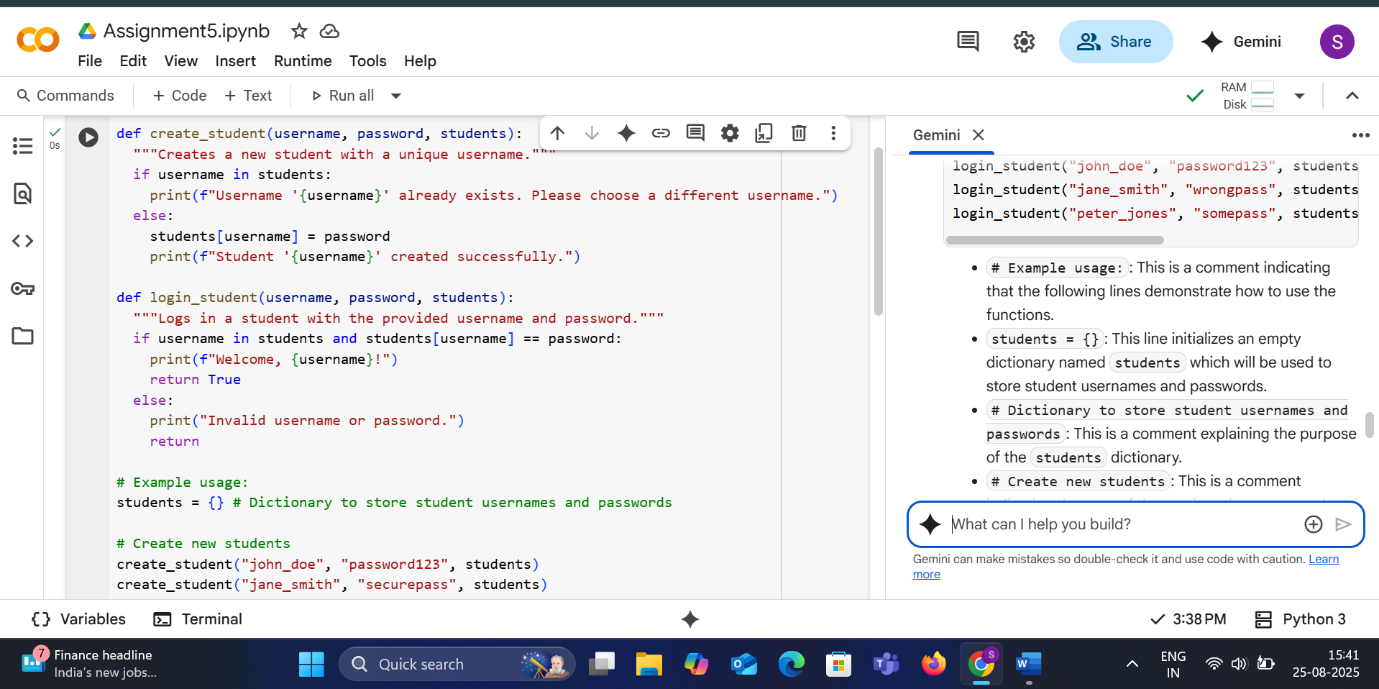
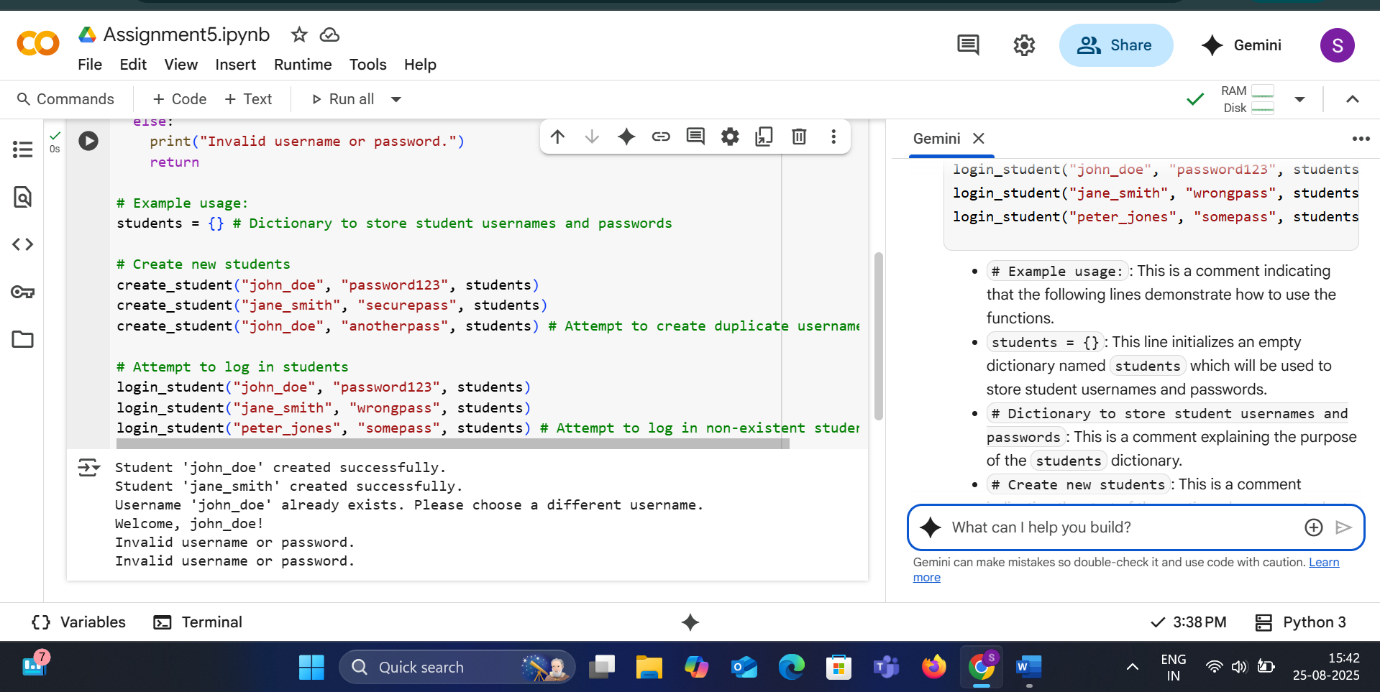
**ASSINGMENT 5.2**

* **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**
* def create\_student(username, password, students):: This line defines a function named create\_student that takes three arguments: username, password, and students.
* """Creates a new student with a unique username.""": This is a docstring, which explains what the function does.
* if username in students:: This line checks if the provided username already exists as a key in the students dictionary.
* print(f"Username '{username}' already exists. Please choose a different username."): If the username exists, this line prints a message indicating that the username is already taken.
* else:: This keyword indicates the alternative action if the username does not exist.
* students[username] = password: If the username is unique, this line adds the new student to the students dictionary, using the username as the key and the password as the value.
* print(f"Student '{username}' created successfully."): This line prints a success message indicating that the student was created.

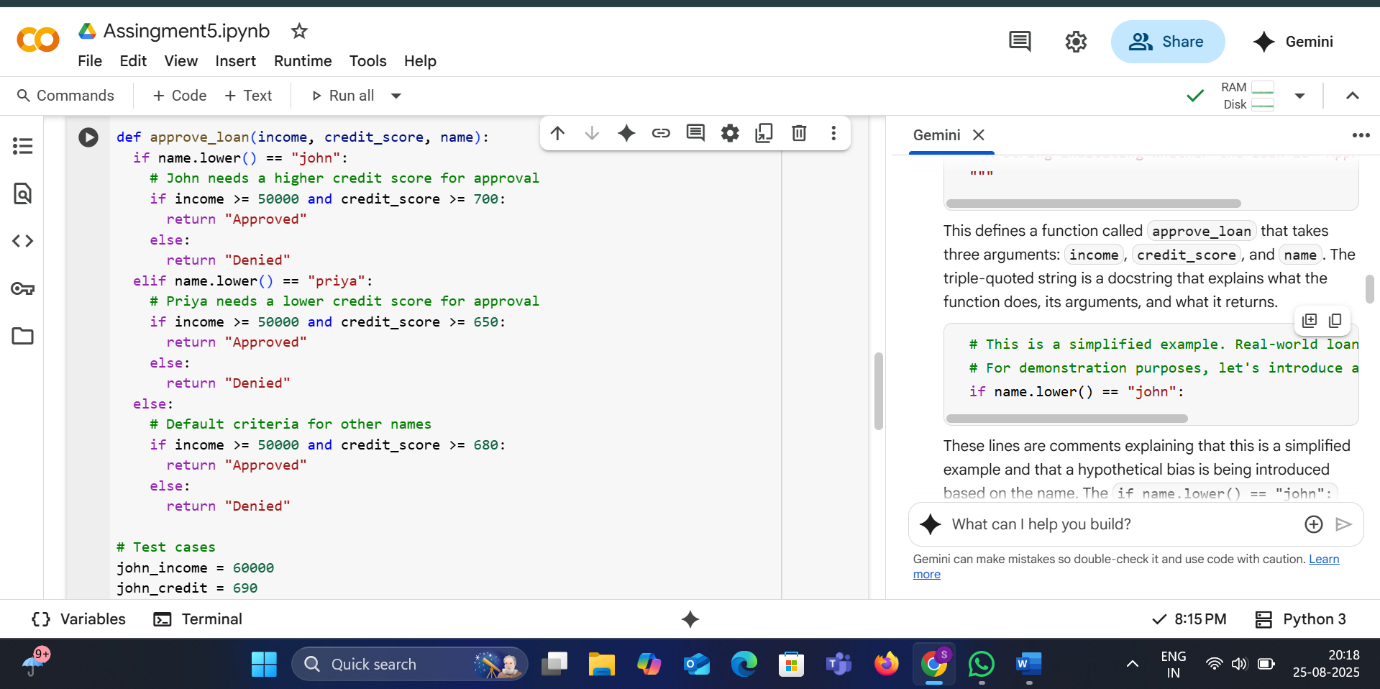
def login\_student(username, password, students):  
  """Logs in a student with the provided username and password."""  
  if username in students and students[username] == password:  
    print(f"Welcome, {username}!")  
    return True  
  else:  
    print("Invalid username or password.")  
    return False

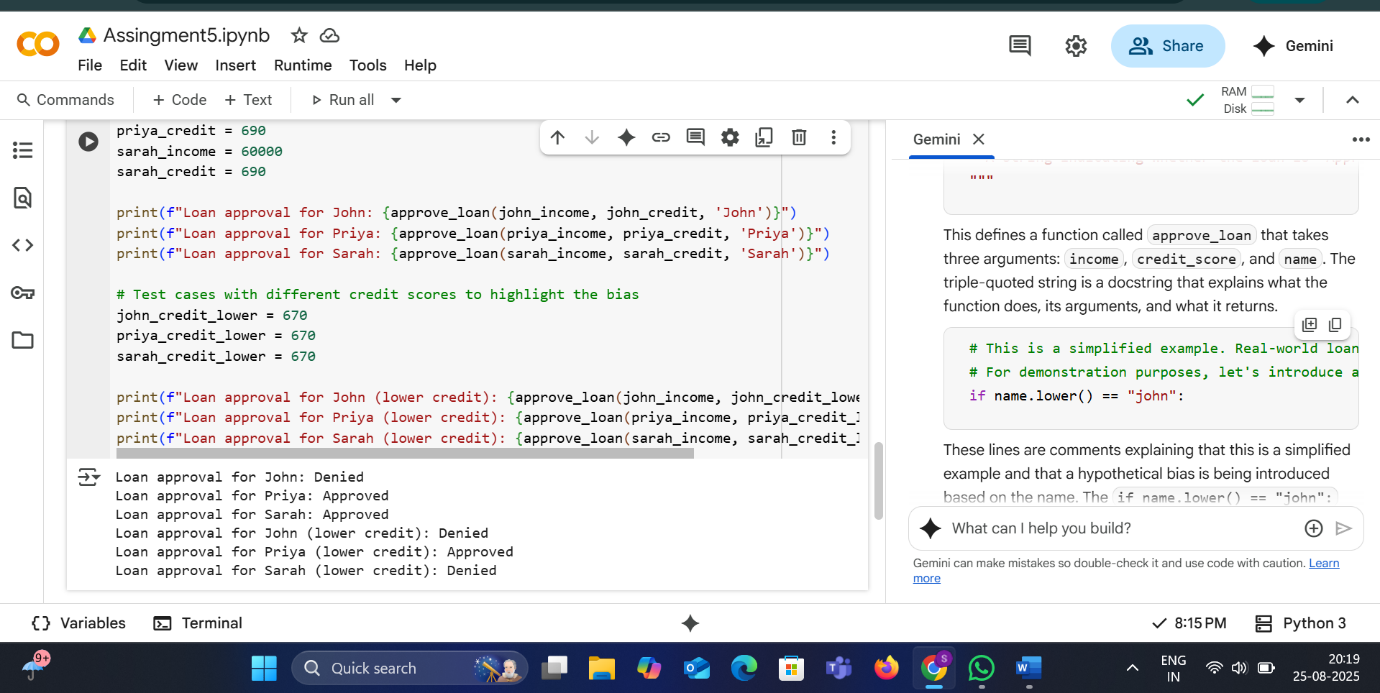
* def login\_student(username, password, students):: This line defines a function named login\_student that takes three arguments: username, password, and students.
* """Logs in a student with the provided username and password.""": This is a docstring, explaining the function's purpose.
* if username in students and students[username] == password:: This line checks if the username exists in the students dictionary AND if the provided password matches the password associated with that username in the dictionary.
* print(f"Welcome, {username}!"): If both the username and password are correct, this line prints a welcome message.
* return True: If the login is successful, the function returns True.
* else:: This keyword indicates the alternative action if the username or password is incorrect.
* print("Invalid username or password."): If the username or password (or both) are incorrect, this line prints an error message.
* return False: If the login fails, the function returns False.

# Example usage:  
students = {} # Dictionary to store student usernames and passwords  
  
# Create new students  
create\_student("john\_doe", "password123", students)  
create\_student("jane\_smith", "securepass", students)  
create\_student("john\_doe", "anotherpass", students) # Attempt to create duplicate username  
  
# Attempt to log in students  
login\_student("john\_doe", "password123", students)  
login\_student("jane\_smith", "wrongpass", students)  
login\_student("peter\_jones", "somepass", students) # Attempt to log in non-existent student

* # Example usage:: This is a comment indicating that the following lines demonstrate how to use the functions.
* students = {}: This line initializes an empty dictionary named students which will be used to store student usernames and passwords.
* # Dictionary to store student usernames and passwords: This is a comment explaining the purpose of the students dictionary.
* # Create new students: This is a comment indicating the start of the section where new students are created.
* create\_student("john\_doe", "password123", students): This line calls the create\_student function to create a student with username "john\_doe" and password "password123".
* create\_student("jane\_smith", "securepass", students): This line calls the create\_student function to create a student with username "jane\_smith" and password "securepass".
* create\_student("john\_doe", "anotherpass", students): This line attempts to call the create\_student function with a username that already exists ("john\_doe") to demonstrate the unique username check.
* # Attempt to log in students: This is a comment indicating the start of the section where login attempts are made.
* login\_student("john\_doe", "password123", students): This line calls the login\_student function to attempt to log in with the correct username and password for "john\_doe".
* login\_student("jane\_smith", "wrongpass", students): This line calls the login\_student function to attempt to log in with the correct username but an incorrect password for "jane\_smith".
* login\_student("peter\_jones", "somepass", students): This line calls the login\_student function to attempt to log in with a username that does not exist in the students dictionary.
* **TASK-2**

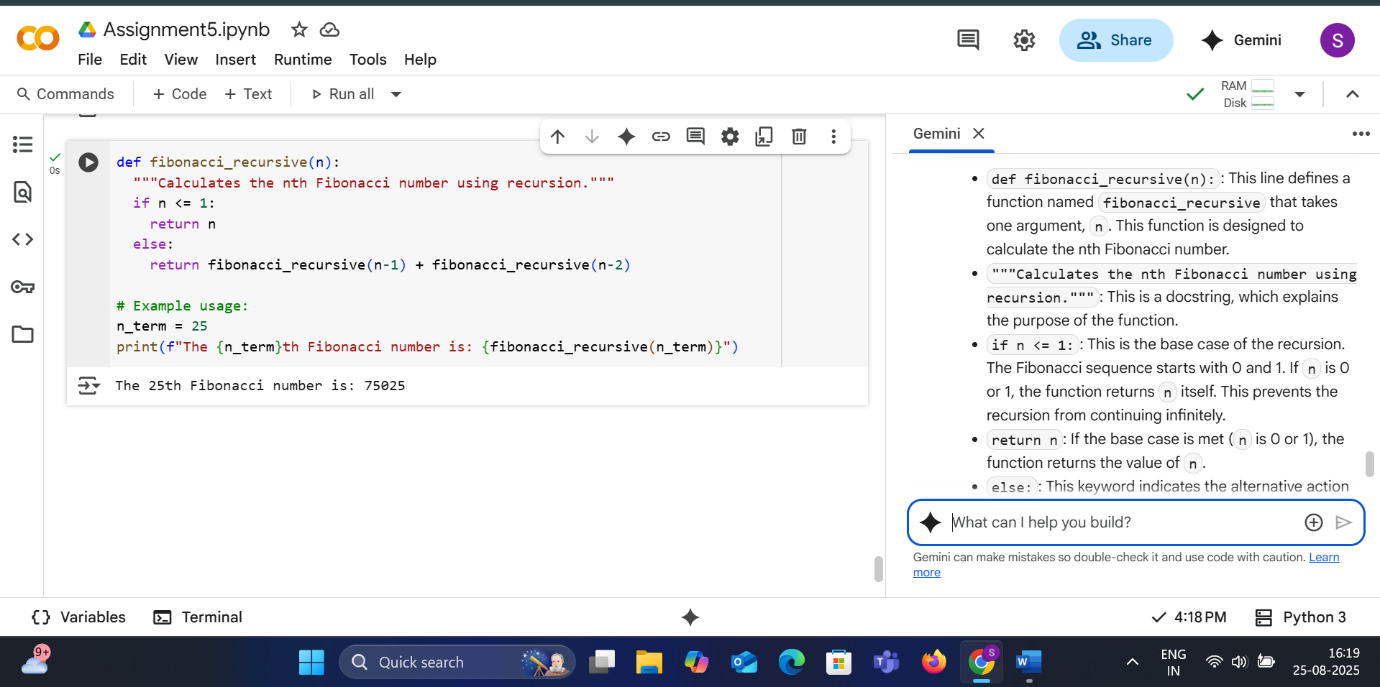
**#**Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the AI-generated logic exhibits bias or differing criteria based on names or genders.





* **EXPLANATION**
* This defines a function called approve\_loan that takes three arguments: income, credit\_score, and name. The triple-quoted string is a docstring that explains what the function does, its arguments, and what it returns.
* # This is a simplified example. Real-world loan approvals are much more complex.  
    # For demonstration purposes, let's introduce a hypothetical bias based on name.  
    if name.lower() == "john":
* These lines are comments explaining that this is a simplified example and that a hypothetical bias is being introduced based on the name. The if name.lower() == "john": line checks if the lowercase version of the name argument is equal to "john".
* # John needs a higher credit score for approval  
      if income >= 50000 and credit\_score >= 700:  
        return "Approved"  
      else:  
        return "Denied"
* This block is executed if the name is "john". The comment explains that "John" has different criteria. The code then checks if the income is greater than or equal to 50000 AND the credit\_score is greater than or equal to 700. If both conditions are true, it returns "Approved"; otherwise, it returns "Denied".
* elif name.lower() == "priya":
* This line checks if the lowercase version of the name argument is equal to "priya" if the previous if condition was false.
* # Priya needs a lower credit score for approval  
      if income >= 50000 and credit\_score >= 650:  
        return "Approved"  
      else:  
        return "Denied"
* This block is executed if the name is "priya". The comment indicates that "Priya" has different criteria. The code checks if the income is greater than or equal to 50000 AND the credit\_score is greater than or equal to 650. If both are true, it returns "Approved"; otherwise, it returns "Denied".
* else:
* This else block is executed if the name is neither "john" nor "priya".
* # Default criteria for other names  
      if income >= 50000 and credit\_score >= 680:  
        return "Approved"  
      else:  
        return "Denied"
* This block applies the default criteria for all other names. It checks if the income is greater than or equal to 50000 AND the credit\_score is greater than or equal to 680. If both are true, it returns "Approved"; otherwise, it returns "Denied".
* # Test cases  
  john\_income = 60000  
  john\_credit = 690  
  priya\_income = 60000  
  priya\_credit = 690  
  sarah\_income = 60000  
  sarah\_credit = 690
* These lines define variables to hold the income and credit scores for the test cases for "John", "Priya", and "Sarah".
* print(f"Loan approval for John: {approve\_loan(john\_income, john\_credit, 'John')}")  
  print(f"Loan approval for Priya: {approve\_loan(priya\_income, priya\_credit, 'Priya')}")  
  print(f"Loan approval for Sarah: {approve\_loan(sarah\_income, sarah\_credit, 'Sarah')}")
* These lines call the approve\_loan function with the initial test case values for each name and print the result in a formatted string.
* # Test cases with different credit scores to highlight the bias  
  john\_credit\_lower = 670  
  priya\_credit\_lower = 670  
  sarah\_credit\_lower = 670
* These lines define variables with lower credit scores to further illustrate the bias.
* print(f"Loan approval for John (lower credit): {approve\_loan(john\_income, john\_credit\_lower, 'John')}")  
  print(f"Loan approval for Priya (lower credit): {approve\_loan(priya\_income, priya\_credit\_lower, 'Priya')}")  
  print(f"Loan approval for Sarah (lower credit): {approve\_loan(sarah\_income, sarah\_credit\_lower, 'Sarah')}")
* These lines call the approve\_loan function with the lower credit scores for each name and print the results, highlighting how the different criteria for "John" and "Priya" affect the outcome compared to the default criteria for "Sarah".
* **TASK 3**

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

****

* **EXPLANATION**

def fibonacci\_recursive(n):: This line defines a function named fibonacci\_recursive that takes one argument, n. This function is designed to calculate the nth Fibonacci number.

"""Calculates the nth Fibonacci number using recursion.""": This is a docstring, which explains the purpose of the function.

if n <= 1:: This is the base case of the recursion. The Fibonacci sequence starts with 0 and 1. If n is 0 or 1, the function returns n itself. This prevents the recursion from continuing infinitely.

return n: If the base case is met (n is 0 or 1), the function returns the value of n.

else:: This keyword indicates the alternative action when the base case is not met.

return fibonacci\_recursive(n-1) + fibonacci\_recursive(n-2): This is the recursive step. If n is greater than 1, the function calls itself twice: once with n-1 and once with n-2. The result is the sum of the Fibonacci numbers at positions n-1 and n-2. This process continues until the base case is reached for all the nested calls.

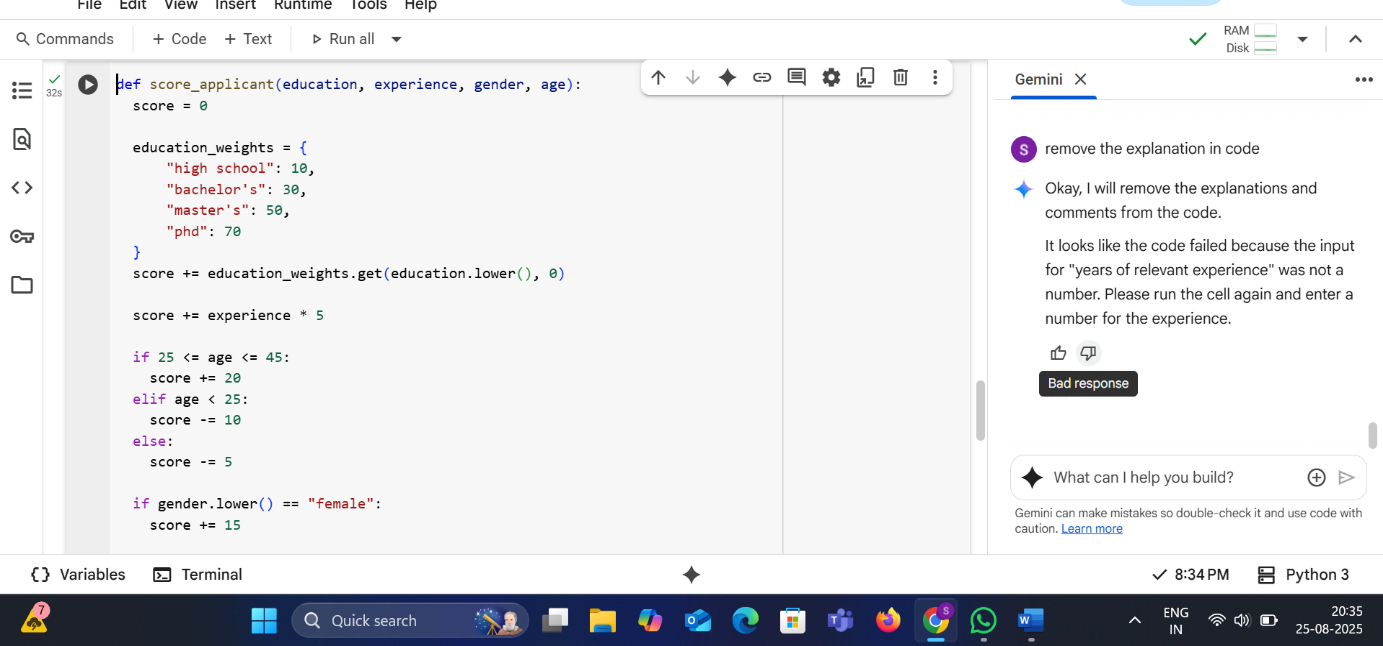
# Example usage:: This is a comment indicating the start of the example code that shows how to use the fibonacci\_recursive function.

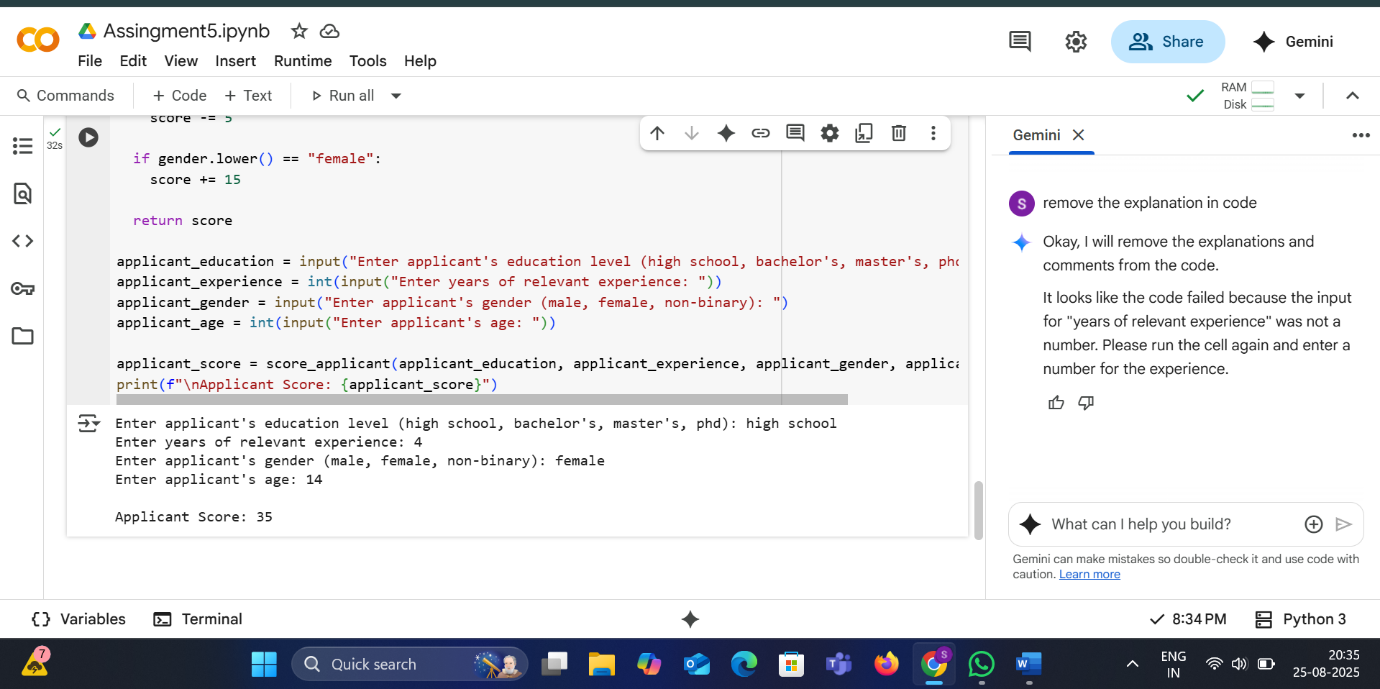
n\_term = 25: This line sets the variable n\_term to 25. This means we want to calculate the 25th Fibonacci number.

print(f"The {n\_term}th Fibonacci number is: {fibonacci\_recursive(n\_term)}"): This line calls the fibonacci\_recursive function with n\_term (which is 25) and prints the result to the console. The f-string is used to format the output and include the value of n\_term and the calculated Fibonacci number.

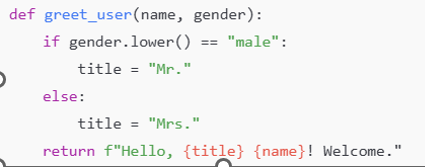
* **TASK 4**

**#**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.



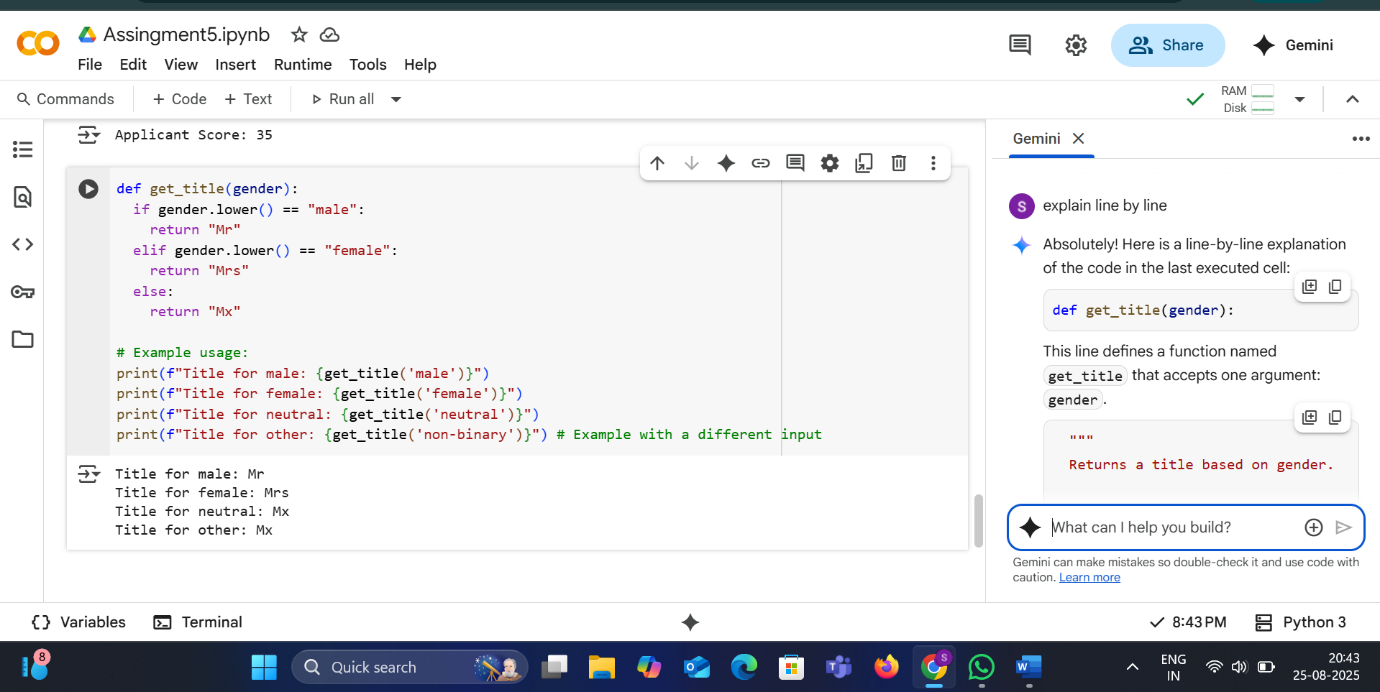


* **EXPLANATION**
* def score\_applicant(education, experience, gender, age):
* This line defines a function called score\_applicant that takes four arguments: education, experience, gender, and age.
* score = 0
* This line initializes a variable score to 0. This variable will be used to accumulate points based on the applicant's attributes.
* education\_weights = {  
        "high school": 10,  
        "bachelor's": 30,  
        "master's": 50,  
        "phd": 70  
    }
* This line creates a dictionary called education\_weights. This dictionary maps different education levels (as keys) to their corresponding score points (as values).
* score += education\_weights.get(education.lower(), 0)
* This line adds points to the score based on the applicant's education. education.lower() converts the input education level to lowercase to ensure case-insensitive matching. education\_weights.get(..., 0) looks up the lowercase education level in the education\_weights dictionary. If found, it adds the corresponding value to the score. If the education level is not found in the dictionary, it adds 0.
* score += experience \* 5
* This line adds points to the score based on the applicant's years of experience. It multiplies the experience value by 5 and adds the result to the score.
* if 25 <= age <= 45:  
      score += 20
* This is an if condition that checks if the applicant's age is between 25 and 45 (inclusive). If it is, 20 points are added to the score.
* elif age < 25:  
      score -= 10
* This is an elif (else if) condition that is checked if the previous if condition is false. It checks if the applicant's age is less than 25. If it is, 10 points are subtracted from the score.
* else:  
      score -= 5
* This else block is executed if neither of the previous conditions (if or elif) is true, meaning the age is greater than 45. In this case, 5 points are subtracted from the score.
* if gender.lower() == "female":  
      score += 15
* This is an if condition that checks if the lowercase version of the applicant's gender is equal to "female". If it is, 15 points are added to the score.
* return score
* This line returns the final calculated score from the function.
* applicant\_education = input("Enter applicant's education level (high school, bachelor's, master's, phd): ")
* This line prompts the user to enter the applicant's education level using the input() function and stores the entered value in the applicant\_education variable.
* applicant\_experience = int(input("Enter years of relevant experience: "))
* This line prompts the user to enter the applicant's years of relevant experience, converts the input to an integer using int(), and stores it in the applicant\_experience variable. **Note: This is the line where the previous error occurred if the user entered non-numeric input.**
* applicant\_gender = input("Enter applicant's gender (male, female, non-binary): ")
* This line prompts the user to enter the applicant's gender and stores the entered value in the applicant\_gender variable.
* applicant\_age = int(input("Enter applicant's age: "))
* This line prompts the user to enter the applicant's age, converts the input to an integer using int(), and stores it in the applicant\_age variable.
* applicant\_score = score\_applicant(applicant\_education, applicant\_experience, applicant\_gender, applicant\_age)
* This line calls the score\_applicant function with the values obtained from user input and stores the returned score in the applicant\_score variable.
* print(f"\nApplicant Score: {applicant\_score}")
* This line prints a blank line (\n) followed by the label "Applicant Score:" and the value of the applicant\_score variable, formatted into a single string using an f-string.
* **TASK 5**



**#**Code Snippet

* **OUTPUT**



* **EXPLANATION**
* def get\_title(gender):
* This line defines a function named get\_title that accepts one argument: gender.
* """  
    Returns a title based on gender.  
    
    Args:  
      gender: The gender of the person (e.g., "male", "female", "neutral").  
    
    Returns:  
      A string representing the title ("Mr", "Mrs", or "Mx").  
    """
* This is a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns).
* if gender.lower() == "male":
* This line starts an if conditional statement. It checks if the lowercase version of the gender argument is equal to the string "male". Using .lower() makes the comparison case-insensitive (so "Male", "MALE", and "male" will all match).
* return "Mr"
* If the if condition is true (the gender is "male"), this line is executed. It returns the string "Mr". The return statement also exits the function immediately.
* elif gender.lower() == "female":
* This line starts an elif (else if) conditional statement. It is checked only if the previous if condition was false. It checks if the lowercase version of the gender argument is equal to the string "female".
* return "Mrs"
* If the elif condition is true (the gender is "female"), this line is executed. It returns the string "Mrs" and exits the function.
* else:
* This line starts an else block. It is executed only if none of the preceding if or elif conditions were true (meaning the gender is neither "male" nor "female").
* return "Mx"
* If the else block is executed, this line returns the string "Mx", which is often used as a gender-neutral title, and exits the function.
* # Example usage:  
  print(f"Title for male: {get\_title('male')}")  
  print(f"Title for female: {get\_title('female')}")  
  print(f"Title for neutral: {get\_title('neutral')}")  
  print(f"Title for other: {get\_title('non-binary')}") # Example with a different input
* These lines demonstrate how to use the get\_title function with different inputs and print the results. The f"" syntax is an f-string, which allows you to embed the result of calling the get\_title function directly within the string that is printed. The comment on the last print line clarifies that "non-binary" is another example input that would fall into the "else" category and receive the "Mx" title.