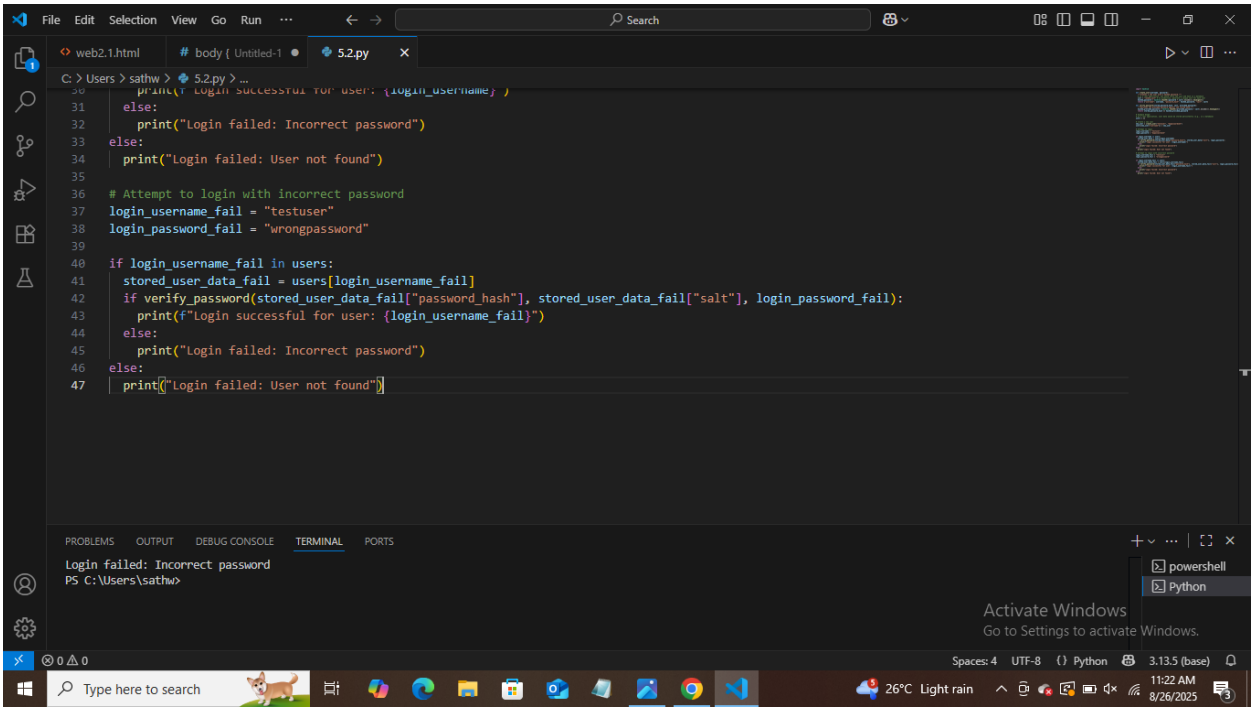
Lab 5.2

```python
import hashlib

def create_user(username, password):
  """Creates a new user with a hashed password."""
  # In a real application, you would store the salt and hash in a database
  salt = "random_salt" # In production, generate a unique salt for each user
  hashed_password = hashlib.sha256((password + salt).encode()).hexdigest()
  return {"username": username, "password_hash": hashed_password, "salt": salt}

def verify_password(stored_password_hash, salt, provided_password):
  """Verifies the provided password against the stored hash."""
  hashed_provided_password = hashlib.sha256((provided_password + salt).encode()).hexdigest()
  return stored_password_hash == hashed_provided_password

# Example Usage:
# In a real application, user data would be stored persistently (e.g., in a database)
users = {}

# Create a new user
new_user = create_user("testuser", "mypassword123")
users[new_user["username"]] = new_user

# Attempt to login
login_username = "testuser"
login_password = "mypassword123"
```

```
Login failed: Incorrect password
PS C:\Users\sathw>
```

```python
users = {}

# Create a new user
new_user = create_user("testuser", "mypassword123")
users[new_user["username"]] = new_user

# Attempt to login
login_username = "testuser"
login_password = "mypassword123"

if login_username in users:
  stored_user_data = users[login_username]
  if verify_password(stored_user_data["password_hash"], stored_user_data["salt"], login_password):
    print(f"Login successful for user: {login_username}")
  else:
    print("Login failed: Incorrect password")
else:
  print("Login failed: User not found")

# Attempt to login with incorrect password
login_username_fail = "testuser"
login_password_fail = "wrongpassword"

if login_username_fail in users:
  stored_user_data_fail = users[login_username_fail]
```

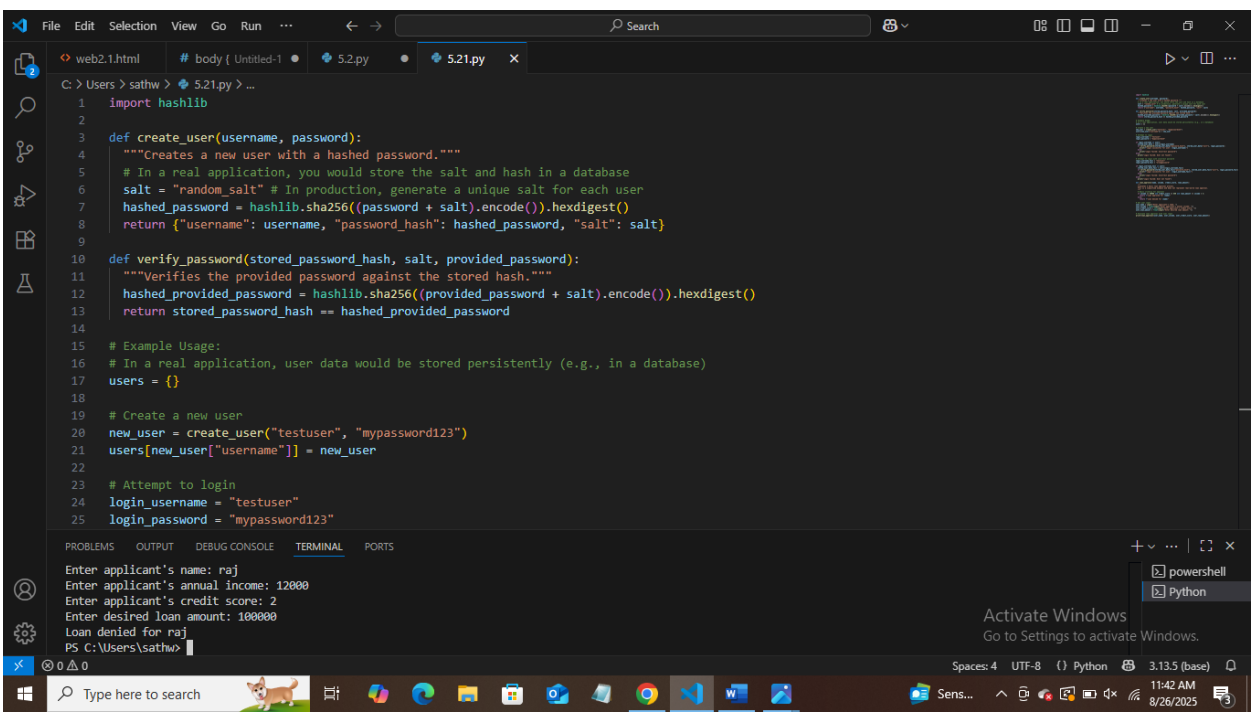
```
Login failed: Incorrect password
PS C:\Users\sathw>
```

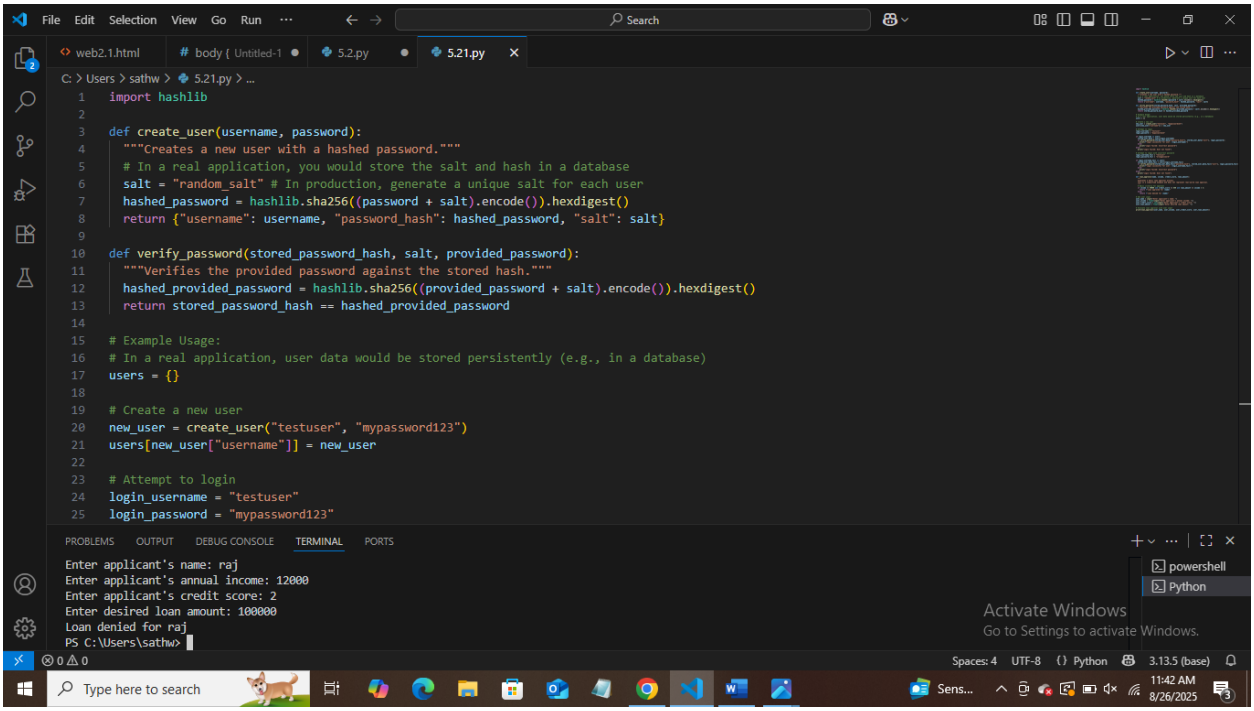


Explanation:

As this ai tool gives the code by which it gives to enter login if it successful it gives successful or it give3s login failure.

LAB 5.22





EXPLANATION:

1. **User Authentication:** It includes functions to securely create users with hashed passwords and verify provided passwords against the stored hash and salt.
2. **Loan Approval Simulation:** It contains a simple function to simulate a loan approval process based on basic income, credit score, and loan amount criteria, taking user input for evaluation.

LAB 5.23

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

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

  Returns:
    The nth Fibonacci number.
  """
  # Base case: The first two Fibonacci numbers are 0 and 1.
  if n <= 1:
    return n
  # Recursive step: The nth Fibonacci number is the sum of the (n-1)th and (n-2)th Fibonacci numbers.
  else:
    return recursive_fibonacci(n-1) + recursive_fibonacci(n-2)

# Example usage:
# Get input from the user
num = int(input("Enter a non-negative integer to find its Fibonacci number: "))

# Check if the input is valid
if num < 0:
  print("Please enter a non-negative integer.")
else:
```

```
Enter desired loan amount: 100000
Loan denied for raj
PS C:\Users\sathw> & C:/Users/sathw/anaconda3/python.exe c:/Users/sathw/lab5.23.py
Enter a non-negative integer to find its Fibonacci number: 7
The 7th Fibonacci number is: 13
PS C:\Users\sathw>
```

```python
def loan_approval(name, income, credit_score, loan_amount):
  # Basic criteria for approval
  if income >= 50000 and credit_score >= 650 and loan_amount <= income * 3:
    return f"Loan approved for {name}"
  else:
    return f"Loan denied for {name}"

# Get user input
user_name = input("Enter applicant's name: ")
user_income = float(input("Enter applicant's annual income: "))
user_credit_score = int(input("Enter applicant's credit score: "))
user_loan_amount = float(input("Enter desired loan amount: "))

# Evaluate loan approval with user input
print(loan_approval(user_name, user_income, user_credit_score, user_loan_amount))
```

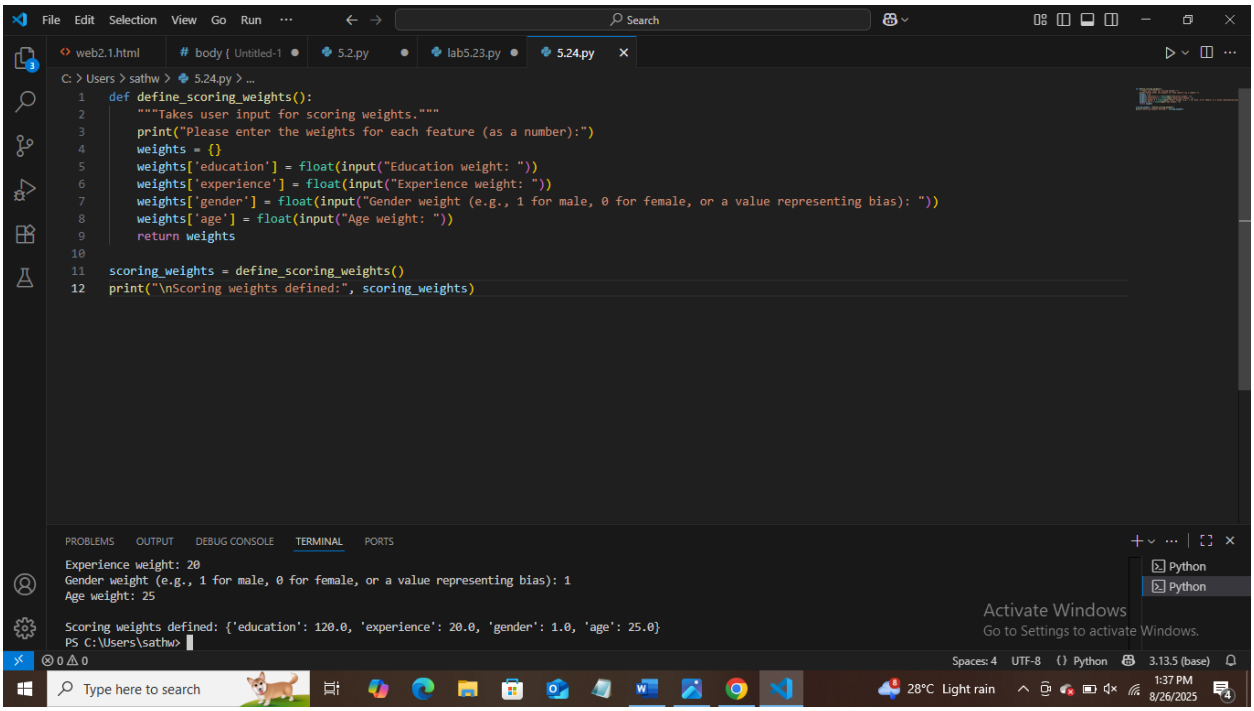
```
Enter applicant's name: raj
Enter applicant's annual income: 12000
Enter applicant's credit score: 2
Enter desired loan amount: 100000
Loan denied for raj
PS C:\Users\sathw>
```

EXPLANATION:

- Defines a recursive function recursive_fibonacci(n) with a base case for n <= 1 and a recursive step where the nth Fibonacci number is the sum of the (n-1)th and (n-2)th numbers.

- Prompts the user to enter a non-negative integer.
- Validates the input and calculates and prints the corresponding Fibonacci number using the recursive_fibonacci function.

LAB 5.24

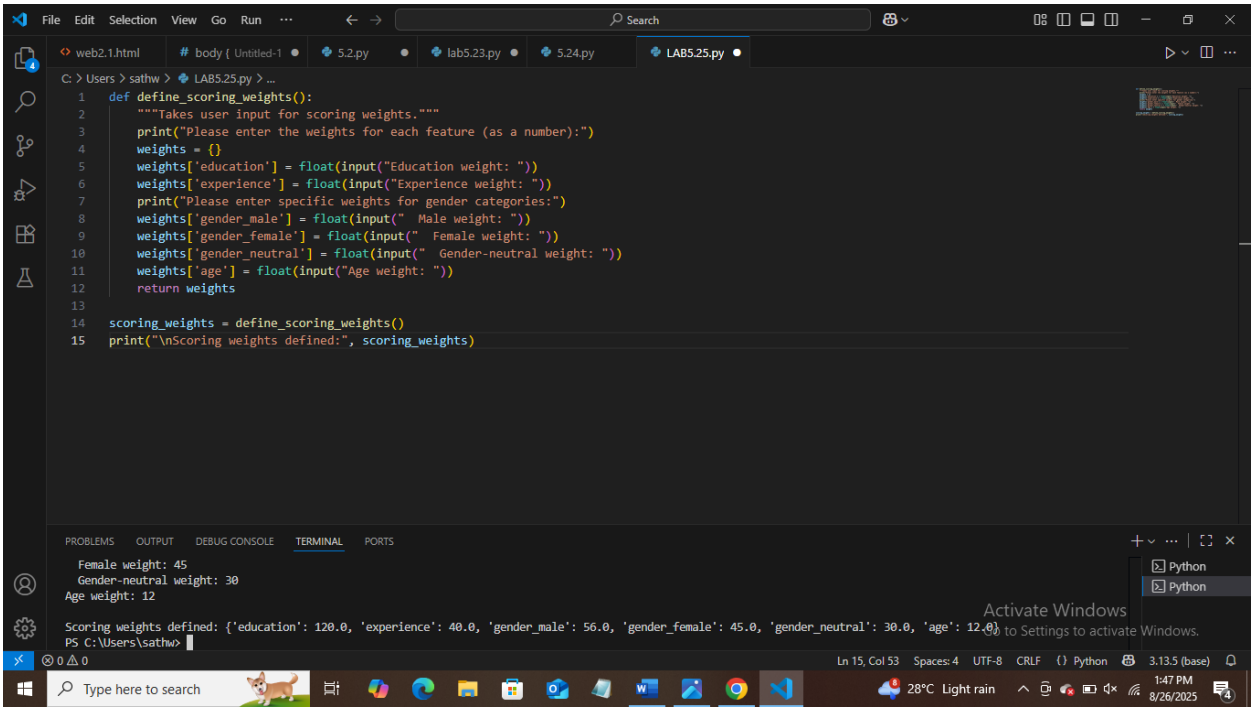


EXPLANATION:

- **def define_scoring_weights():**: This line defines a function named define_scoring_weights.
- **print("Please enter the weights for each feature (as a number):")**: This line prints a message to the console asking the user to input the weights.
- **weights = {}**: This line initializes an empty dictionary called weights which will store the feature weights.
- **weights['education'] = float(input("Education weight: "))**: This line prompts the user to enter the weight for education using the input() function. The entered value is converted to a floating-point number using float() and stored in the weights dictionary with the key 'education'. The same process is repeated for 'experience', 'gender', and 'age'.
- **return weights**: This line returns the weights dictionary containing the user-defined weights.

- **scoring_weights = define_scoring_weights()**: This line calls the define_scoring_weights() function and stores the returned dictionary of weights in the variable scoring_weights.
- **print("\nScoring weights defined:", scoring_weights)**: This line prints the scoring_weights dictionary to the console, showing the user the weights they entered.

LAB 5.25



EXPLANATION:

THIS CODE IS AS SAME AS THE PREVIOUS ONE BUT WHICH AS DIFFERENT OUT PUT.