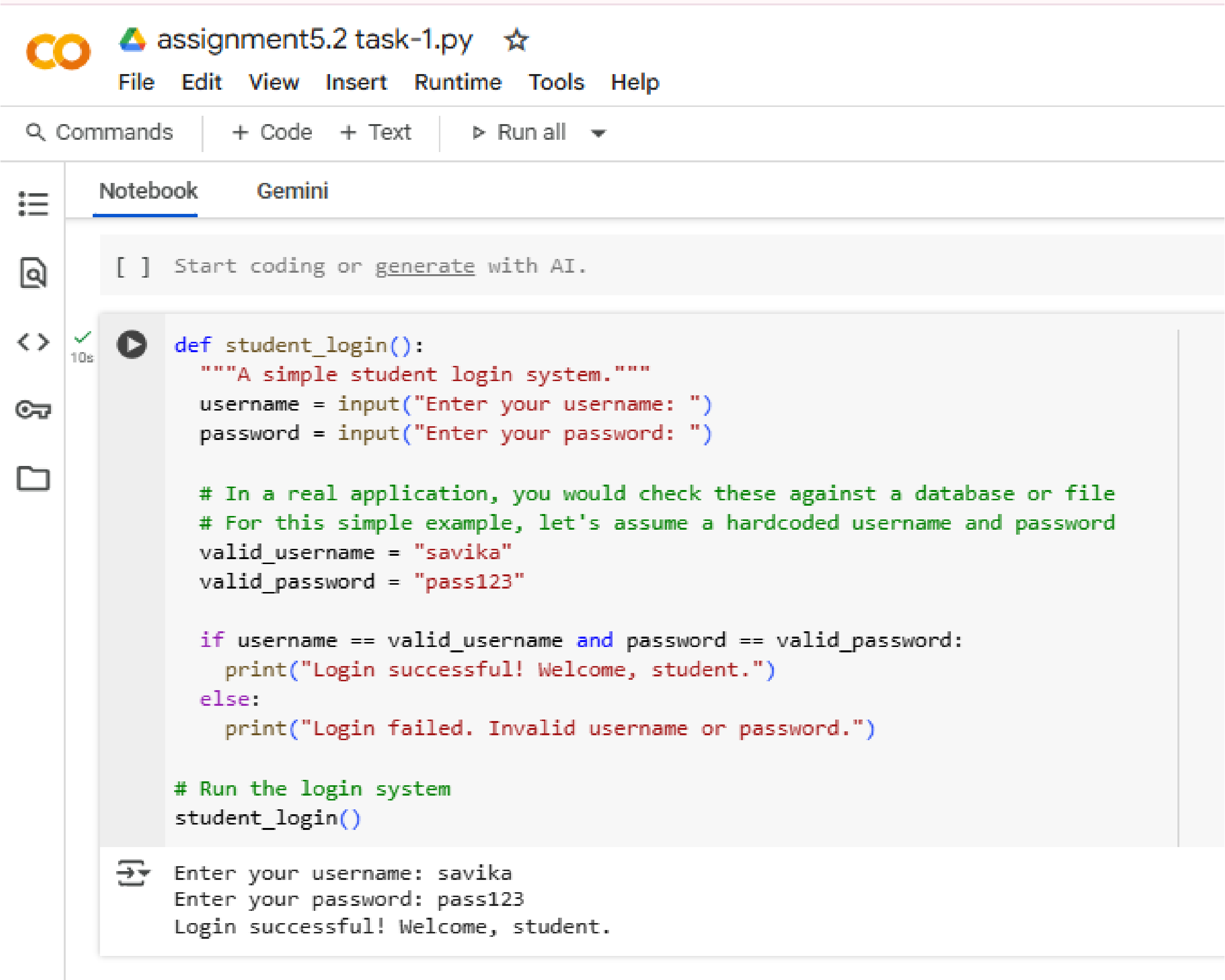
AI ASSIGNMENT: 5.2

TASK-1:

Prompt:

#write python program to develop a simple student login system that takes input username and password.

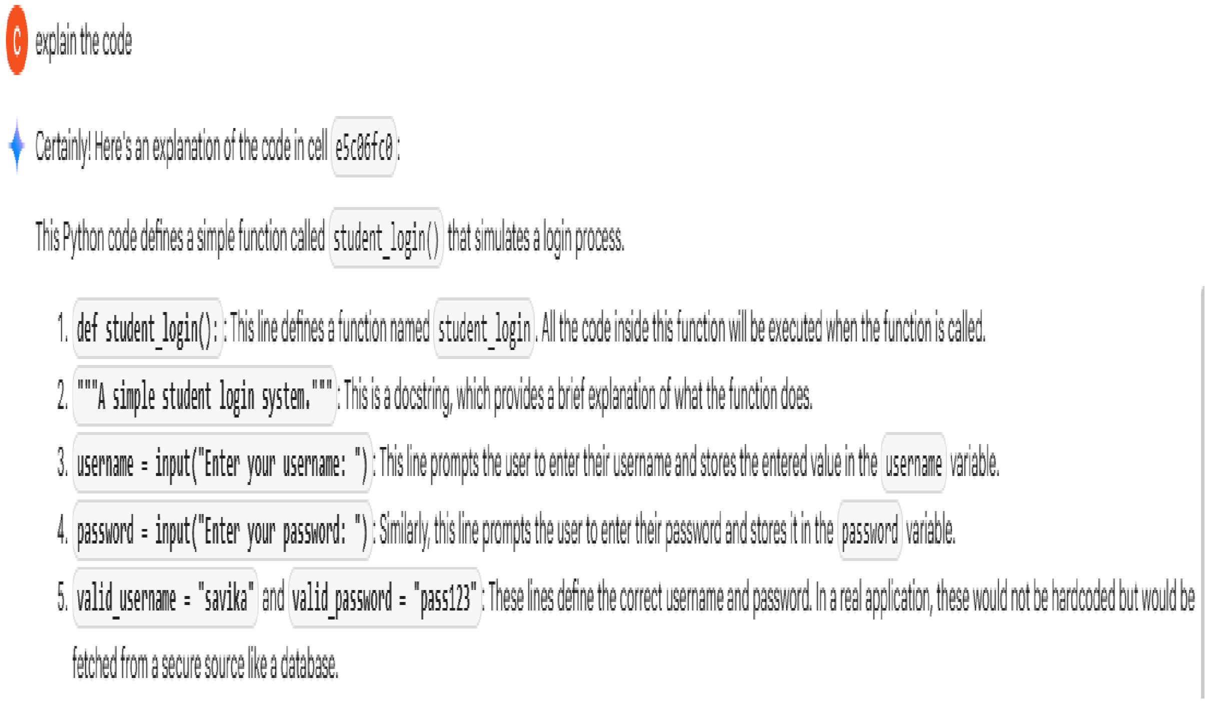
Code with Output:



```python
def student_login():
  """A simple student login system."""
  username = input("Enter your username: ")
  password = input("Enter your password: ")

  # In a real application, you would check these against a database or file
  # For this simple example, let's assume a hardcoded username and password
  valid_username = "savika"
  valid_password = "pass123"

  if username == valid_username and password == valid_password:
    print("Login successful! Welcome, student.")
  else:
    print("Login failed. Invalid username or password.")

# Run the login system
student_login()
```

```
Enter your username: savika
Enter your password: pass123
Login successful! Welcome, student.
```

Explanation:

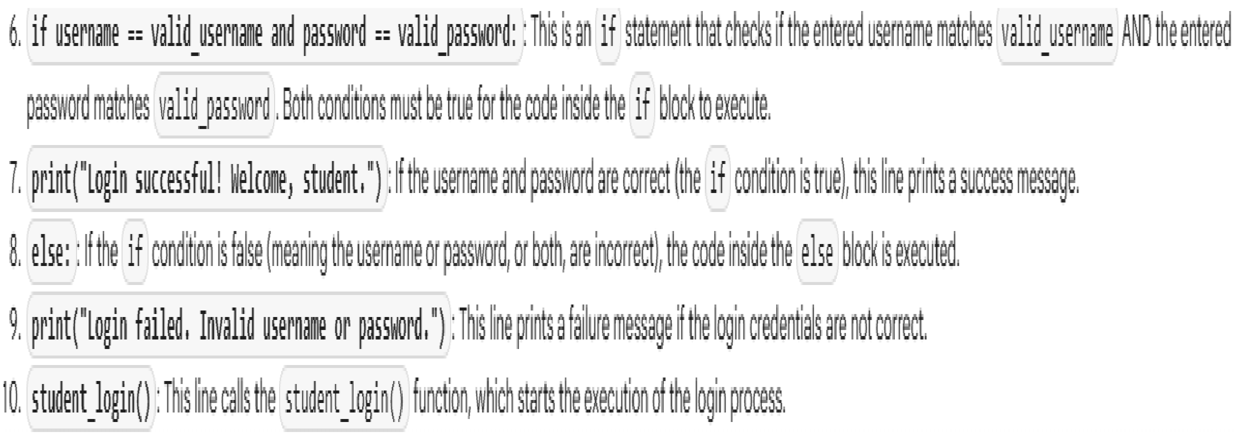


explain the code

Certainly! Here's an explanation of the code in cell `e5c06fc0`:

This Python code defines a simple function called `student_login()` that simulates a login process.

1. `def student_login():`: This line defines a function named `student_login`. All the code inside this function will be executed when the function is called.
2. `"""A simple student login system."""`: This is a docstring, which provides a brief explanation of what the function does.
3. `username = input("Enter your username: ")`: This line prompts the user to enter their username and stores the entered value in the `username` variable.
4. `password = input("Enter your password: ")`: Similarly, this line prompts the user to enter their password and stores it in the `password` variable.
5. `valid_username = "savika"` and `valid_password = "pass123"`: These lines define the correct username and password. In a real application, these would not be hardcoded but would be fetched from a secure source like a database.



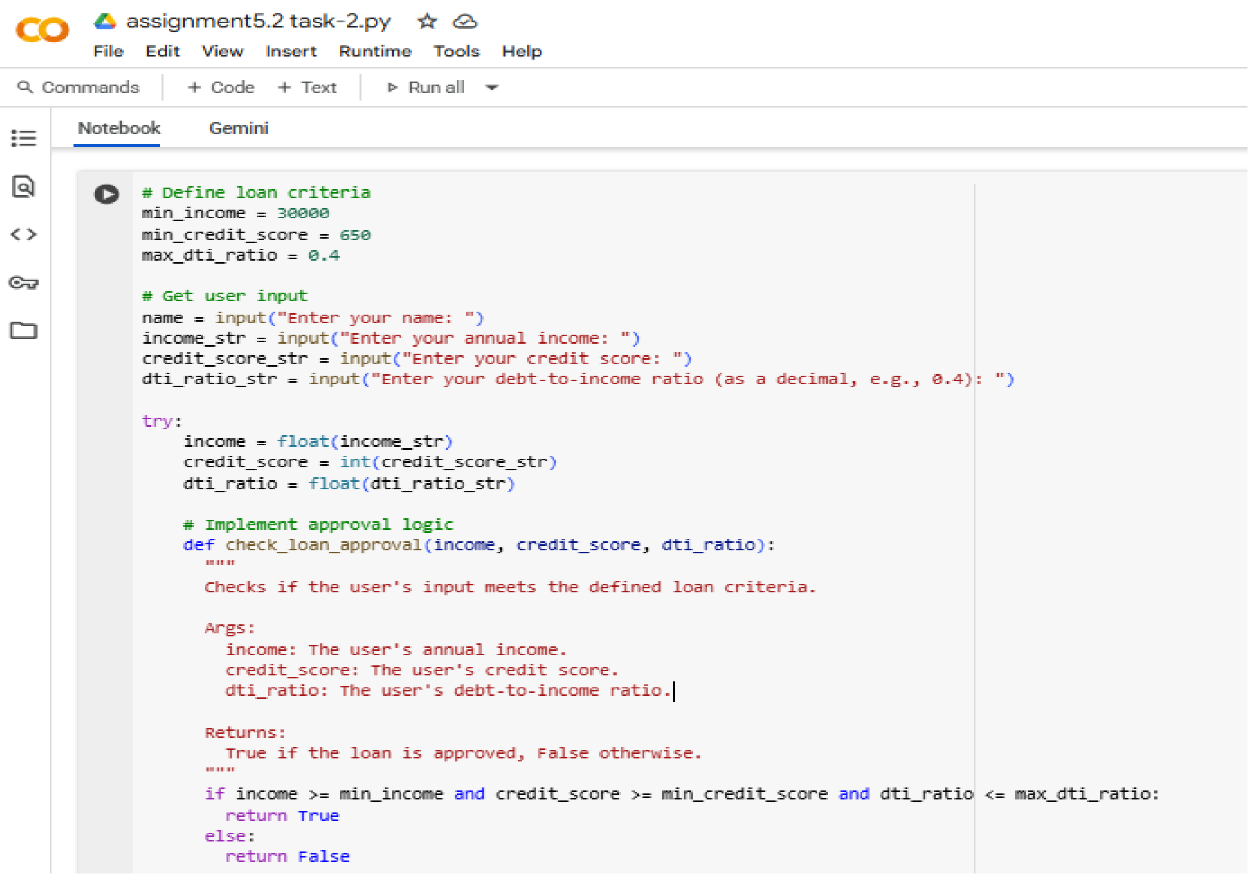
6. `if username == valid_username and password == valid_password:`: This is an `if` statement that checks if the entered username matches `valid_username` AND the entered password matches `valid_password`. Both conditions must be true for the code inside the `if` block to execute.
7. `print("Login successful! Welcome, student.")`: If the username and password are correct (the `if` condition is true), this line prints a success message.
8. `else:`: If the `if` condition is false (meaning the username or password, or both, are incorrect), the code inside the `else` block is executed.
9. `print("Login failed. Invalid username or password.")`: This line prints a failure message if the login credentials are not correct.
10. `student_login()`: This line calls the `student_login()` function, which starts the execution of the login process.

TASK-2:

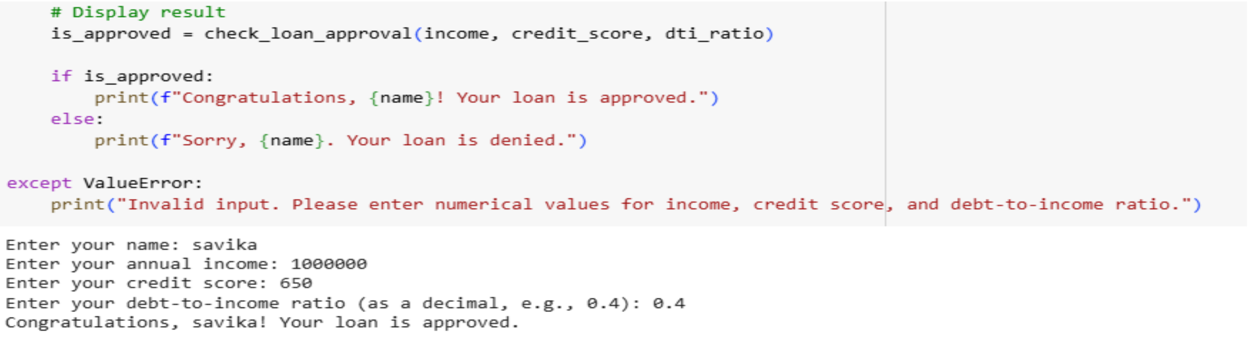
Prompt:

#write a python program to develop a simple loan approval system with name, income, credit score, dept to income ratio.

Code with Output:



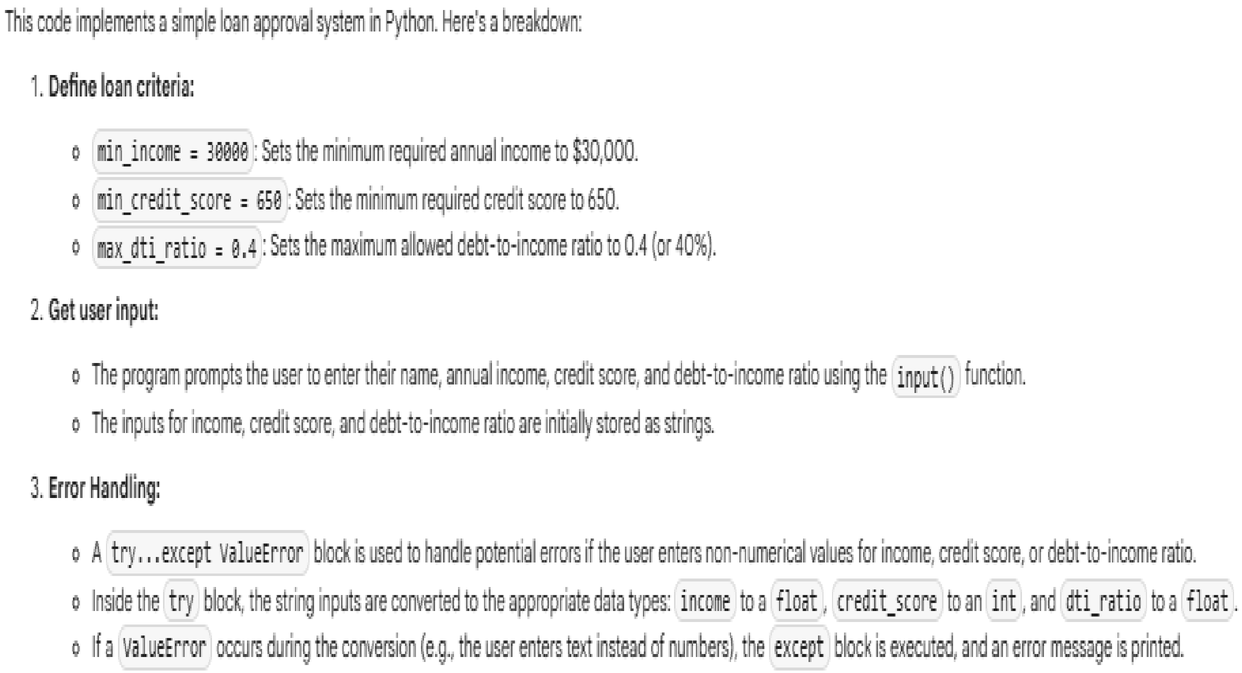
```python
# Define loan criteria
min_income = 30000
min_credit_score = 650
max_dti_ratio = 0.4

# Get user input
name = input("Enter your name: ")
income_str = input("Enter your annual income: ")
credit_score_str = input("Enter your credit score: ")
dti_ratio_str = input("Enter your debt-to-income ratio (as a decimal, e.g., 0.4): ")

try:
    income = float(income_str)
    credit_score = int(credit_score_str)
    dti_ratio = float(dti_ratio_str)

    # Implement approval logic
    def check_loan_approval(income, credit_score, dti_ratio):
      """
      Checks if the user's input meets the defined loan criteria.

      Args:
        income: The user's annual income.
        credit_score: The user's credit score.
        dti_ratio: The user's debt-to-income ratio.

      Returns:
        True if the loan is approved, False otherwise.
      """
      if income >= min_income and credit_score >= min_credit_score and dti_ratio <= max_dti_ratio:
        return True
      else:
        return False
```



```python
    # Display result
    is_approved = check_loan_approval(income, credit_score, dti_ratio)

    if is_approved:
        print(f"Congratulations, {name}! Your loan is approved.")
    else:
        print(f"Sorry, {name}. Your loan is denied.")

except ValueError:
    print("Invalid input. Please enter numerical values for income, credit score, and debt-to-income ratio.")
```

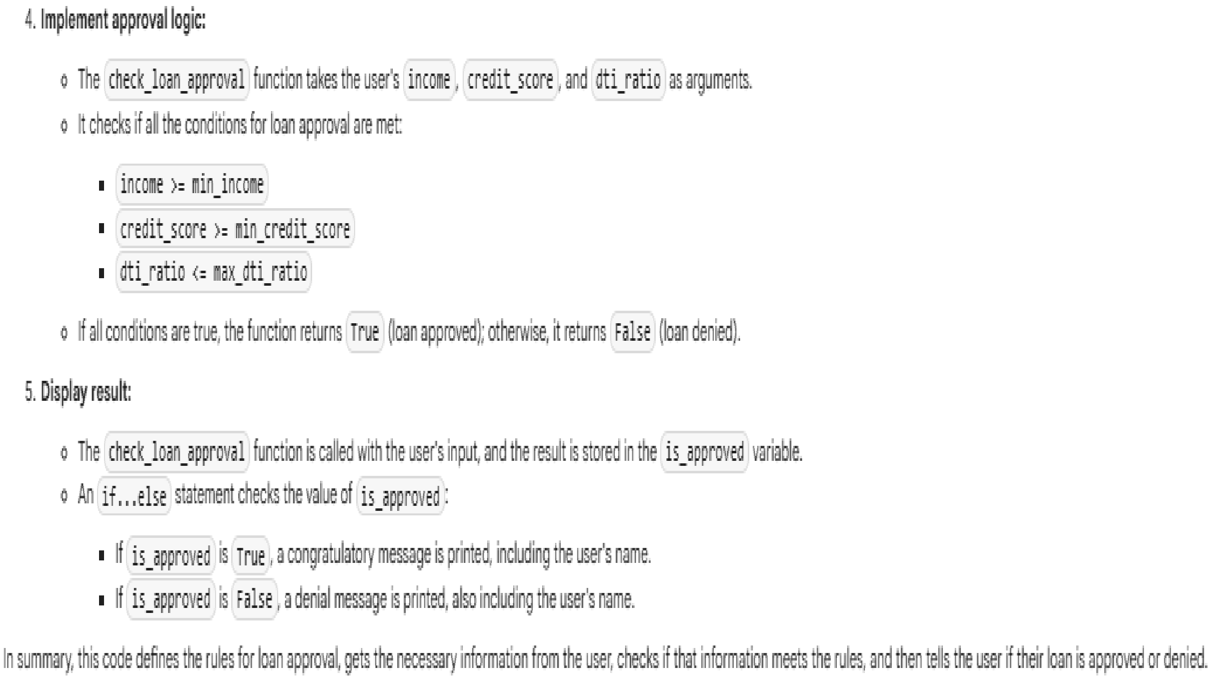
```
Enter your name: savika
Enter your annual income: 1000000
Enter your credit score: 650
Enter your debt-to-income ratio (as a decimal, e.g., 0.4): 0.4
Congratulations, savika! Your loan is approved.
```

Explanation:



This code implements a simple loan approval system in Python. Here's a breakdown:

1. **Define loan criteria:**
    - `min_income = 30000` : Sets the minimum required annual income to $30,000.
    - `min_credit_score = 650` : Sets the minimum required credit score to 650.
    - `max_dti_ratio = 0.4` : Sets the maximum allowed debt-to-income ratio to 0.4 (or 40%).
2. **Get user input:**
    - The program prompts the user to enter their name, annual income, credit score, and debt-to-income ratio using the `input()` function.
    - The inputs for income, credit score, and debt-to-income ratio are initially stored as strings.
3. **Error Handling:**
    - A `try...except ValueError` block is used to handle potential errors if the user enters non-numerical values for income, credit score, or debt-to-income ratio.
    - Inside the `try` block, the string inputs are converted to the appropriate data types: `income` to a `float` , `credit_score` to an `int` , and `dti_ratio` to a `float` .
    - If a `ValueError` occurs during the conversion (e.g., the user enters text instead of numbers), the `except` block is executed, and an error message is printed.



4. **Implement approval logic:**
    - The `check_loan_approval` function takes the user's `income` , `credit_score` , and `dti_ratio` as arguments.
    - It checks if all the conditions for loan approval are met:
        - `income >= min_income`
        - `credit_score >= min_credit_score`
        - `dti_ratio <= max_dti_ratio`
    - If all conditions are true, the function returns `True` (loan approved); otherwise, it returns `False` (loan denied).
5. **Display result:**
    - The `check_loan_approval` function is called with the user's input, and the result is stored in the `is_approved` variable.
    - An `if...else` statement checks the value of `is_approved` :
        - If `is_approved` is `True` , a congratulatory message is printed, including the user's name.
        - If `is_approved` is `False` , a denial message is printed, also including the user's name.

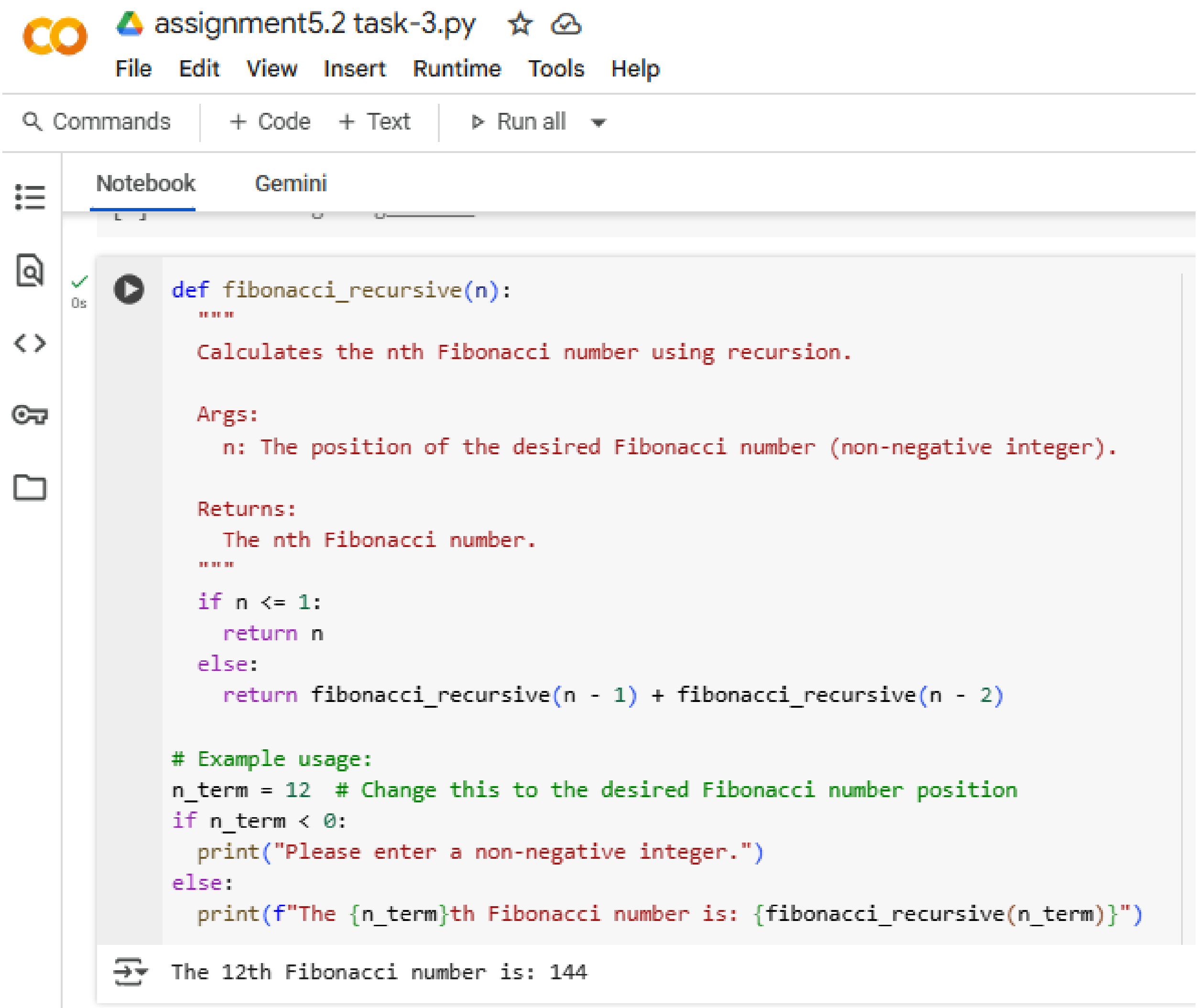
In summary, this code defines the rules for loan approval, gets the necessary information from the user, checks if that information meets the rules, and then tells the user if their loan is approved or denied.

TASK-3:

Prompt:

#write a python program to calculate the nth Fibonacci number using recursion and using functions.

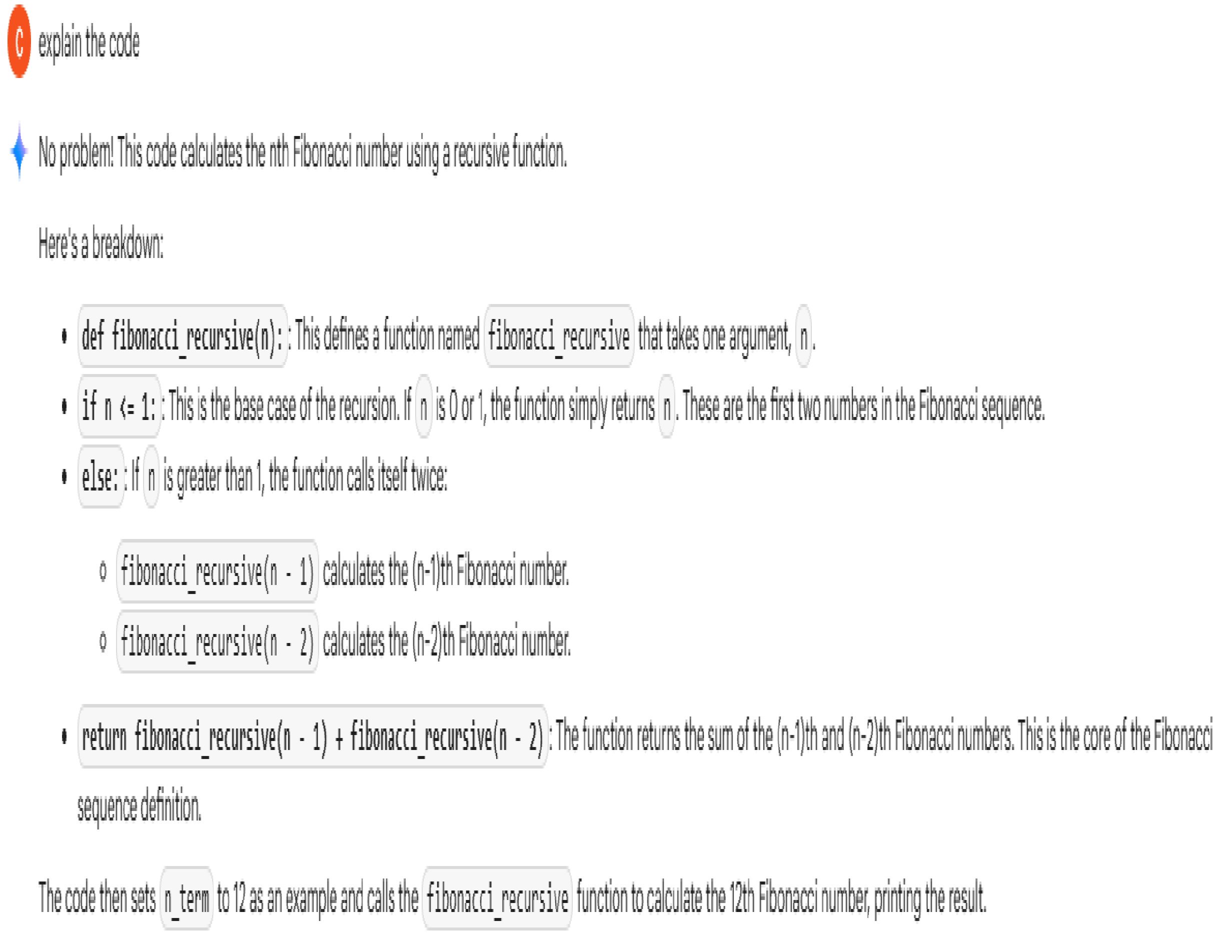
Code and Output:



```python
def fibonacci_recursive(n):
  """
  Calculates the nth Fibonacci number using recursion.

  Args:
    n: The position of the desired Fibonacci number (non-negative integer).

  Returns:
    The nth Fibonacci number.
  """
  if n <= 1:
    return n
  else:
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

# Example usage:
n_term = 12  # Change this to the desired Fibonacci number position
if n_term < 0:
  print("Please enter a non-negative integer.")
else:
  print(f"The {n_term}th Fibonacci number is: {fibonacci_recursive(n_term)}")
```

```
The 12th Fibonacci number is: 144
```

Explanation:



explain the code

No problem! This code calculates the nth Fibonacci number using a recursive function.

Here's a breakdown:

- `def fibonacci_recursive(n):` : This defines a function named `fibonacci_recursive` that takes one argument, `n`.
- `if n <= 1:` : This is the base case of the recursion. If `n` is 0 or 1, the function simply returns `n`. These are the first two numbers in the Fibonacci sequence.
- `else:` : If `n` is greater than 1, the function calls itself twice:
    - `fibonacci_recursive(n - 1)` calculates the (n-1)th Fibonacci number.
    - `fibonacci_recursive(n - 2)` calculates the (n-2)th Fibonacci number.
- `return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)` : The function returns the sum of the (n-1)th and (n-2)th Fibonacci numbers. This is the core of the Fibonacci sequence definition.

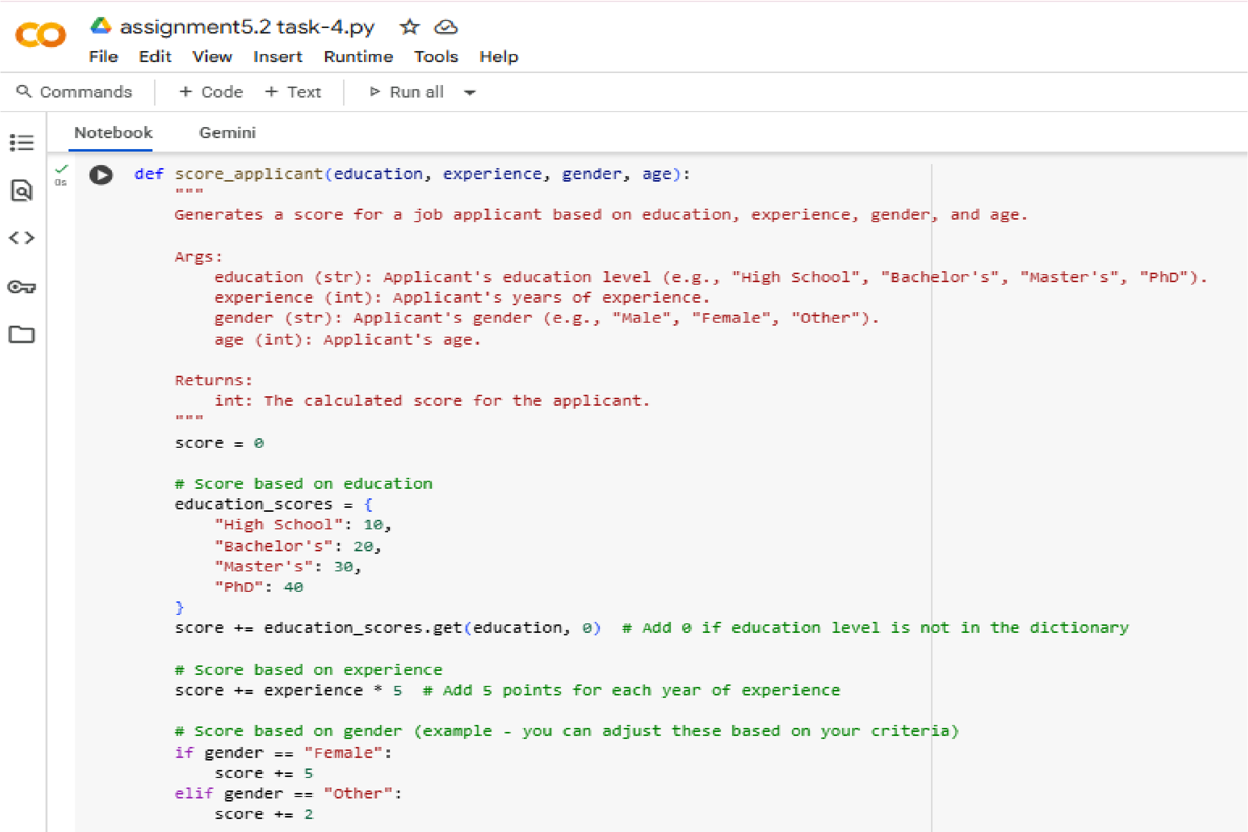
The code then sets `n_term` to 12 as an example and calls the `fibonacci_recursive` function to calculate the 12th Fibonacci number, printing the result.

TASK-4:

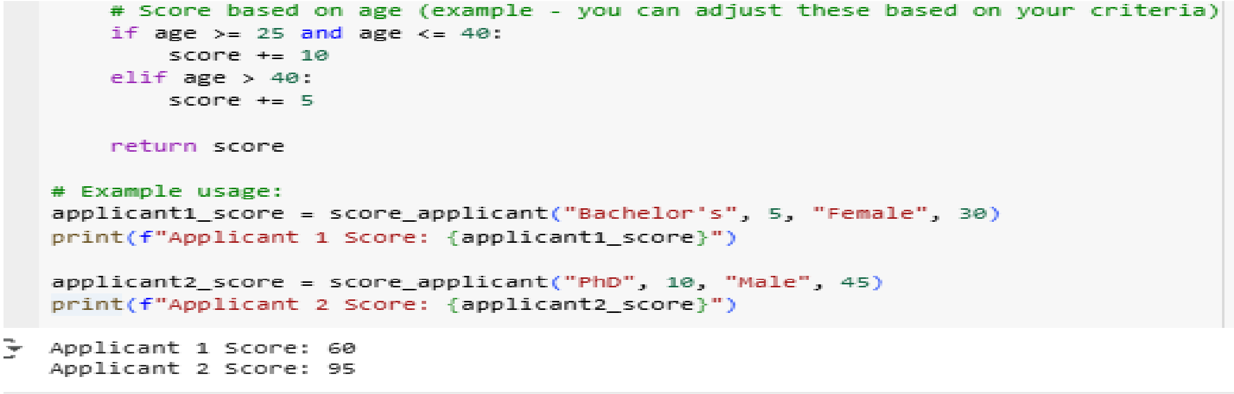
Prompt:

#write a python program to generate a job applicant scoring system based takes input education, experience, gender, age.

Code with Output:



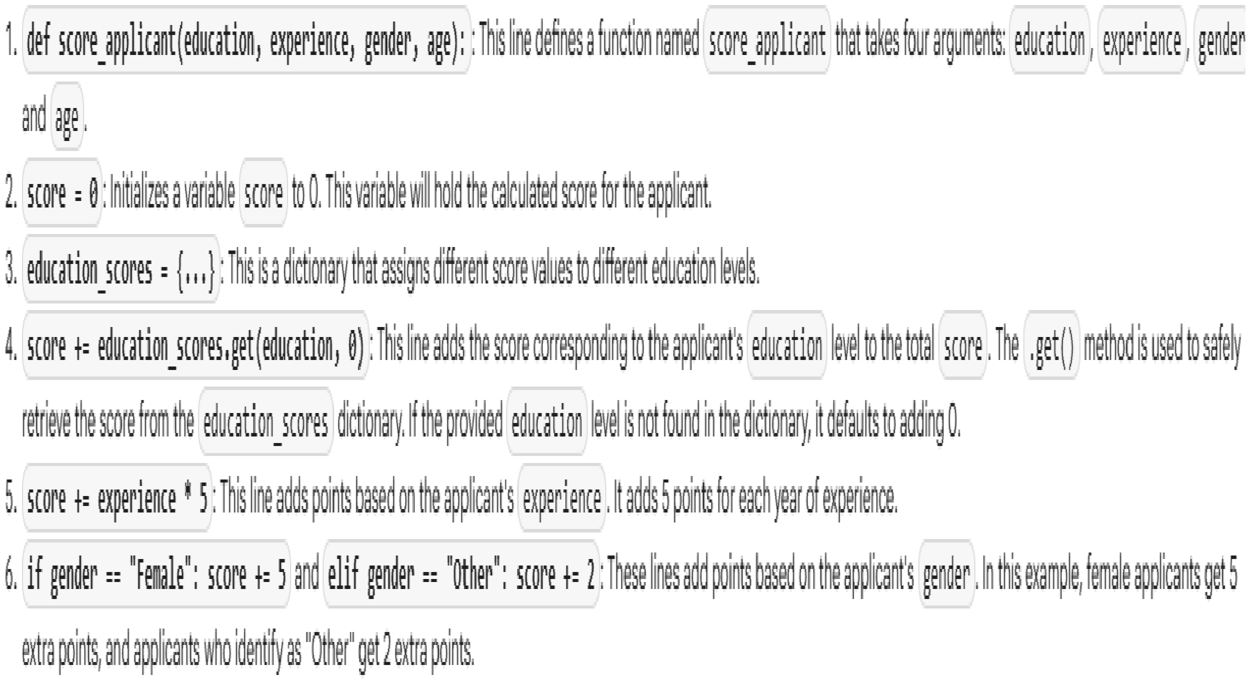
```
def score_applicant(education, experience, gender, age):
    """
    Generates a score for a job applicant based on education, experience, gender, and age.

    Args:
        education (str): Applicant's education level (e.g., "High School", "Bachelor's", "Master's", "PhD").
        experience (int): Applicant's years of experience.
        gender (str): Applicant's gender (e.g., "Male", "Female", "Other").
        age (int): Applicant's age.

    Returns:
        int: The calculated score for the applicant.
    """
    score = 0

    # Score based on education
    education_scores = {
        "High School": 10,
        "Bachelor's": 20,
        "Master's": 30,
        "PhD": 40
    }
    score += education_scores.get(education, 0)  # Add 0 if education level is not in the dictionary

    # Score based on experience
    score += experience * 5  # Add 5 points for each year of experience

    # Score based on gender (example - you can adjust these based on your criteria)
    if gender == "Female":
        score += 5
    elif gender == "Other":
        score += 2
```



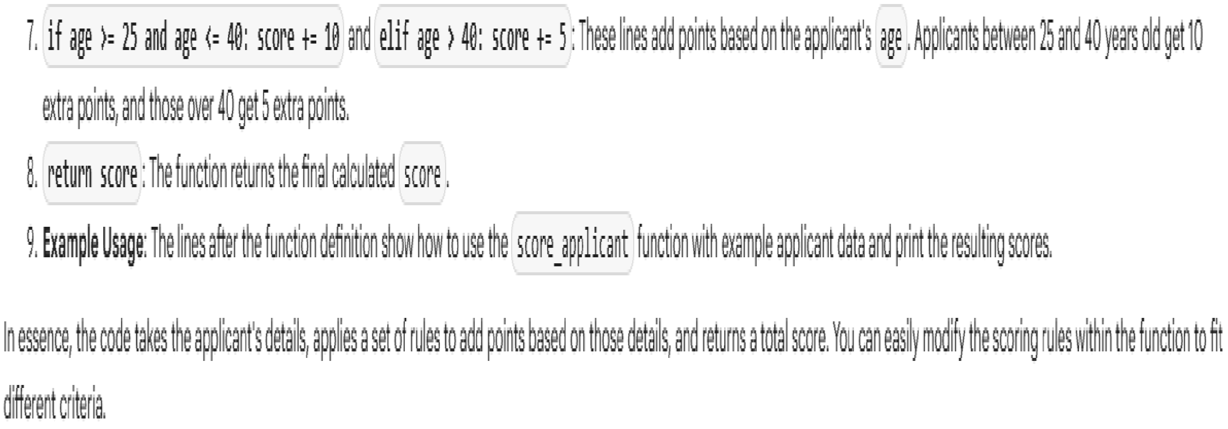
```
    # Score based on age (example - you can adjust these based on your criteria)
    if age >= 25 and age <= 40:
        score += 10
    elif age > 40:
        score += 5

    return score

# Example usage:
applicant1_score = score_applicant("Bachelor's", 5, "Female", 30)
print(f"Applicant 1 Score: {applicant1_score}")

applicant2_score = score_applicant("PhD", 10, "Male", 45)
print(f"Applicant 2 Score: {applicant2_score}")
```

```
Applicant 1 Score: 60
Applicant 2 Score: 95
```

Explanation:



1. `def score_applicant(education, experience, gender, age):` : This line defines a function named `score_applicant` that takes four arguments: `education` , `experience` , `gender` and `age` .
2. `score = 0` : Initializes a variable `score` to 0. This variable will hold the calculated score for the applicant.
3. `education_scores = {...}` : This is a dictionary that assigns different score values to different education levels.
4. `score += education_scores.get(education, 0)` : This line adds the score corresponding to the applicant's `education` level to the total `score` . The `.get()` method is used to safely retrieve the score from the `education_scores` dictionary. If the provided `education` level is not found in the dictionary, it defaults to adding 0.
5. `score += experience * 5` : This line adds points based on the applicant's `experience` . It adds 5 points for each year of experience.
6. `if gender == "Female": score += 5` and `elif gender == "Other": score += 2` : These lines add points based on the applicant's `gender` . In this example, female applicants get 5 extra points, and applicants who identify as "Other" get 2 extra points.



7. `if age >= 25 and age <= 40: score += 10` and `elif age > 40: score += 5` : These lines add points based on the applicant's `age` . Applicants between 25 and 40 years old get 10 extra points, and those over 40 get 5 extra points.
8. `return score` : The function returns the final calculated `score` .
9. **Example Usage:** The lines after the function definition show how to use the `score_applicant` function with example applicant data and print the resulting scores.

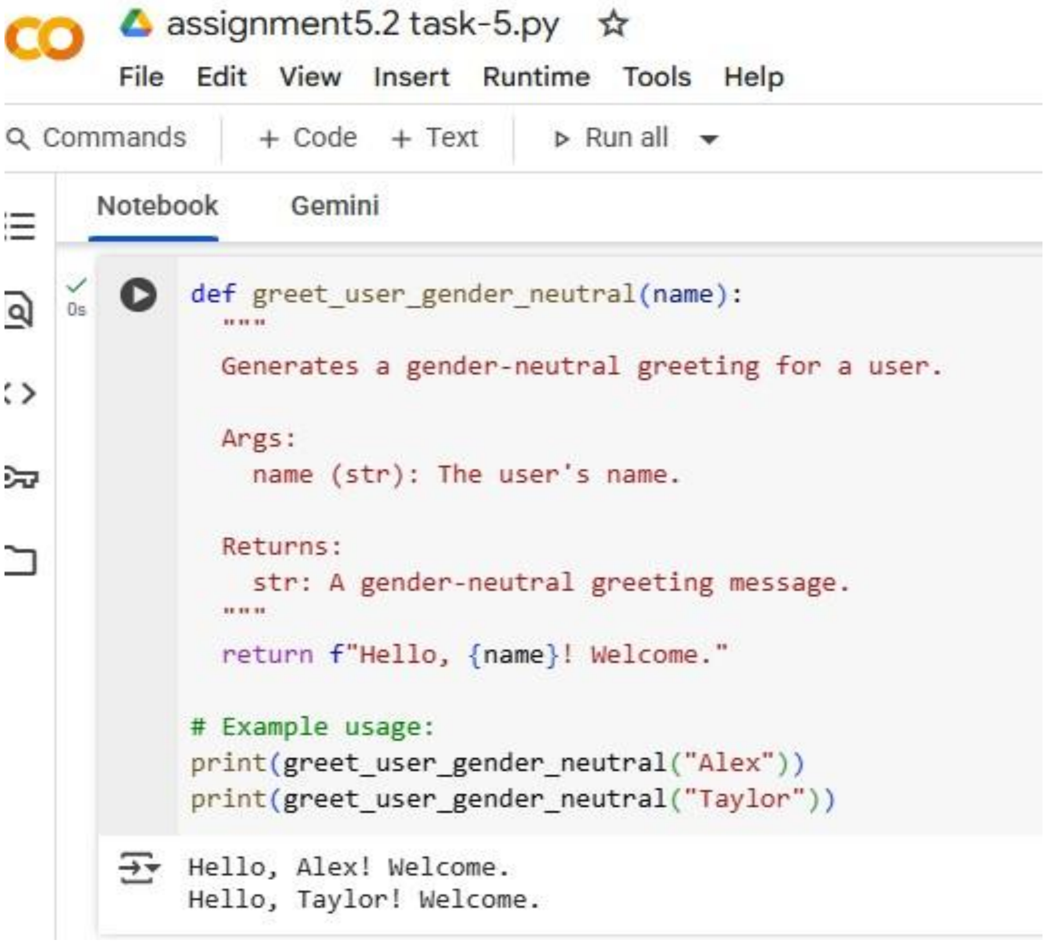
In essence, the code takes the applicant's details, applies a set of rules to add points based on those details, and returns a total score. You can easily modify the scoring rules within the function to fit different criteria.

TASK-5:

Prompt:

#write a python program to regenerate a code into gender-neutral.

Code with Output:

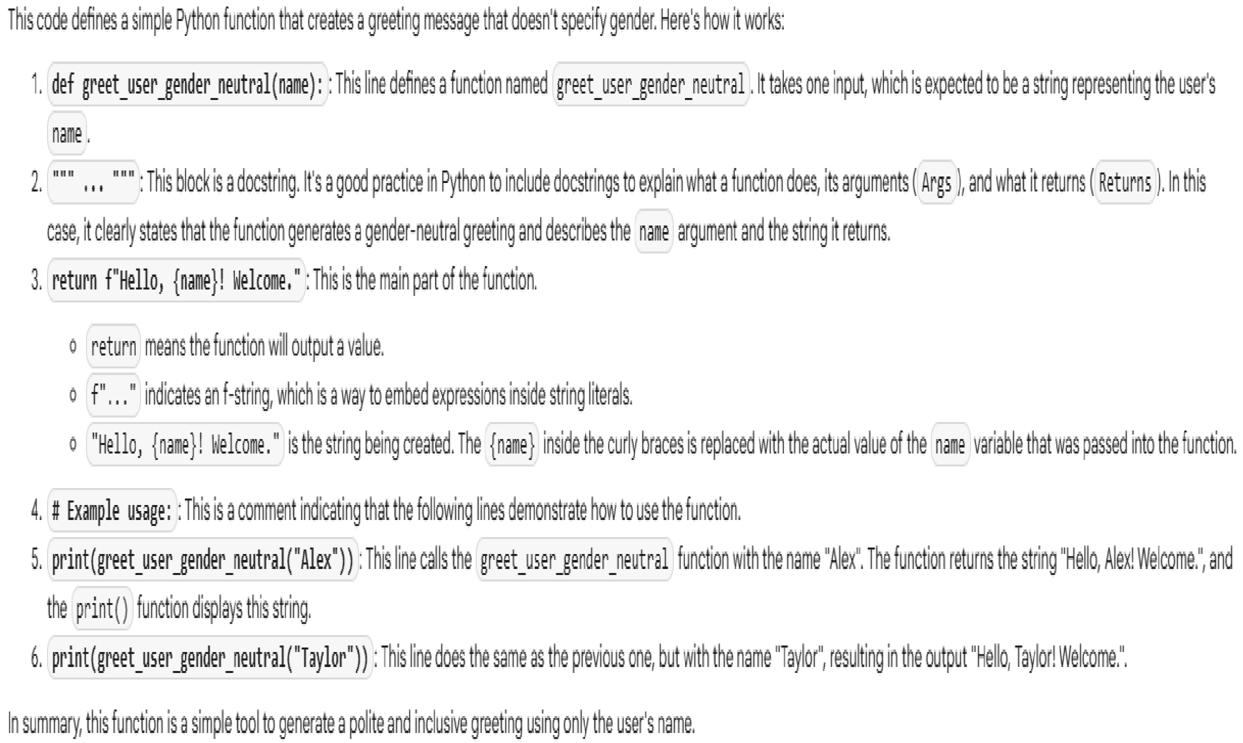


assignment5.2 task-5.py
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Notebook Gemini
0s

```
def greet_user_gender_neutral(name):
  """
  Generates a gender-neutral greeting for a user.

  Args:
    name (str): The user's name.

  Returns:
    str: A gender-neutral greeting message.
  """
  return f"Hello, {name}! Welcome."

# Example usage:
print(greet_user_gender_neutral("Alex"))
print(greet_user_gender_neutral("Taylor"))
```

```
Hello, Alex! Welcome.
Hello, Taylor! Welcome.
```

Explanation:



This code defines a simple Python function that creates a greeting message that doesn't specify gender. Here's how it works:

1. **`def greet_user_gender_neutral(name):`**: This line defines a function named `greet_user_gender_neutral`. It takes one input, which is expected to be a string representing the user's `name`.
2. **`""" ... """`**: This block is a docstring. It's a good practice in Python to include docstrings to explain what a function does, its arguments (`Args`), and what it returns (`Returns`). In this case, it clearly states that the function generates a gender-neutral greeting and describes the `name` argument and the string it returns.
3. **`return f"Hello, {name}! Welcome."`**: This is the main part of the function.
    - `return` means the function will output a value.
    - `f"..."` indicates an f-string, which is a way to embed expressions inside string literals.
    - `"Hello, {name}! Welcome."` is the string being created. The `{name}` inside the curly braces is replaced with the actual value of the `name` variable that was passed into the function.
4. **`# Example usage:`**: This is a comment indicating that the following lines demonstrate how to use the function.
5. **`print(greet_user_gender_neutral("Alex"))`**: This line calls the `greet_user_gender_neutral` function with the name "Alex". The function returns the string "Hello, Alex! Welcome.", and the `print()` function displays this string.
6. **`print(greet_user_gender_neutral("Taylor"))`**: This line does the same as the previous one, but with the name "Taylor", resulting in the output "Hello, Taylor! Welcome.".

In summary, this function is a simple tool to generate a polite and inclusive greeting using only the user's name.