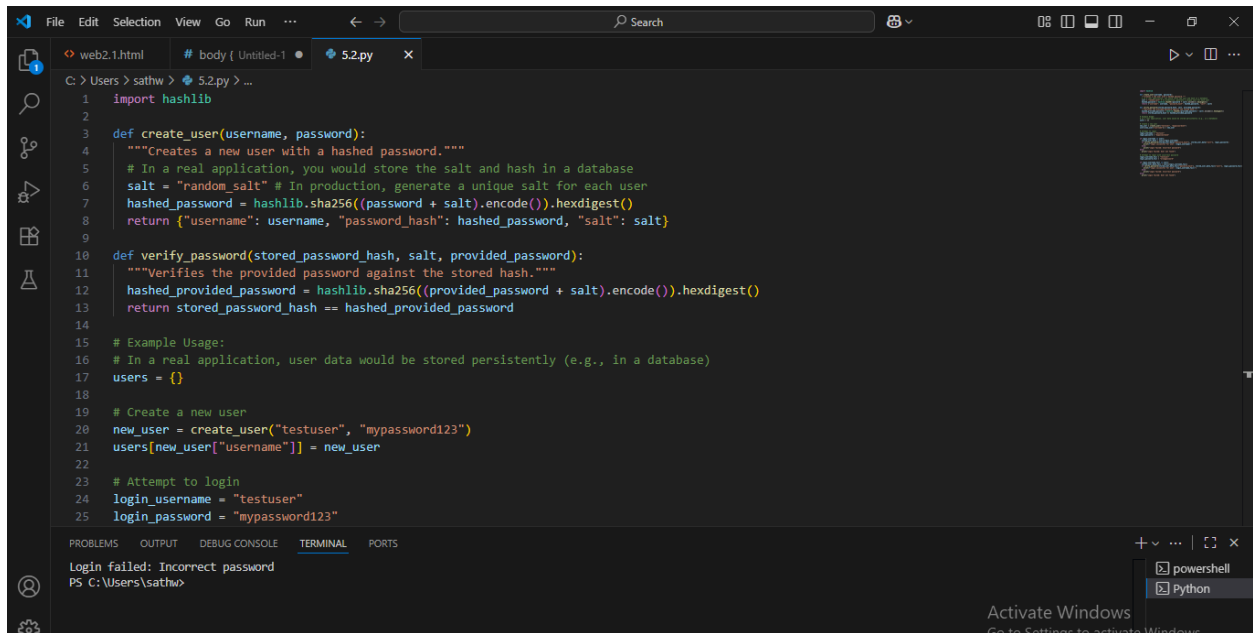
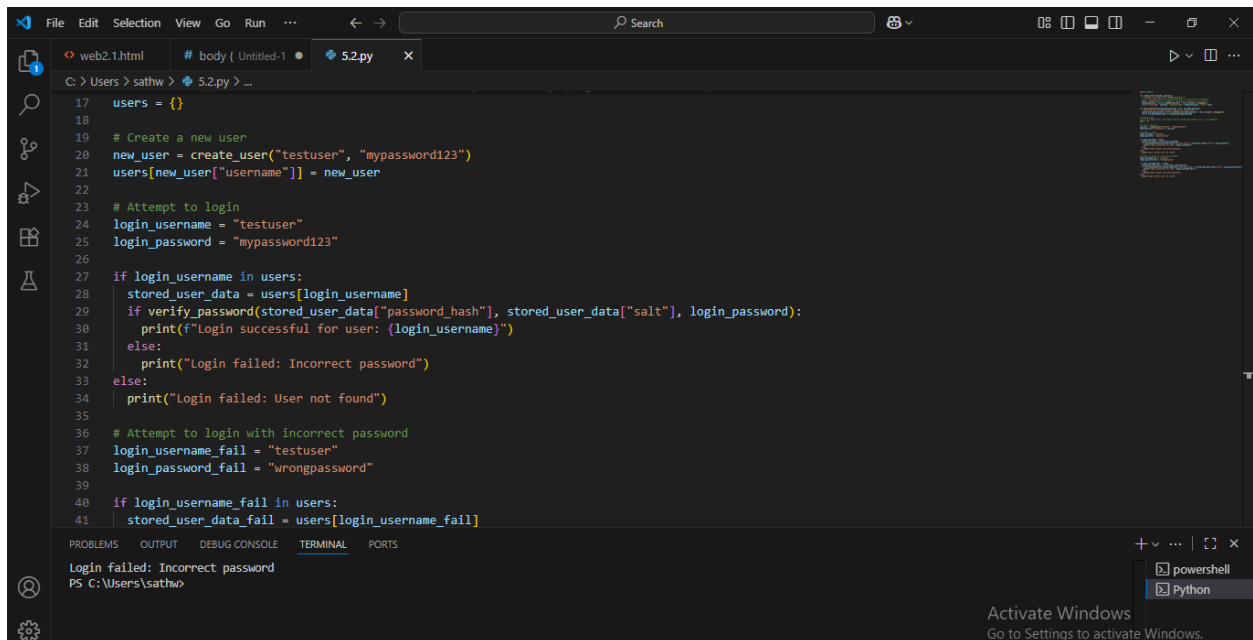


Lab 5.2



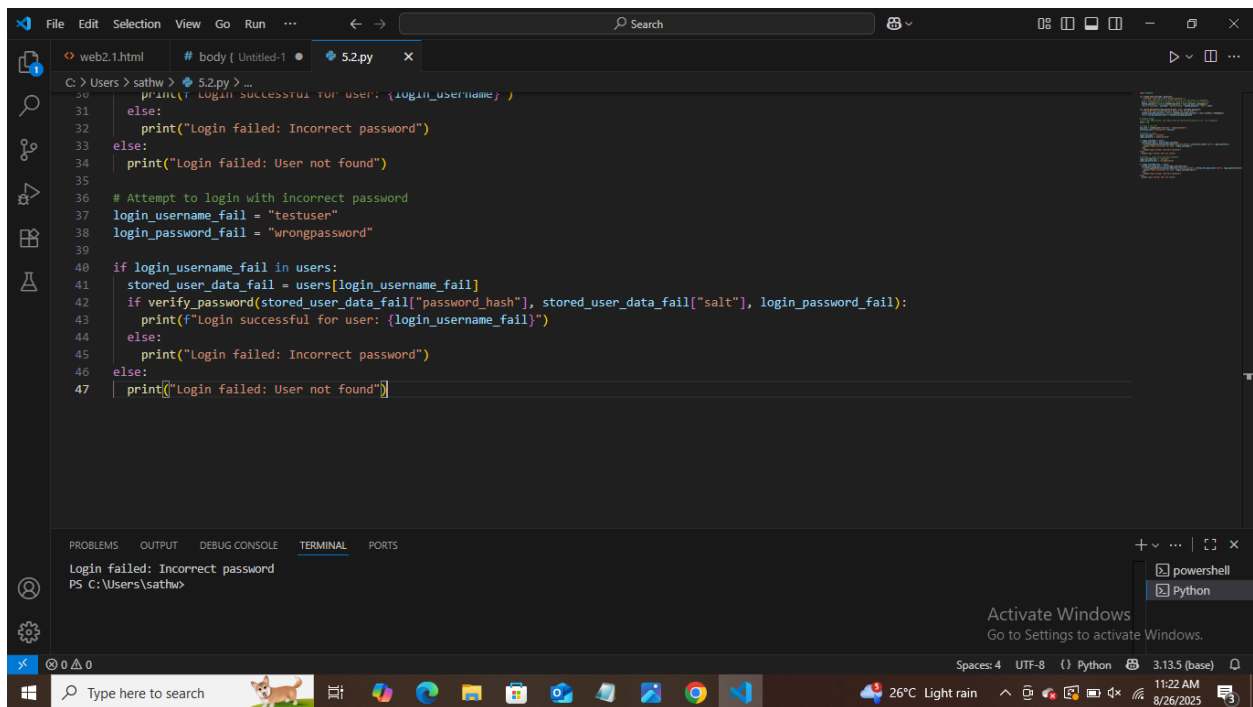
The screenshot shows a Visual Studio Code editor window with a file named `5.2.py` open. The code defines two functions: `create_user` and `verify_password`. The `create_user` function takes a username and password, generates a unique salt, hashes the password with the salt using SHA-256, and returns a dictionary containing the username, password hash, and salt. The `verify_password` function takes a stored password hash, a salt, and a provided password, hashes the provided password with the salt, and compares it to the stored hash. Below the functions, there is an example usage section that creates a new user 'testuser' with password 'mypassword123' and attempts to login with the same credentials. The terminal output shows 'Login failed: Incorrect password'.

```
File Edit Selection View Go Run ... Search
web2.1.html # body (Untitled-1) 5.2.py x
C:\Users\sathw> 5.2.py > ...
1 import hashlib
2
3
4 def create_user(username, password):
5     """Creates a new user with a hashed password."""
6     # In a real application, you would store the salt and hash in a database
7     salt = "random_salt" # In production, generate a unique salt for each user
8     hashed_password = hashlib.sha256((password + salt).encode()).hexdigest()
9     return {"username": username, "password_hash": hashed_password, "salt": salt}
10
11 def verify_password(stored_password_hash, salt, provided_password):
12     """Verifies the provided password against the stored hash."""
13     hashed_provided_password = hashlib.sha256((provided_password + salt).encode()).hexdigest()
14     return stored_password_hash == hashed_provided_password
15
16 # Example Usage:
17 # In a real application, user data would be stored persistently (e.g., in a database)
18 users = {}
19
20 # Create a new user
21 new_user = create_user("testuser", "mypassword123")
22 users[new_user["username"]] = new_user
23
24 # Attempt to login
25 login_username = "testuser"
26 login_password = "mypassword123"
27
28 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Login failed: Incorrect password
PS C:\Users\sathw>
powershell
Python
Activate Windows
Go to Settings to activate Windows.
```



The screenshot shows the same Visual Studio Code editor window with the `5.2.py` file. The code now includes a login logic section that checks if the login username is in the `users` dictionary. If it is, it calls `verify_password` to check the password. If the password is correct, it prints a success message. If the password is incorrect, it prints 'Login failed: Incorrect password'. If the username is not found, it prints 'Login failed: User not found'. Below this, there is an example usage section that attempts to login with the correct credentials 'testuser' and 'mypassword123', and then with incorrect credentials 'testuser' and 'wrongpassword'. The terminal output shows 'Login failed: Incorrect password'.

```
File Edit Selection View Go Run ... Search
web2.1.html # body (Untitled-1) 5.2.py x
C:\Users\sathw> 5.2.py > ...
17 users = {}
18
19 # Create a new user
20 new_user = create_user("testuser", "mypassword123")
21 users[new_user["username"]] = new_user
22
23 # Attempt to login
24 login_username = "testuser"
25 login_password = "mypassword123"
26
27 if login_username in users:
28     stored_user_data = users[login_username]
29     if verify_password(stored_user_data["password_hash"], stored_user_data["salt"], login_password):
30         print(f"Login successful for user: {login_username}")
31     else:
32         print("Login failed: Incorrect password")
33 else:
34     print("Login failed: User not found")
35
36 # Attempt to login with incorrect password
37 login_username_fail = "testuser"
38 login_password_fail = "wrongpassword"
39
40 if login_username_fail in users:
41     stored_user_data_fail = users[login_username_fail]
42
43 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Login failed: Incorrect password
PS C:\Users\sathw>
powershell
Python
Activate Windows
Go to Settings to activate Windows.
```



```
File Edit Selection View Go Run ... Search
web2.1.html # body (Untitled-1) 5.2.py x
C:\Users\sathw > 5.2.py > ...
30 print("Login successful for user: {login_username}")
31 else:
32     print("Login failed: Incorrect password")
33 else:
34     print("Login failed: User not found")
35
36 # Attempt to login with incorrect password
37 login_username_fail = "testuser"
38 login_password_fail = "wrongpassword"
39
40 if login_username_fail in users:
41     stored_user_data_fail = users[login_username_fail]
42     if verify_password(stored_user_data_fail["password_hash"], stored_user_data_fail["salt"], login_password_fail):
43         print(f"Login successful for user: {login_username_fail}")
44     else:
45         print("Login failed: Incorrect password")
46 else:
47     print("Login failed: User not found")

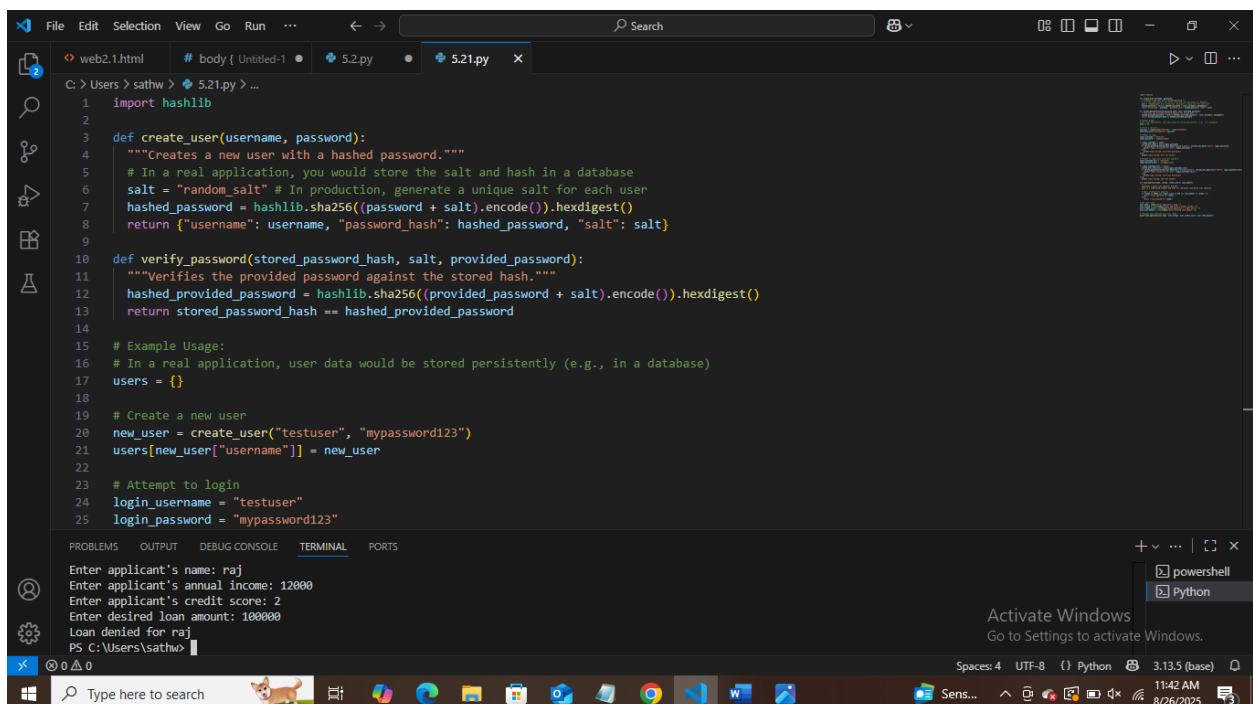
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Login failed: Incorrect password
PS C:\Users\sathw>

Activate Windows
Go to Settings to activate Windows.
Spaces: 4 UTF-8 Python 3.13.5 (base)
Type here to search 26°C Light rain 11:22 AM 8/26/2025
```

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



```
File Edit Selection View Go Run ... Search
web2.1.html # body (Untitled-1) 5.2.py 5.21.py x
C:\Users\sathw > 5.21.py > ...
1 import hashlib
2
3 def create_user(username, password):
4     """Creates a new user with a hashed password."""
5     # In a real application, you would store the salt and hash in a database
6     salt = "random.salt" # In production, generate a unique salt for each user
7     hashed_password = hashlib.sha256((password + salt).encode()).hexdigest()
8     return {"username": username, "password_hash": hashed_password, "salt": salt}
9
10 def verify_password(stored_password_hash, salt, provided_password):
11     """Verifies the provided password against the stored hash."""
12     hashed_provided_password = hashlib.sha256((provided_password + salt).encode()).hexdigest()
13     return stored_password_hash == hashed_provided_password
14
15 # Example Usage:
16 # In a real application, user data would be stored persistently (e.g., in a database)
17 users = {}
18
19 # Create a new user
20 new_user = create_user("testuser", "mypassword123")
21 users[new_user["username"]] = new_user
22
23 # Attempt to login
24 login_username = "testuser"
25 login_password = "mypassword123"

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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>

Activate Windows
Go to Settings to activate Windows.
Spaces: 4 UTF-8 Python 3.13.5 (base)
Type here to search Sens... 11:42 AM 8/26/2025
```

The screenshot shows a Visual Studio Code editor window with a Python file named `5.21.py`. The script implements two main functions: `create_user` and `verify_password`, both using the `hashlib` module for hashing. The `create_user` function generates a random salt and a hashed password. The `verify_password` function compares a provided password with a stored hash and salt. Below these functions, there is an example usage section that creates a new user and attempts a login. At the bottom of the editor, the TERMINAL panel is open, showing the execution of the script. The terminal output displays prompts for user name, income, credit score, and loan amount, followed by a successful login message and a loan denial message. The Windows taskbar is visible at the bottom, showing the time as 11:42 AM on 8/26/2025.

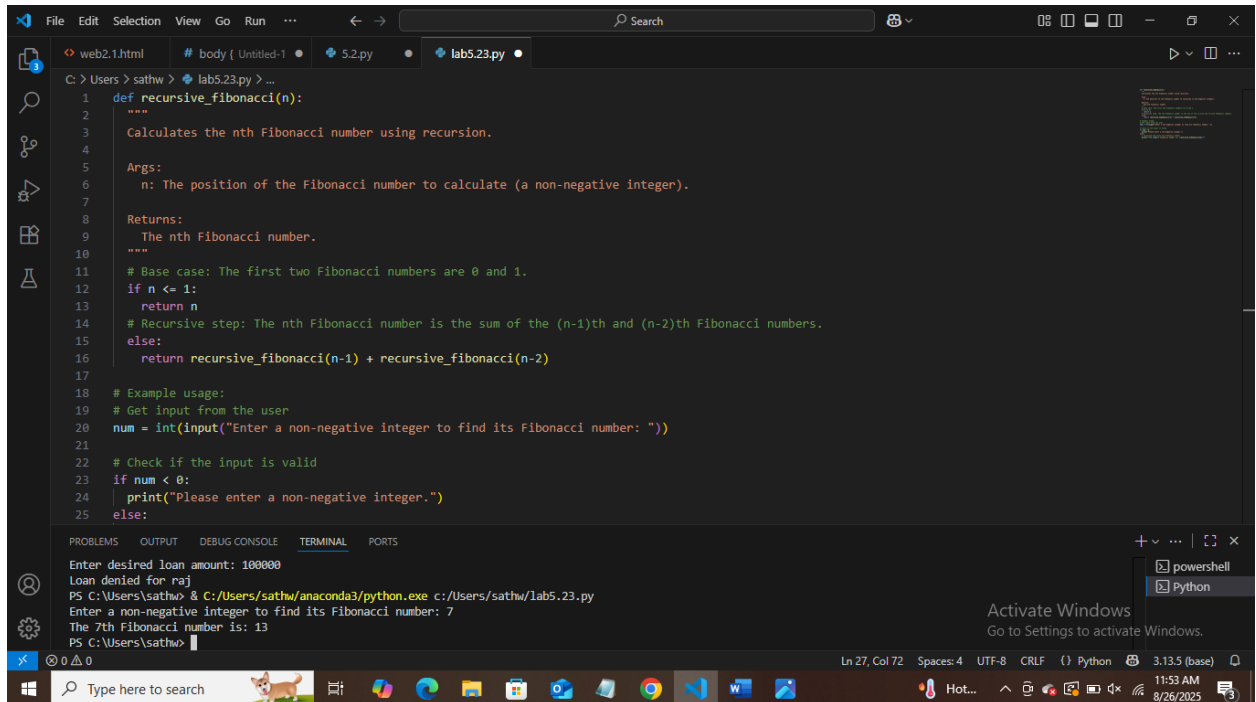
```
1 import hashlib
2
3 def create_user(username, password):
4     """Creates a new user with a hashed password."""
5     # In a real application, you would store the salt and hash in a database
6     salt = "random_salt" # In production, generate a unique salt for each user
7     hashed_password = hashlib.sha256((password + salt).encode()).hexdigest()
8     return {"username": username, "password_hash": hashed_password, "salt": salt}
9
10 def verify_password(stored_password_hash, salt, provided_password):
11     """Verifies the provided password against the stored hash."""
12     hashed_provided_password = hashlib.sha256((provided_password + salt).encode()).hexdigest()
13     return stored_password_hash == hashed_provided_password
14
15 # Example Usage:
16 # In a real application, user data would be stored persistently (e.g., in a database)
17 users = {}
18
19 # Create a new user
20 new_user = create_user("testuser", "mypassword123")
21 users[new_user["username"]] = new_user
22
23 # Attempt to login
24 login_username = "testuser"
25 login_password = "mypassword123"
```

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:

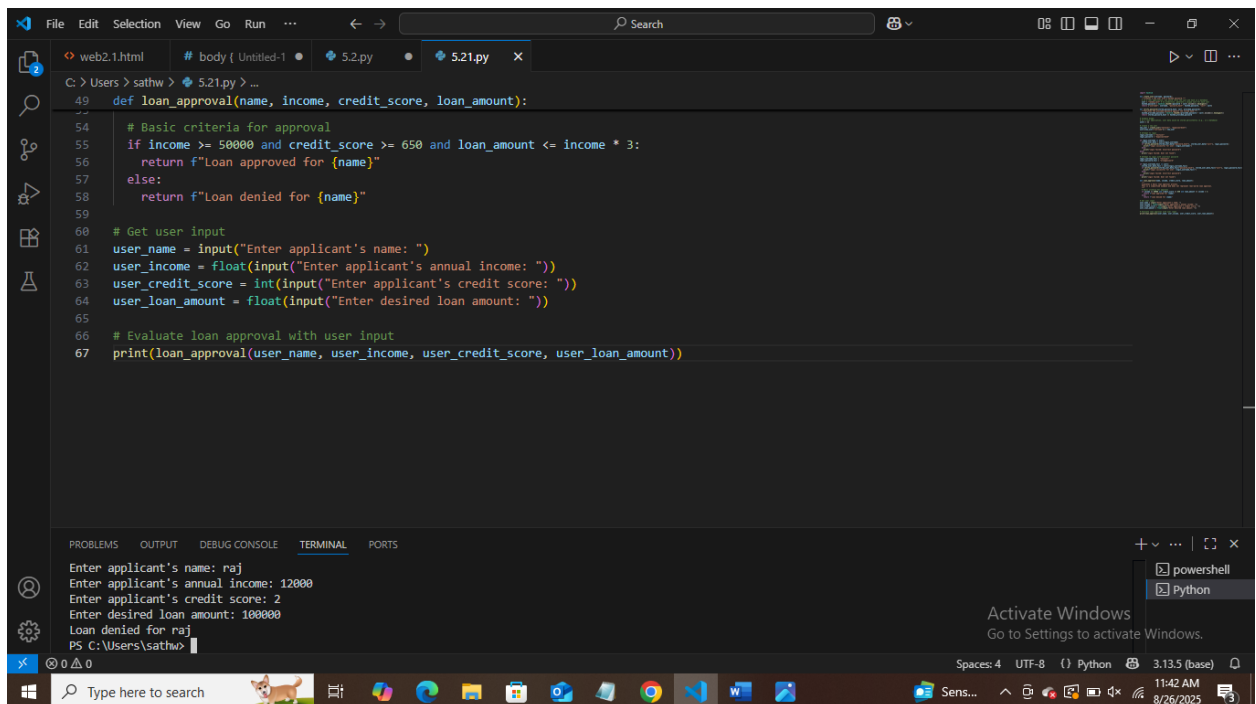
1. **User Authentication:** It includes functions to securely create users with hashed passwords and verify provided passwords against the stored hash and salt.
2. **Loan Approval Simulation:** It contains a simple function to simulate a loan approval process based on basic income, credit score, and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

LAB 5.23



```
File Edit Selection View Go Run ... Search
C:\Users\sathw> lab5.23.py ...
1 def recursive_fibonacci(n):
2     """
3     Calculates the nth Fibonacci number using recursion.
4
5     Args:
6         n: The position of the Fibonacci number to calculate (a non-negative integer).
7
8     Returns:
9         The nth Fibonacci number.
10    """
11    # Base case: The first two Fibonacci numbers are 0 and 1.
12    if n <= 1:
13        return n
14    # Recursive step: The nth Fibonacci number is the sum of the (n-1)th and (n-2)th Fibonacci numbers.
15    else:
16        return recursive_fibonacci(n-1) + recursive_fibonacci(n-2)
17
18 # Example usage:
19 # Get input from the user
20 num = int(input("Enter a non-negative integer to find its Fibonacci number: "))
21
22 # Check if the input is valid
23 if num < 0:
24     print("Please enter a non-negative integer.")
25 else:
26     print("The 7th Fibonacci number is: 13")
PS C:\Users\sathw>
```

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>



```
File Edit Selection View Go Run ... Search
C:\Users\sathw> 5.21.py ...
49 def loan_approval(name, income, credit_score, loan_amount):
50     """
51     Basic criteria for approval
52     if income >= 50000 and credit_score >= 650 and loan_amount <= income * 3:
53         return f"Loan approved for {name}"
54     else:
55         return f"Loan denied for {name}"
56
57 # Get user input
58 user_name = input("Enter applicant's name: ")
59 user_income = float(input("Enter applicant's annual income: "))
60 user_credit_score = int(input("Enter applicant's credit score: "))
61 user_loan_amount = float(input("Enter desired loan amount: "))
62
63 # Evaluate loan approval with user input
64 print(loan_approval(user_name, user_income, user_credit_score, user_loan_amount))
65
66 Enter applicant's name: raj
67 Enter applicant's annual income: 12000
68 Enter applicant's credit score: 2
69 Enter desired loan amount: 100000
70 Loan denied for raj
PS C:\Users\sathw>
```

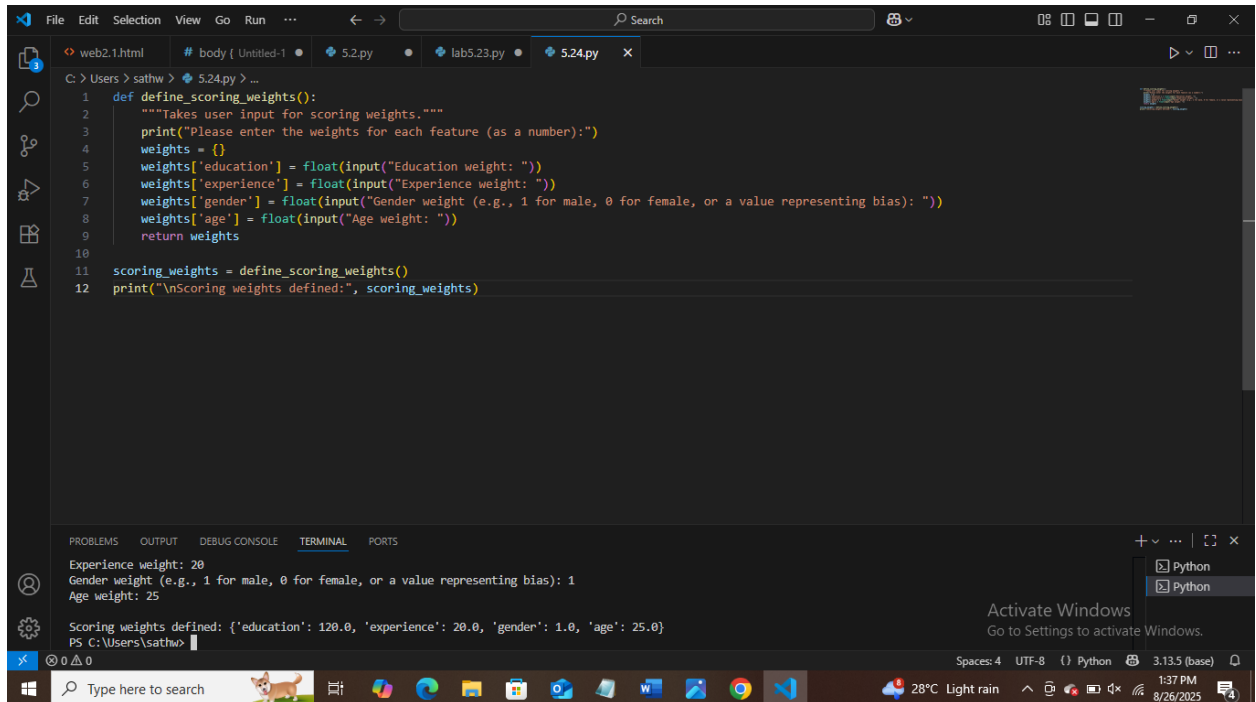
Enter applicant's name: raj
Enter applicant's annual income: 12000
Enter applicant's credit score: 2
Enter desired loan amount: 100000
Loan denied for raj
PS C:\Users\sathw>

EXPLANATION:

- Defines a recursive function `recursive_fibonacci(n)` with a base case for $n \leq 1$ and a recursive step where the n th 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



```

1 def define_scoring_weights():
2     """Takes user input for scoring weights."""
3     print("Please enter the weights for each feature (as a number):")
4     weights = {}
5     weights['education'] = float(input("Education weight: "))
6     weights['experience'] = float(input("Experience weight: "))
7     weights['gender'] = float(input("Gender weight (e.g., 1 for male, 0 for female, or a value representing bias): "))
8     weights['age'] = float(input("Age weight: "))
9     return weights
10
11 scoring_weights = define_scoring_weights()
12 print("\nScoring weights defined:", scoring_weights)

```

Terminal Output:

```

Experience weight: 20
Gender weight (e.g., 1 for male, 0 for female, or a value representing bias): 1
Age weight: 25

Scoring weights defined: {'education': 120.0, 'experience': 20.0, 'gender': 1.0, 'age': 25.0}

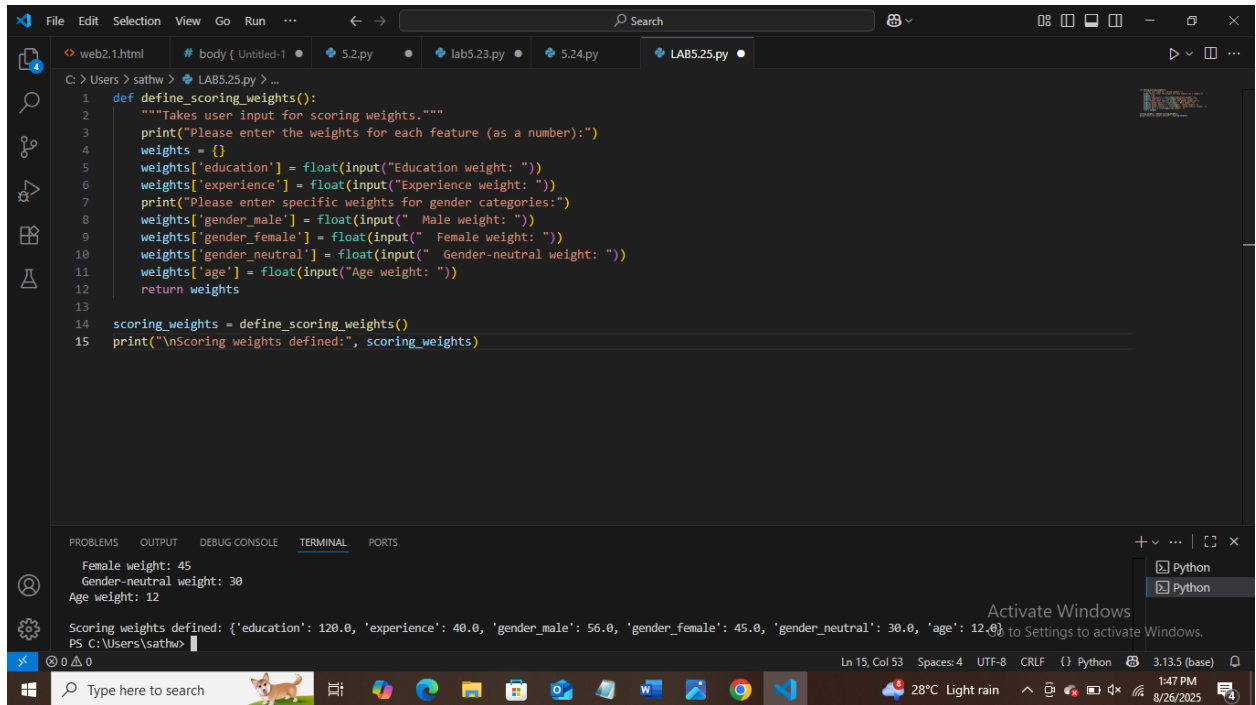
```

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



The screenshot shows a Visual Studio Code editor window with a Python file named `LAB5.25.py`. The code defines a function `define_scoring_weights()` that prompts the user for weights for various features. The function returns a dictionary of these weights. Below the function definition, the function is called, and the resulting dictionary is printed to the console.

```
1 def define_scoring_weights():
2     """Takes user input for scoring weights."""
3     print("Please enter the weights for each feature (as a number):")
4     weights = {}
5     weights['education'] = float(input("Education weight: "))
6     weights['experience'] = float(input("Experience weight: "))
7     print("Please enter specific weights for gender categories:")
8     weights['gender_male'] = float(input(" Male weight: "))
9     weights['gender_female'] = float(input(" Female weight: "))
10    weights['gender_neutral'] = float(input(" Gender-neutral weight: "))
11    weights['age'] = float(input("Age weight: "))
12    return weights
13
14 scoring_weights = define_scoring_weights()
15 print("\nScoring weights defined:", scoring_weights)
```

The terminal output shows the following input and output:

```
Female weight: 45
Gender-neutral weight: 30
Age weight: 12

Scoring weights defined: {'education': 120.0, 'experience': 40.0, 'gender_male': 50.0, 'gender_female': 45.0, 'gender_neutral': 30.0, 'age': 12.0}
```

EXPLANATION:

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