HTNO:2403A52134

Assignment-5

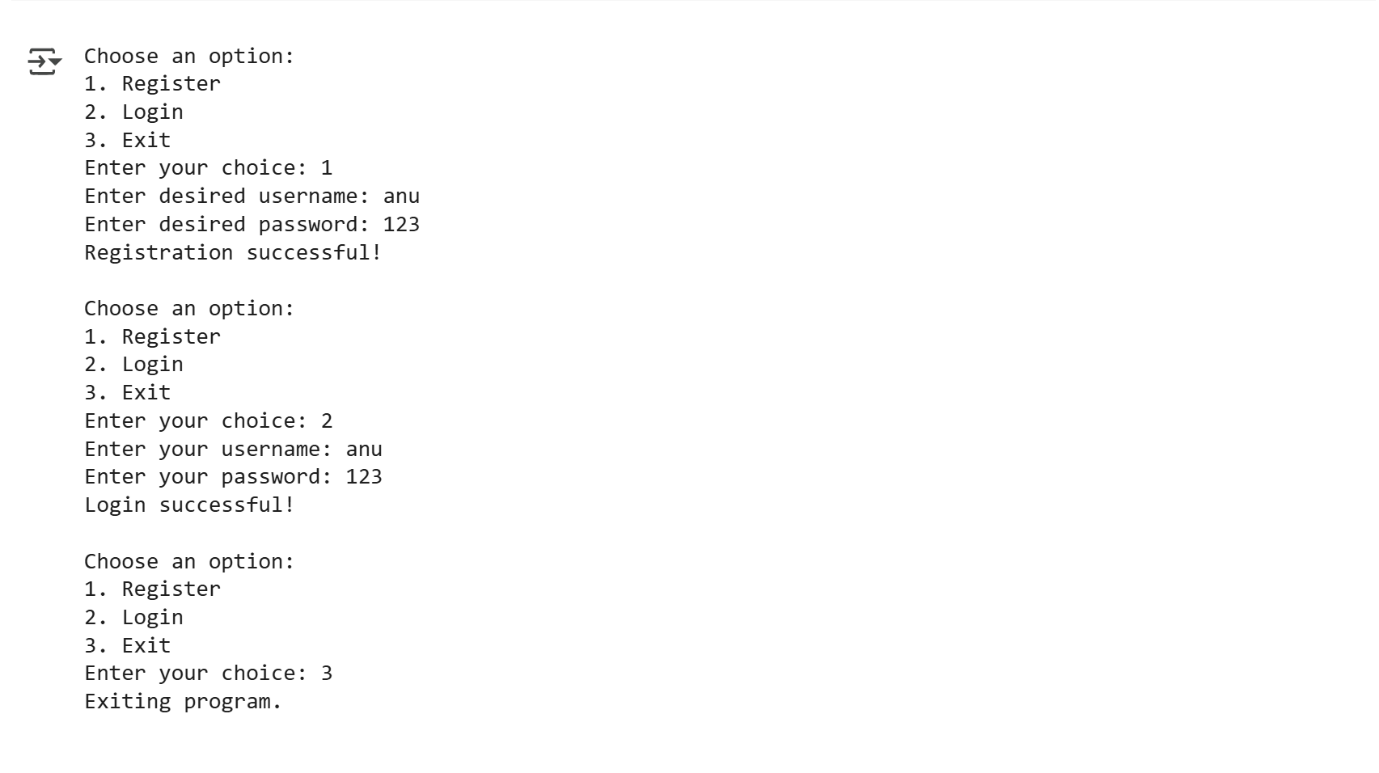
Task-1:

To generate a login system.

Code:



Output:



Explanation:

This Python program creates a simple command-line student login system. Here's how it works:

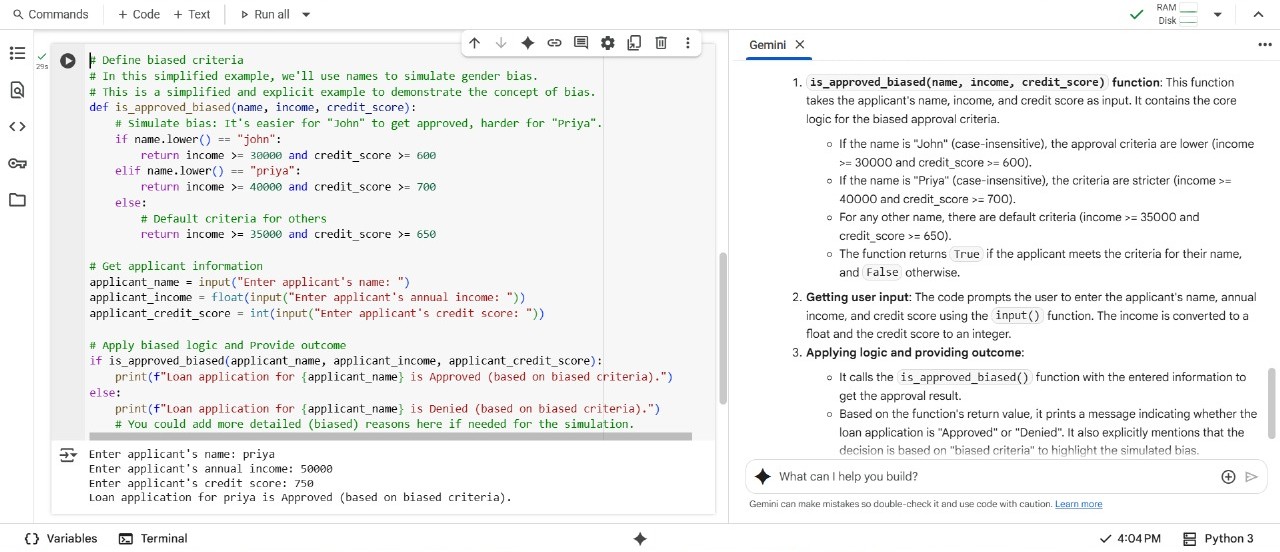
1. **Storing Credentials**: The program uses a dictionary called student\_credentials to store the usernames and passwords of registered students. Initially, this dictionary is empty.
2. **Registration**: There's a function designed for new students to register. When you choose to register, the program asks for a desired username and password. It then checks if the username already exists in the student\_credentials dictionary.
   * If the username is already taken, it prints a message saying so.
   * If the username is available, it adds the new username and password as a key-value pair to the student\_credentials dictionary and confirms that the registration was successful.
3. **Login**: There's another function for existing students to log in. When you choose to log in, the program prompts for your username and password. It then checks if the entered username exists in the student\_credentials dictionary.
   * If the username is found, it then compares the entered password with the password stored for that username. If they match, it prints "Login successful!".
   * If the username is not found, or if the password doesn't match, it prints "Invalid username or password."
4. **User Interface**: The main part of the program runs a loop that presents a menu to the user with three options: Register, Login, or Exit.
   * It waits for the user to enter their choice (1, 2, or 3).
   * Based on the choice, it calls either the registration function, the login function, or breaks out of the loop to exit the program.
   * If the user enters an invalid choice, it prompts them to try again.
5. **Displaying Credentials**: After the main interface loop finishes (if you choose to exit), the program can display the current contents of the student\_credentials dictionary, showing all the usernames and passwords that were registered during that run of the program.

In essence, the program takes user input for registration and login, stores the credentials temporarily in a dictionary, and provides a simple menu for users to interact with these functionalities.

Task-2:

To generate logic exhibits bias or differing criteria based on names or genders.

Code and Output:



Explanation:

# Create new sample applicants with John and Priya  
applicants\_john\_priya = [  
    LoanApplicant(income=75000, credit\_score=720, gender='Male', name='John'),  
    LoanApplicant(income=58000, credit\_score=680, gender='Female', name='Priya'),  
]

This code block creates a Python list named applicants\_john\_priya. This list contains two LoanApplicant objects. Each LoanApplicant object is created by calling the LoanApplicant class (defined in a previous cell) and providing values for income, credit\_score, gender, and name. This essentially sets up data for two new loan applicants, John and Priya.

# Apply the existing approval logic to the new applicants  
approval\_results\_john\_priya = {}

This line initializes an empty Python dictionary called approval\_results\_john\_priya. This dictionary will be used to store the loan approval results for the applicants in the applicants\_john\_priya list. The keys of the dictionary will be the applicant names, and the values will be their approval status (True for approved, False for denied).

for applicant in applicants\_john\_priya:  
    is\_approved = determine\_loan\_approval(applicant)  
    approval\_results\_john\_priya[applicant.name] = is\_approved

This is a for loop that iterates through each applicant object in the applicants\_john\_priya list. Inside the loop:

is\_approved = determine\_loan\_approval(applicant): This line calls the determine\_loan\_approval function (defined in a previous cell), pas# Create new sample applicants with John and Priya  
applicants\_john\_priya = [  
    LoanApplicant(income=75000, credit\_score=720, gender='Male', name='John'),  
    LoanApplicant(income=58000, credit\_score=680, gender='Female', name='Priya'),  
]

This code block creates a Python list named applicants\_john\_priya. This list contains two LoanApplicant objects. Each LoanApplicant object is created by calling the LoanApplicant class (defined in a previous cell) and providing values for income, credit\_score, gender, and name. This essentially sets up data for two new loan applicants, John and Priya.

# Apply the existing approval logic to the new applicants  
approval\_results\_john\_priya = {}

This line initializes an empty Python dictionary called approval\_results\_john\_priya. This dictionary will be used to store the loan approval results for the applicants in the applicants\_john\_priya list. The keys of the dictionary will be the applicant names, and the values will be their approval status (True for approved, False for denied).

for applicant in applicants\_john\_priya:  
    is\_approved = determine\_loan\_approval(applicant)  
    approval\_results\_john\_priya[applicant.name] = is\_approved

This is a for loop that iterates through each applicant object in the applicants\_john\_priya list. Inside the loop:

* is\_approved = determine\_loan\_approval(applicant): This line calls the determine\_loan\_approval function (defined in a previous cell), passing the current applicant object as an argument. The function checks if the applicant meets the loan approval criteria (minimum income and credit score) and returns True if approved, False otherwise. The returned boolean value is stored in the is\_approved variable.
* approval\_results\_john\_priya[applicant.name] = is\_approved: This line adds an entry to the approval\_results\_john\_priya dictionary. The key is the name of the current applicant, and the value is the is\_approved status for that applicant.

# Display the approval results for John and Priya  
print("Loan Approval Results for John and Priya:")

This line prints a header "Loan Approval Results for John and Priya:" to the console, making the output more readable.

for name, approved in approval\_results\_john\_priya.items():  
    print(f"{name}: {'Approved' if approved else 'Denied'}")

This is another for loop that iterates through the key-value pairs in the approval\_results\_john\_priya dictionary. The .items() method returns a view object that displays a list of a dictionary's key-value tuple pairs. Inside the loop:

* name gets the key (the applicant's name).
* approved gets the value (the boolean approval status).
* print(f"{name}: {'Approved' if approved else 'Denied'}"): This line prints the applicant's name followed by their approval status. It uses an f-string for formatting. The expression {'Approved' if approved else 'Denied'} is a conditional expression that prints "Approved" if the approved variable is True, and "Denied" if it is False.

Let me know if you would like me to explain any othe

* sing the current applicant object as an argument. The function checks if the applicant meets the loan approval criteria (minimum income and credit score) and returns True if approved, False otherwise. The returned boolean value is stored in the is\_approved variable.
* approval\_results\_john\_priya[applicant.name] = is\_approved: This line adds an entry to the approval\_results\_john\_priya dictionary. The key is the name of the current applicant, and the value is the is\_approved status for that applicant.

# Display the approval results for John and Priya  
print("Loan Approval Results for John and Priya:")

This line prints a header "Loan Approval Results for John and Priya:" to the console, making the output more readable.

for name, approved in approval\_results\_john\_priya.items():  
    print(f"{name}: {'Approved' if approved else 'Denied'}")

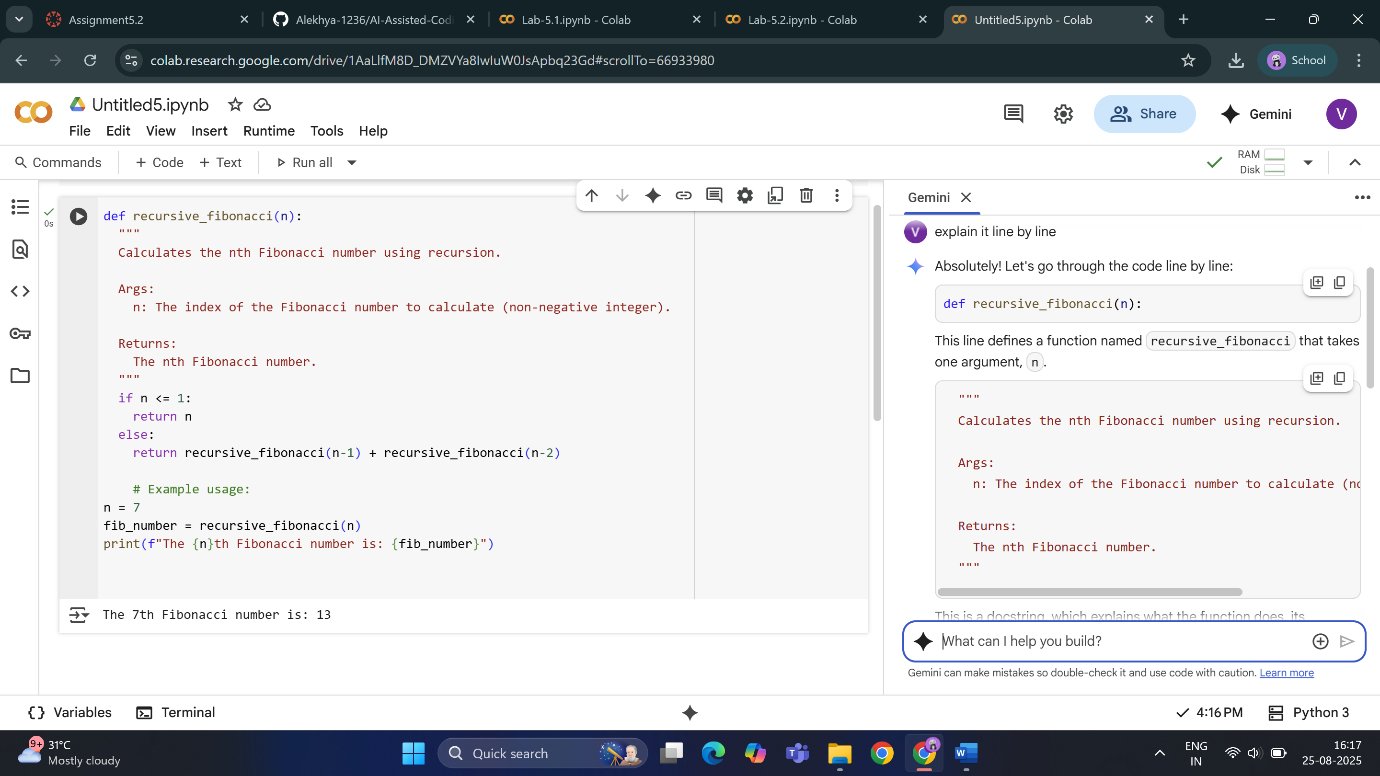
This is another for loop that iterates through the key-value pairs in the approval\_results\_john\_priya dictionary. The .items() method returns a view object that displays a list of a dictionary's key-value tuple pairs. Inside the loop:

* name gets the key (the applicant's name).
* approved gets the value (the boolean approval status).
* print(f"{name}: {'Approved' if approved else 'Denied'}"): This line prints the applicant's name followed by their approval status. It uses an f-string for formatting. The expression {'Approved' if approved else 'Denied'} is a conditional expression that prints "Approved" if the approved variable is True, and "Denied" if it is False.

Task-3:

To calculate the nth Fibonacci number using recursion.

Code and Output:



Explanation:

def recursive\_fibonacci(n):

This line defines a function named recursive\_fibonacci that takes one argument, 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.  
  """

This is a docstring, which explains what the function does, its arguments, and what it returns. It's good practice to include these for clarity.

  if n <= 1:  
    return n

This is the base case for the recursion. The Fibonacci sequence starts with 0 and 1. If n is 0 or 1, the function returns n directly, stopping the recursion.

  else:  
    return recursive\_fibonacci(n-1) + recursive\_fibonacci(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. It then returns the sum of the results of these two calls. This is how the Fibonacci sequence is generated (each number is the sum of the two preceding ones).

    # Example usage:  
n = 7

This line sets the variable n to 7. This is the input for which we want to calculate the Fibonacci number.

fib\_number = recursive\_fibonacci(n)

This line calls the recursive\_fibonacci function with the value of n (which is 7) and stores the returned Fibonacci number in the variable fib\_number.

print(f"The {n}th Fibonacci number is: {fib\_number}")

This line prints the result to the console, using an f-string to include the values of n and fib\_number in the output message.

Task-4:

To generate a job applicant scoring system based on input features (e.g., education, experience, gender, age).

Code:

# 1. Represent Applicant Data

class Applicant:

    def \_\_init\_\_(self, education, experience, gender, age):

        self.education = education  # e.g., 'High School', 'Bachelor', 'Master', 'PhD'

        self.experience = experience # in years

        self.gender = gender      # e.g., 'Male', 'Female', 'Other'

        self.age = age          # in years

# 2. Define Scoring Logic

# This is a simple example; you can adjust the weights and scoring based on your needs.

education\_scores = {

    'High School': 1,

    'Bachelor': 3,

    'Master': 5,

    'PhD': 7

}

# Scoring based on experience (linear scale)

experience\_weight = 2

# Scoring based on age (you might penalize very young or very old applicants)

# This is a simple example: points for a certain age range

def score\_age(age):

    if 25 <= age <= 45:

        return 5

    elif 20 <= age < 25 or 45 < age <= 55:

        return 3

    else:

        return 1

# Gender could be neutral or have a slight bias depending on the job (be mindful of fairness and bias)

# In this example, gender is neutral

gender\_scores = {

    'Male': 0,

    'Female': 0,

    'Other': 0

}

# 3. Implement Scoring Function

def calculate\_score(applicant):

    score = 0

    # Add score for education

    score += education\_scores.get(applicant.education, 0)

    # Add score for experience

    score += applicant.experience \* experience\_weight

    # Add score for age

    score += score\_age(applicant.age)

    # Add score for gender (neutral in this case)

    score +=

gender\_scores.get(applicant.gender, 0)

    return score

# 4. Apply Scoring to Applicants

# Create some sample applicants

applicant1 = Applicant("Bachelor", 5, "Female", 30)

applicant2 = Applicant("Master", 10, "Male", 40)

applicant3 = Applicant("High School", 2, "Other", 22)

applicant4 = Applicant("PhD", 15, "Female", 50)

applicants = [applicant1, applicant2, applicant3, applicant4]

# Calculate and store scores

applicant\_scores = {}

for i, applicant in enumerate(applicants):

    score = calculate\_score(applicant)

    applicant\_scores[f"Applicant {i+1}"] = score

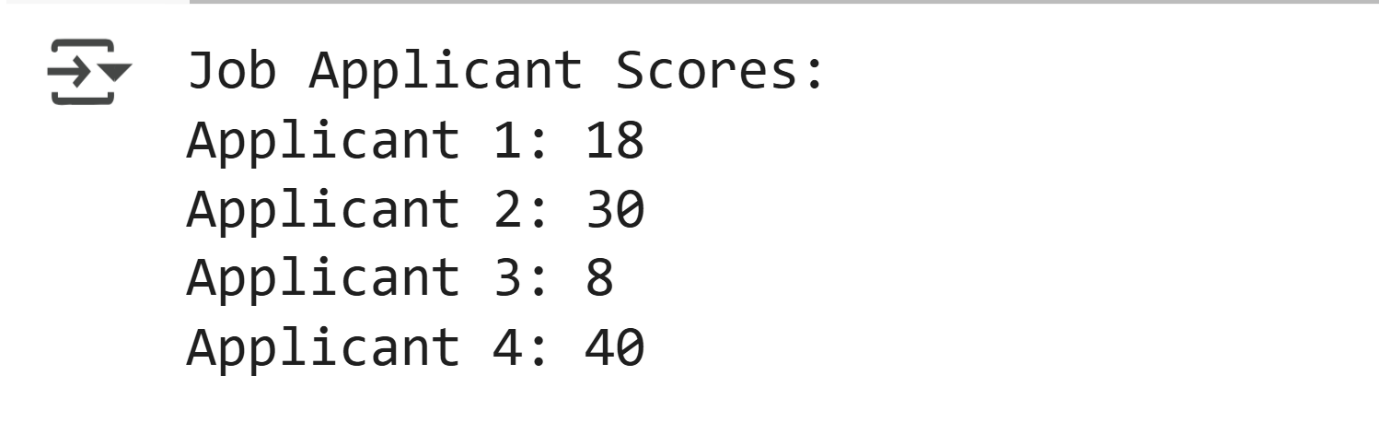
# 5. Display Results

print("Job Applicant Scores:")

for applicant, score in applicant\_scores.items():

    print(f"{applicant}: {score}")

Output:



Explanation:

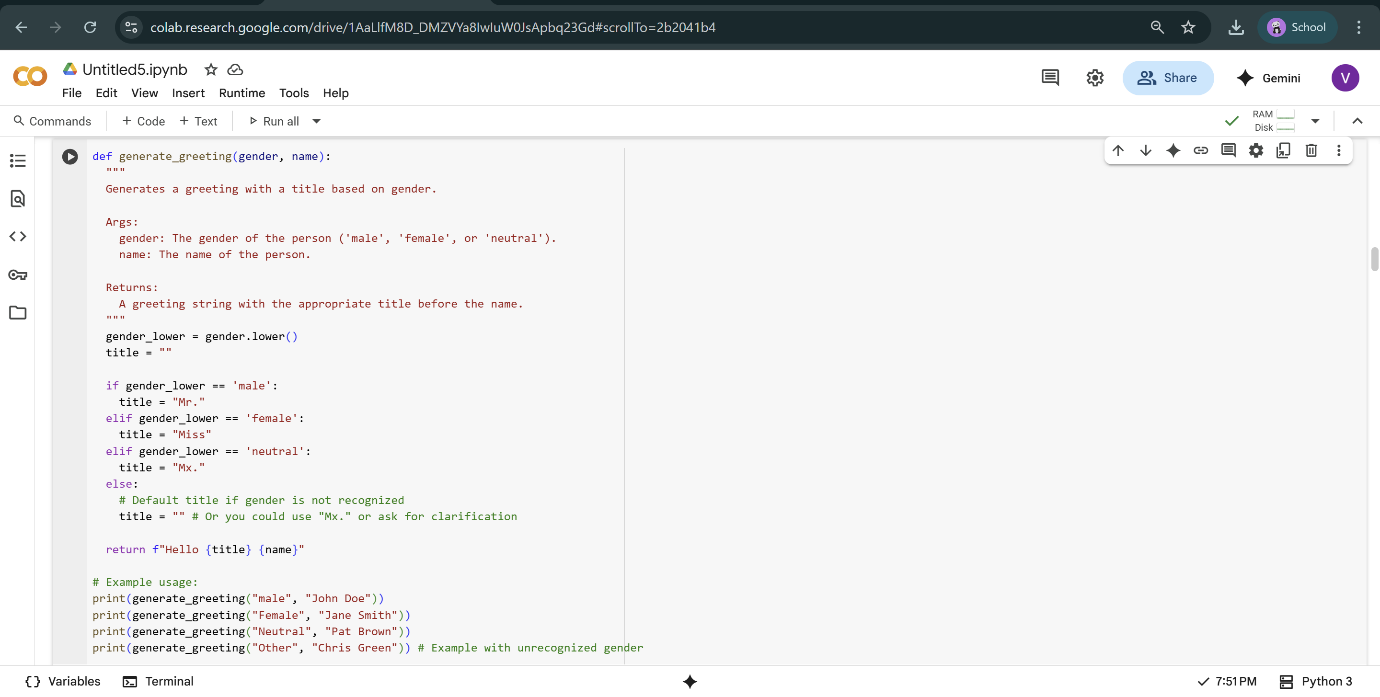
Here's a breakdown of the code:

* **Representing Applicant Data:**
  + A class named Applicant is defined to represent each job applicant.
  + The constructor (\_\_init\_\_) of the class takes education, experience, gender, and age as input.
  + These input values are stored as attributes of the Applicant object.
* **Defining Scoring Logic:**
  + A dictionary education\_scores is created to assign numerical scores to different education levels (e.g., High School, Bachelor, Master, PhD).
  + An experience\_weight variable is set to 2, which will be used to weight the years of experience in the scoring.
  + A function score\_age is defined to assign scores based on age ranges, giving higher scores to ages within a certain preferred range (25-45).
  + A dictionary gender\_scores is created to assign scores based on gender. In this specific example, all genders are assigned a score of 0, making gender neutral in the scoring.
* **Implementing Scoring Function:**
  + A function calculate\_score is defined to take an Applicant object and compute a total score.
  + It initializes a score variable to 0.
  + It adds the score for education by looking up the applicant's education level in the education\_scores dictionary.
  + It adds the score for experience by multiplying the applicant's experience by the experience\_weight.
  + It calls the score\_age function with the applicant's age and adds the returned score.
  + It adds the score for gender from the gender\_scores dictionary.
  + Finally, it returns the calculated total score.
* **Applying Scoring to Applicants:**
  + Several sample Applicant objects are created with different features.
  + These sample applicant objects are stored in a list called applicants.
  + An empty dictionary applicant\_scores is created to hold the results.
  + A loop iterates through the applicants list, calculates the score for each applicant using the calculate\_score function, and stores the score in the applicant\_scores dictionary with a key like "Applicant 1", "Applicant 2", etc.
* **Displaying Results:**
  + A header "Job Applicant Scores:" is printed.
  + Another loop iterates through the applicant\_scores dictionary and prints each applicant's identifier and their calculated score.

Task-5:

To generate code that includes **gender-neutral**.

Code:



Output:



Explanation:

* The code defines a function called recursive\_fibonacci that takes one input, a number n.
* It includes a description of what the function does (calculates the nth Fibonacci number recursively), its input (n), and what it returns (the nth Fibonacci number).
* Inside the function, it first checks if the input number n is less than or equal to 1.
* If n is less than or equal to 1, the function directly returns the value of n. This is the base case for the recursion.
* If n is greater than 1, the code enters the else block.
* Inside the else block, the function returns the sum of calling itself with n-1 and calling itself again with n-2. This is the recursive step where the function calls itself with smaller inputs.
* After the function definition, the code sets a variable n to the value 7 as an example.
* It then calls the recursive\_fibonacci function with this value of n (which is 7) and stores the result in a variable called fib\_number.
* Finally, it prints a formatted string that shows the value of n and the calculated fib\_number, stating "The [n]th Fibonacci number is: [fib\_number]".