts the input to a floating-point number.
    - If a `ValueError` occurs (invalid input), it prints an error message and the `return` statement exits the function.
4. **Approval Criteria**:
    - `min_income = 30000`: Sets the minimum required annual income for loan approval.
    - `min_credit_score = 600`: Sets the minimum required credit score for loan approval.
    - `max_debt_to_income_ratio = 40`: Sets the maximum allowed debt-to-income ratio for loan approval.
5. Evaluation:

What can I help you build?

4:10 PM Python 3

Gemini

the function.

4. **Approval Criteria**:
   - `min_income = 30000`: Sets the minimum required annual income for loan approval.
   - `min_credit_score = 600`: Sets the minimum required credit score for loan approval.
   - `max_debt_to_income_ratio = 40`: Sets the maximum allowed debt-to-income ratio for loan approval.
5. **Evaluation**:
   - `print(f"\nEvaluating loan application for {name}...")`: Prints a message indicating that the evaluation is in progress, including the applicant's name.
   - `if income >= min_income and credit_score >= min_credit_score and debt_to_income_ratio <= max_debt_to_income_ratio:`: This is the core logic for loan approval. It checks if all three conditions (income meets or exceeds the minimum, credit score meets or exceeds the minimum, and debt-to-income ratio is less than or equal to the maximum) are true.
   - `print("Loan Approved!")`: If all conditions are met, this message is printed.
   - `else:`: If any of the conditions are not met.
   - `print("Loan Denied.")`: This message is printed if the loan is denied.
   - `print("Reasons for denial (based on criteria):")`: Prints a header for the reasons for denial.
   - The following `if` statements check which specific criteria were not met and print corresponding messages.
6. `loan_approval_system()`: This line calls the `loan_approval_system` function to start the program when the script is executed.

In essence, this program collects financial information from a loan applicant and applies a simple set of rules to determine whether to approve or deny the loan, providing reasons for denial if applicable.

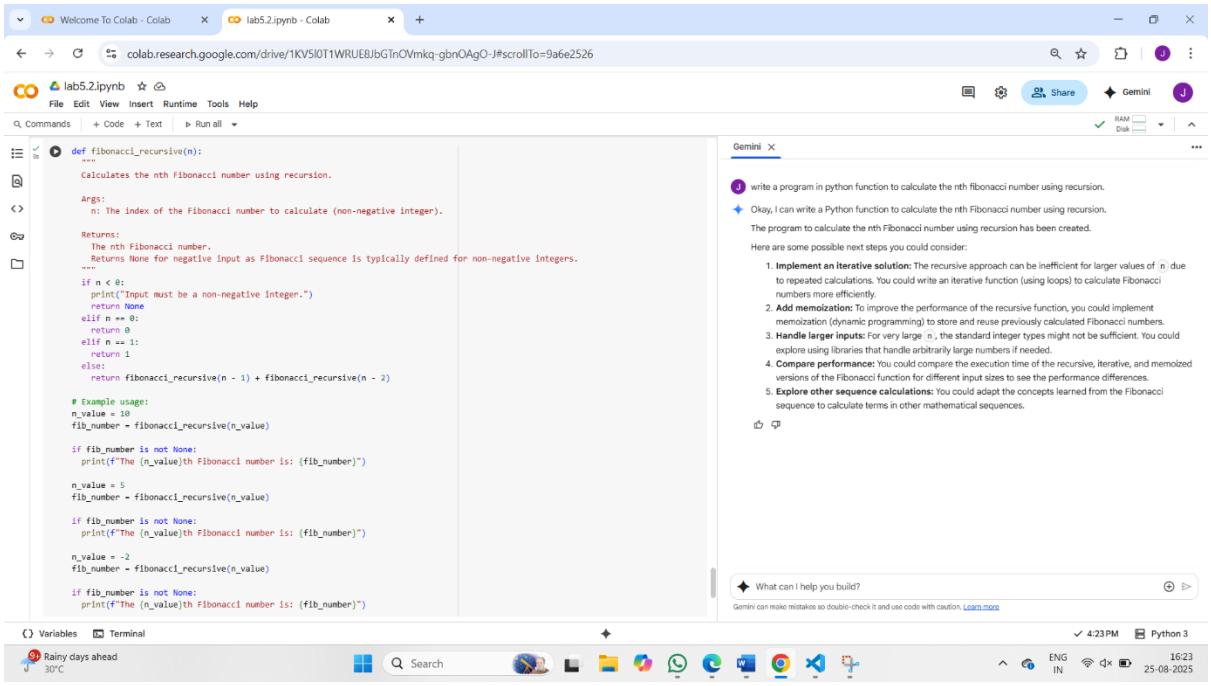
What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

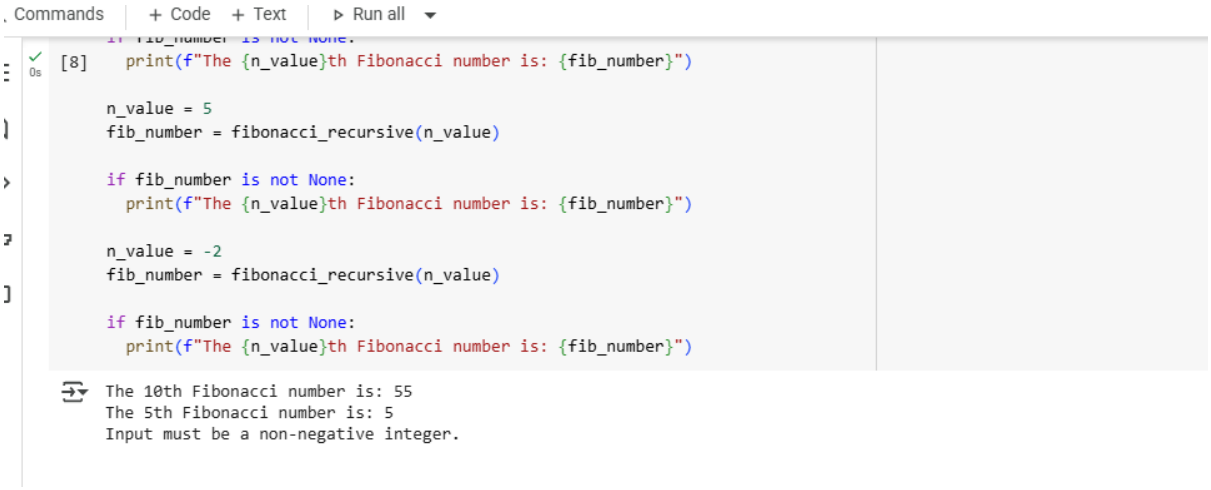
4:10 PM Python 3

TASK-3:

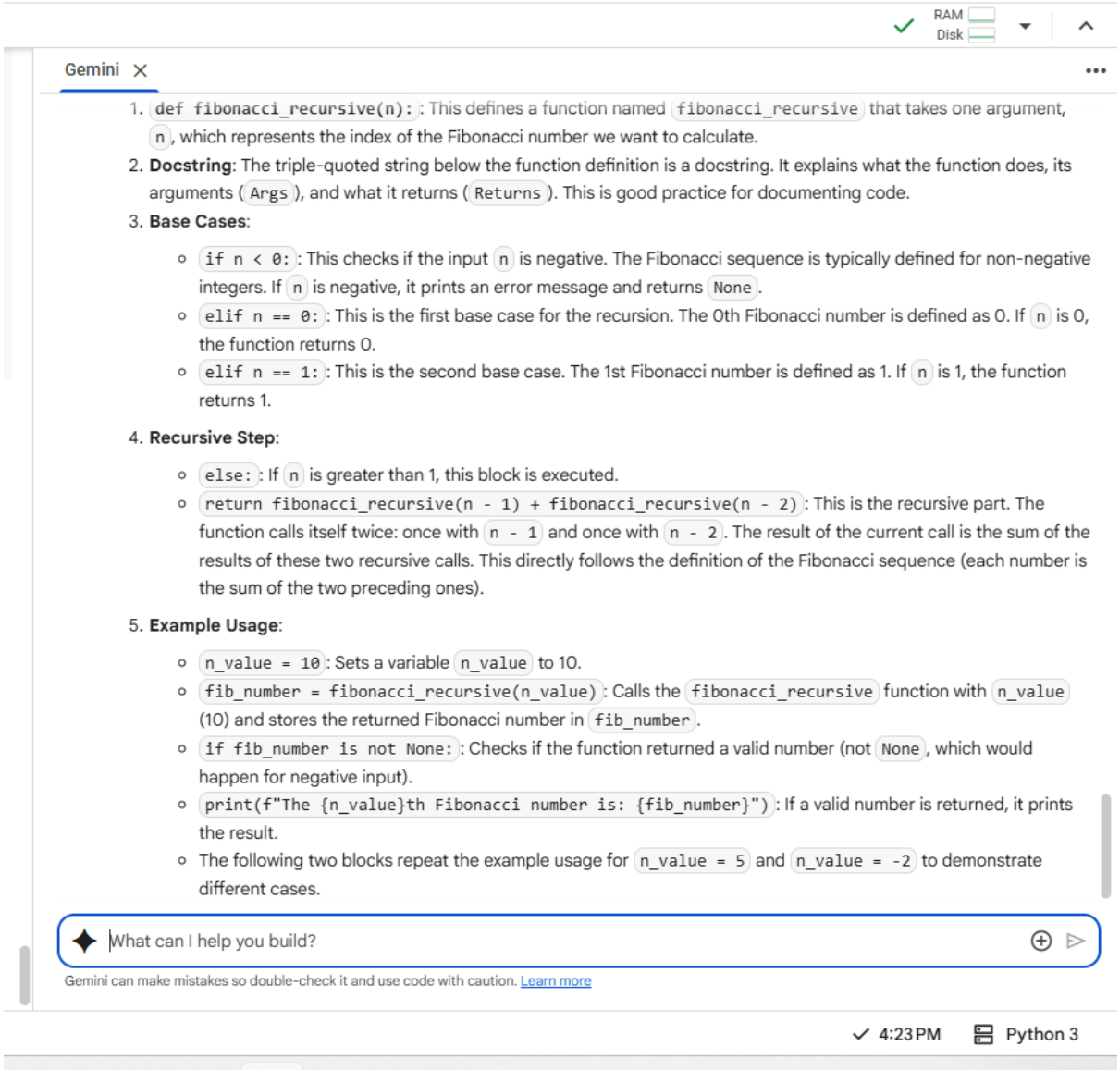
Code:



Output:

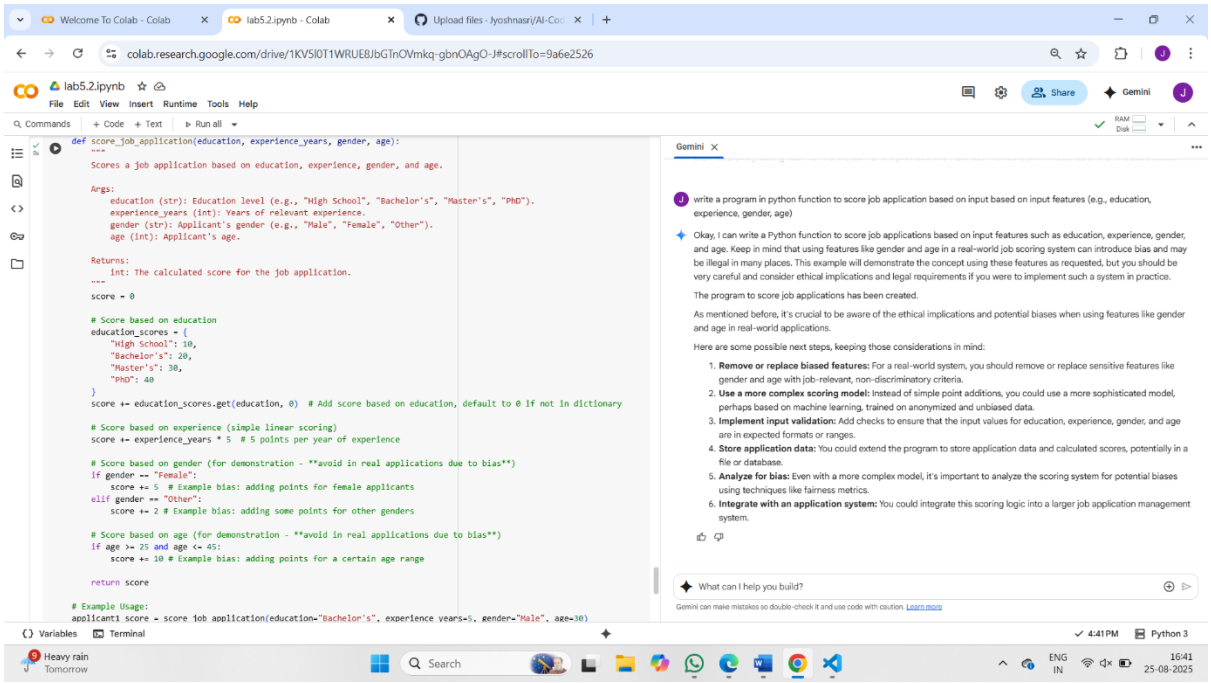


Explanation:


RAM
Disk
Gemini
1. def fibonacci_recursive(n): : This defines a function named fibonacci_recursive that takes one argument, n , which represents the index of the Fibonacci number we want to calculate.
2. Docstring: The triple-quoted string below the function definition is a docstring. It explains what the function does, its arguments (Args), and what it returns (Returns). This is good practice for documenting code.
3. Base Cases:
- if n < 0: : This checks if the input n is negative. The Fibonacci sequence is typically defined for non-negative integers. If n is negative, it prints an error message and returns None .
- elif n == 0: : This is the first base case for the recursion. The 0th Fibonacci number is defined as 0. If n is 0, the function returns 0.
- elif n == 1: : This is the second base case. The 1st Fibonacci number is defined as 1. If n is 1, the function returns 1.
4. Recursive Step:
- else: : If n is greater than 1, this block is executed.
- return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2) : This is the recursive part. The function calls itself twice: once with n - 1 and once with n - 2 . The result of the current call is the sum of the results of these two recursive calls. This directly follows the definition of the Fibonacci sequence (each number is the sum of the two preceding ones).
5. Example Usage:
- n_value = 10 : Sets a variable n_value to 10.
- fib_number = fibonacci_recursive(n_value) : Calls the fibonacci_recursive function with n_value (10) and stores the returned Fibonacci number in fib_number .
- if fib_number is not None: : Checks if the function returned a valid number (not None , which would happen for negative input).
- print(f"The {n_value}th Fibonacci number is: {fib_number}") : If a valid number is returned, it prints the result.
- The following two blocks repeat the example usage for n_value = 5 and n_value = -2 to demonstrate different cases.
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more
4:23 PM
Python 3

TASK-4:

CODE:



Output:

```
    return score

# Example Usage:
applicant1_score = score_job_application(education="Bachelor's", experience_years=5, gender="Male", age=30)
print(f"Applicant 1 Score: {applicant1_score}")

applicant2_score = score_job_application(education="Master's", experience_years=10, gender="Female", age=40)
print(f"Applicant 2 Score: {applicant2_score}")

applicant3_score = score_job_application(education="High School", experience_years=2, gender="Other", age=20)
print(f"Applicant 3 Score: {applicant3_score}")
```

```
Applicant 1 Score: 55
Applicant 2 Score: 95
Applicant 3 Score: 22
```

Code explanation:

Share Gemini J

RAM Disk

Gemini ×

J explain the above code

Certainly! I can explain the Python code for the job application scoring function.

Here's a breakdown of the code:

1. `def score_job_application(education, experience_years, gender, age):`: This defines a function named `score_job_application` that takes four arguments: `education`, `experience_years`, `gender`, and `age`. These represent the features of a job applicant used for scoring.
2. **Docstring**: The triple-quoted string explains the purpose of the function, its arguments, and what it returns. It also includes a crucial warning about the potential for bias and illegality when using gender and age in real-world systems.
3. `score = 0`: Initializes a variable `score` to 0. This variable will accumulate points based on the applicant's features.
4. **Scoring based on Education**:
   - `education_scores = {...}`: Creates a dictionary `education_scores` that maps different education levels to specific point values.
   - `score += education_scores.get(education, 0)`: This line adds points to the `score` based on the applicant's `education` level. `education_scores.get(education, 0)` attempts to retrieve the score for the given `education` level from the dictionary. If the education level is not found in the dictionary, it defaults to adding 0 points.
5. **Scoring based on Experience**:
   - `score += experience_years * 5`: This line adds points based on the number of years of relevant experience. It's a simple linear scoring where each year of experience adds 5 points to the score.
6. **Scoring based on Gender (Demonstration of Bias)**:
   - `if gender == "Female": score += 5`: This `if` statement demonstrates how bias could be introduced. It adds 5 points to the score specifically for applicants whose `gender` is "Female".
   - `elif gender == "Other": score += 2`: This `elif` statement adds 2 points for applicants whose `gender` is "Other". The comment explicitly mentions this is an example of bias and should be avoided in real applications.
7. **Scoring based on Age (Demonstration of Bias)**:

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

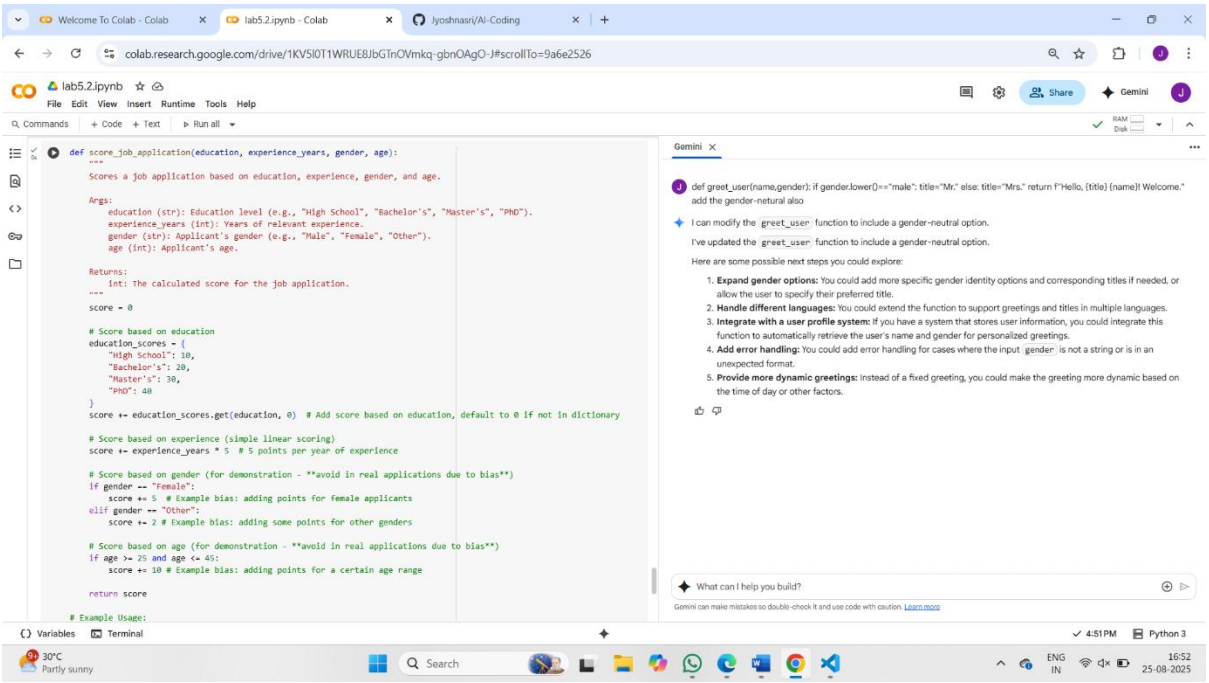
It's a simple linear scoring where each year of experience adds 5 points to the score.

6. **Scoring based on Gender (Demonstration of Bias)**:
    - `if gender == "Female": score += 5`: This `if` statement demonstrates how bias could be introduced. It adds 5 points to the score specifically for applicants whose `gender` is "Female".
    - `elif gender == "Other": score += 2`: This `elif` statement adds 2 points for applicants whose `gender` is "Other". The comment explicitly mentions this is an example of bias and should be avoided in real applications.
7. **Scoring based on Age (Demonstration of Bias)**:
    - `if age >= 25 and age <= 45: score += 10`: This `if` statement demonstrates another example of potential bias. It adds 10 points to the score for applicants whose `age` falls within the range of 25 to 45. The comment again highlights this as an example of bias to avoid.
8. `return score`: The function returns the final calculated `score` for the job application.
9. **Example Usage**:
    - The code then provides three example calls to the `score_job_application` function with different input values for education, experience, gender, and age.
    - `applicant1_score = ...`: Calls the function for Applicant 1 and stores the result.
    - `print(f"Applicant 1 Score: {applicant1_score}")`: Prints the calculated score for Applicant 1.
    - This is repeated for Applicant 2 and Applicant 3 to show how different inputs result in different scores based on the defined logic.

In essence, this function takes several features of a job applicant and calculates a numerical score based on predefined rules. The code includes explicit examples of how including sensitive features like gender and age can lead to biased outcomes, serving as a caution for real-world implementations.

TASK-5:

CODE:



Output:

```
        return score

    # Example Usage:
    applicant1_score = score_job_application(education="Bachelor's", experience_years=5, gender="Male", age=30)
    print(f"Applicant 1 Score: {applicant1_score}")

    applicant2_score = score_job_application(education="Master's", experience_years=10, gender="Female", age=40)
    print(f"Applicant 2 Score: {applicant2_score}")

    applicant3_score = score_job_application(education="High School", experience_years=2, gender="Other", age=20)
    print(f"Applicant 3 Score: {applicant3_score}")
```

```
Applicant 1 Score: 55
Applicant 2 Score: 95
Applicant 3 Score: 22
```

Code explanation: