Assignment 5.2

Task-1

Code:

A screenshot of a computer program

AI-generated content may be incorrect.

Output:

A screen shot of a computer

AI-generated content may be incorrect.

Explanation:

This notebook demonstrates a very basic example of user creation and login functionality.

The first code cell is empty and doesn't contain any code.

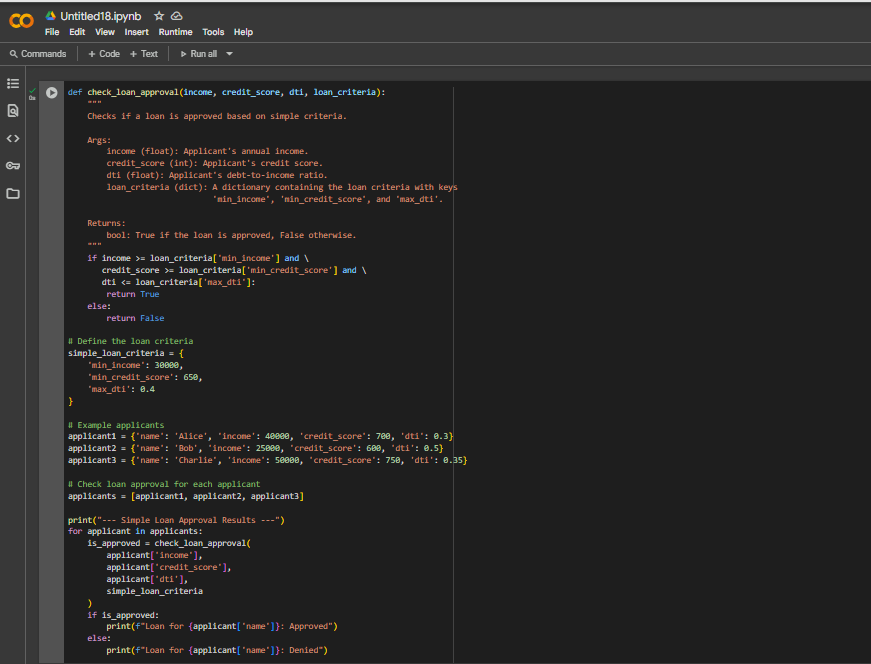
The markdown cell titled "Disclaimer" clearly states that the provided code is for demonstration purposes only and should *not* be used in a production environment due to significant security vulnerabilities. It highlights the need for more robust security measures.

The second code cell defines two functions:

* create\_user(username, password): This function simulates creating a new user. It takes a username and password, hashes the password using SHA-256 (though the markdown cell correctly points out that stronger algorithms should be used in a real application), and stores the username and hashed password in a dictionary called users (which simulates a database).
* login(username, password): This function simulates logging in a user. It checks if the username exists in the users dictionary and then compares the hash of the input password with the stored hashed password. It prints a success message if the credentials match and an error message otherwise.

The markdown cell titled "Review of the generated code for security issues" provides a critical analysis of the code from a security perspective. It points out that while the code avoids hardcoded passwords and doesn't store them in plain text, it uses a weak hashing algorithm (SHA-256) without salting, which is a major security flaw. It also mentions the lack of encryption and lists additional security considerations necessary for a real-world system.

Task-2

Code: 

Output:

A screenshot of a computer

AI-generated content may be incorrect.

Explanation:

Of course! I can give you a line-by-line explanation of the simple loan approval system code.

Here is the explanation for each line in the code cell 1db3d80b:

def check\_loan\_approval(income, credit\_score, dti, loan\_criteria):

This line defines a function named check\_loan\_approval. This function takes four arguments: income, credit\_score, dti (Debt-to-Income ratio), and loan\_criteria.

    """  
    Checks if a loan is approved based on simple criteria.  
  
    Args:  
        income (float): Applicant's annual income.  
        credit\_score (int): Applicant's credit score.  
        dti (float): Applicant's debt-to-income ratio.  
        loan\_criteria (dict): A dictionary containing the loan criteria with keys  
                              'min\_income', 'min\_credit\_score', and 'max\_dti'.  
  
    Returns:  
        bool: True if the loan is approved, False otherwise.  
    """

This is a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns). It's good practice to include these in your functions.

    if income >= loan\_criteria['min\_income'] and \  
       credit\_score >= loan\_criteria['min\_credit\_score'] and \  
       dti <= loan\_criteria['max\_dti']:

This is the core logic of the function. It's an if statement that checks three conditions joined by and:

* income >= loan\_criteria['min\_income']: Checks if the applicant's income is greater than or equal to the minimum income specified in the loan\_criteria dictionary.
* credit\_score >= loan\_criteria['min\_credit\_score']: Checks if the applicant's credit score is greater than or equal to the minimum credit score in loan\_criteria.
* dti <= loan\_criteria['max\_dti']: Checks if the applicant's debt-to-income ratio is less than or equal to the maximum DTI in loan\_criteria. The \ at the end of the first two lines allows the condition to be split across multiple lines for readability.

        return True

If all the conditions in the if statement are True (meaning the applicant meets all the criteria), the function returns the boolean value True.

    else:  
        return False

If any of the conditions in the if statement are False, the else block is executed, and the function returns the boolean value False, indicating the loan is denied.

# Define the loan criteria  
simple\_loan\_criteria = {  
    'min\_income': 30000,  
    'min\_credit\_score': 650,  
    'max\_dti': 0.4  
}

This section defines a dictionary called simple\_loan\_criteria. It stores the specific numerical values for the minimum income, minimum credit score, and maximum debt-to-income ratio that will be used to evaluate loan applications.

# Example applicants  
applicant1 = {'name': 'Alice', 'income': 40000, 'credit\_score': 700, 'dti': 0.3}  
applicant2 = {'name': 'Bob', 'income': 25000, 'credit\_score': 600, 'dti': 0.5}  
applicant3 = {'name': 'Charlie', 'income': 50000, 'credit\_score': 750, 'dti': 0.35}

These lines create three dictionaries, each representing a different loan applicant. Each dictionary contains the applicant's name and their financial details (income, credit score, and dti).

# Check loan approval for each applicant  
applicants = [applicant1, applicant2, applicant3]

This line creates a list named applicants that contains the three applicant dictionaries defined above. This makes it easy to loop through each applicant.

print("--- Simple Loan Approval Results ---")

This line simply prints a header to the console to make the output more organized.

for applicant in applicants:

This is a for loop that iterates through each dictionary (applicant) in the applicants list. In each iteration, the current applicant's dictionary is assigned to the variable applicant.

    is\_approved = check\_loan\_approval(  
        applicant['income'],  
        applicant['credit\_score'],  
        applicant['dti'],  
        simple\_loan\_criteria  
    )

Inside the loop, this line calls the check\_loan\_approval function for the current applicant. It passes the applicant's income, credit score, and DTI (accessed using dictionary keys) and the simple\_loan\_criteria dictionary to the function. The boolean result (True or False) returned by the function is stored in the variable is\_approved.

    if is\_approved:  
        print(f"Loan for {applicant['name']}: Approved")  
    else:  
        print(f"Loan for {applicant['name']}: Denied")

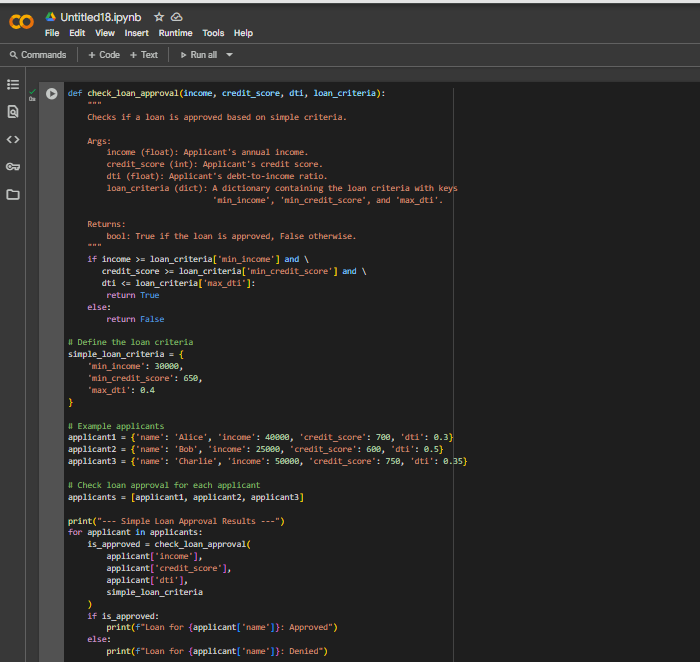
This if-else block checks the value of the is\_approved variable.

* If is\_approved is True, it prints a message indicating that the loan for the applicant (using their name from the dictionary) is "Approved".
* If is\_approved is False, it prints a message indicating that the loan for the applicant is "Denied".

This loop continues until all applicants in the list have been processed.

Task:3

Code:

Output:

A screen shot of a black screen

AI-generated content may be incorrect.

Explanation:

Of course! I can give you a line-by-line explanation of the simple loan approval system code.

Here is the explanation for each line in the code cell 1db3d80b:

def check\_loan\_approval(income, credit\_score, dti, loan\_criteria):

This line defines a function named check\_loan\_approval. This function takes four arguments: income, credit\_score, dti (Debt-to-Income ratio), and loan\_criteria.

    """  
    Checks if a loan is approved based on simple criteria.  
  
    Args:  
        income (float): Applicant's annual income.  
        credit\_score (int): Applicant's credit score.  
        dti (float): Applicant's debt-to-income ratio.  
        loan\_criteria (dict): A dictionary containing the loan criteria with keys  
                              'min\_income', 'min\_credit\_score', and 'max\_dti'.  
  
    Returns:  
        bool: True if the loan is approved, False otherwise.  
    """

This is a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns). It's good practice to include these in your functions.

    if income >= loan\_criteria['min\_income'] and \  
       credit\_score >= loan\_criteria['min\_credit\_score'] and \  
       dti <= loan\_criteria['max\_dti']:

This is the core logic of the function. It's an if statement that checks three conditions joined by and:

* income >= loan\_criteria['min\_income']: Checks if the applicant's income is greater than or equal to the minimum income specified in the loan\_criteria dictionary.
* credit\_score >= loan\_criteria['min\_credit\_score']: Checks if the applicant's credit score is greater than or equal to the minimum credit score in loan\_criteria.
* dti <= loan\_criteria['max\_dti']: Checks if the applicant's debt-to-income ratio is less than or equal to the maximum DTI in loan\_criteria. The \ at the end of the first two lines allows the condition to be split across multiple lines for readability.

        return True

If all the conditions in the if statement are True (meaning the applicant meets all the criteria), the function returns the boolean value True.

    else:  
        return False

If any of the conditions in the if statement are False, the else block is executed, and the function returns the boolean value False, indicating the loan is denied.

# Define the loan criteria  
simple\_loan\_criteria = {  
    'min\_income': 30000,  
    'min\_credit\_score': 650,  
    'max\_dti': 0.4  
}

This section defines a dictionary called simple\_loan\_criteria. It stores the specific numerical values for the minimum income, minimum credit score, and maximum debt-to-income ratio that will be used to evaluate loan applications.

# Example applicants  
applicant1 = {'name': 'Alice', 'income': 40000, 'credit\_score': 700, 'dti': 0.3}  
applicant2 = {'name': 'Bob', 'income': 25000, 'credit\_score': 600, 'dti': 0.5}  
applicant3 = {'name': 'Charlie', 'income': 50000, 'credit\_score': 750, 'dti': 0.35}

These lines create three dictionaries, each representing a different loan applicant. Each dictionary contains the applicant's name and their financial details (income, credit score, and dti).

# Check loan approval for each applicant  
applicants = [applicant1, applicant2, applicant3]

This line creates a list named applicants that contains the three applicant dictionaries defined above. This makes it easy to loop through each applicant.

print("--- Simple Loan Approval Results ---")

This line simply prints a header to the console to make the output more organized.

for applicant in applicants:

This is a for loop that iterates through each dictionary (applicant) in the applicants list. In each iteration, the current applicant's dictionary is assigned to the variable applicant.

    is\_approved = check\_loan\_approval(  
        applicant['income'],  
        applicant['credit\_score'],  
        applicant['dti'],  
        simple\_loan\_criteria  
    )

Inside the loop, this line calls the check\_loan\_approval function for the current applicant. It passes the applicant's income, credit score, and DTI (accessed using dictionary keys) and the simple\_loan\_criteria dictionary to the function. The boolean result (True or False) returned by the function is stored in the variable is\_approved.

    if is\_approved:  
        print(f"Loan for {applicant['name']}: Approved")  
    else:  
        print(f"Loan for {applicant['name']}: Denied")

This if-else block checks the value of the is\_approved variable.

* If is\_approved is True, it prints a message indicating that the loan for the applicant (using their name from the dictionary) is "Approved".
* If is\_approved is False, it prints a message indicating that the loan for the applicant is "Denied".

This loop continues until all applicants in the list have been processed.

Task-3

Code:

A screenshot of a computer program

AI-generated content may be incorrect.

Output:

A grey screen with white text

AI-generated content may be incorrect.

Explanation:

def recursive\_fibonacci(n):

"""

Calculates the nth Fibonacci number using recursion.

Args:

n: The desired position in the Fibonacci sequence (a non-negative integer).

Returns:

The nth Fibonacci number.

Raises:

ValueError: If n is a negative integer.

"""

# This is the start of the function definition for recursive\_fibonacci which takes an integer 'n' as input.

if n < 0:

# This checks if the input 'n' is a negative integer.

raise ValueError("Input must be a non-negative integer")

# If 'n' is negative, a ValueError is raised with a message indicating that the input must be non-negative.

elif n <= 1:

# This checks if 'n' is less than or equal to 1. These are the base cases for the recursion.

return n # Base cases: Fibonacci(0) is 0, Fibonacci(1) is 1

# If 'n' is 0 or 1, the function returns 'n' directly, as these are the defined starting points of the Fibonacci sequence.

else:

# This is the recursive step for when 'n' is greater than 1.

# Recursive step: Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)

# This line is the core of the recursion. It calculates the Fibonacci number for 'n' by adding the Fibonacci numbers of 'n-1' and 'n-2'.

return recursive\_fibonacci(n-1) + recursive\_fibonacci(n-2)

# The function calls itself with smaller arguments (n-1 and n-2) until it reaches the base cases (n <= 1).

# Example usage:

# This is a comment indicating the start of the example usage section.

n = 10

# This line assigns the integer 10 to the variable 'n', which will be used as the input to the function.

fib\_number = recursive\_fibonacci(n)

# This line calls the recursive\_fibonacci function with 'n' (which is 10) 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 using an f-string to include the value of 'n' and 'fib\_number' in a formatted output.

# Example with negative input (will raise an error)

# This is a comment indicating an example of how to handle negative input, which is commented out.

# try:

# This is the start of a try block to handle potential exceptions.

# recursive\_fibonacci(-5)

# This line attempts to call the function with a negative input (-5), which is expected to raise a ValueError.

# except ValueError as e:

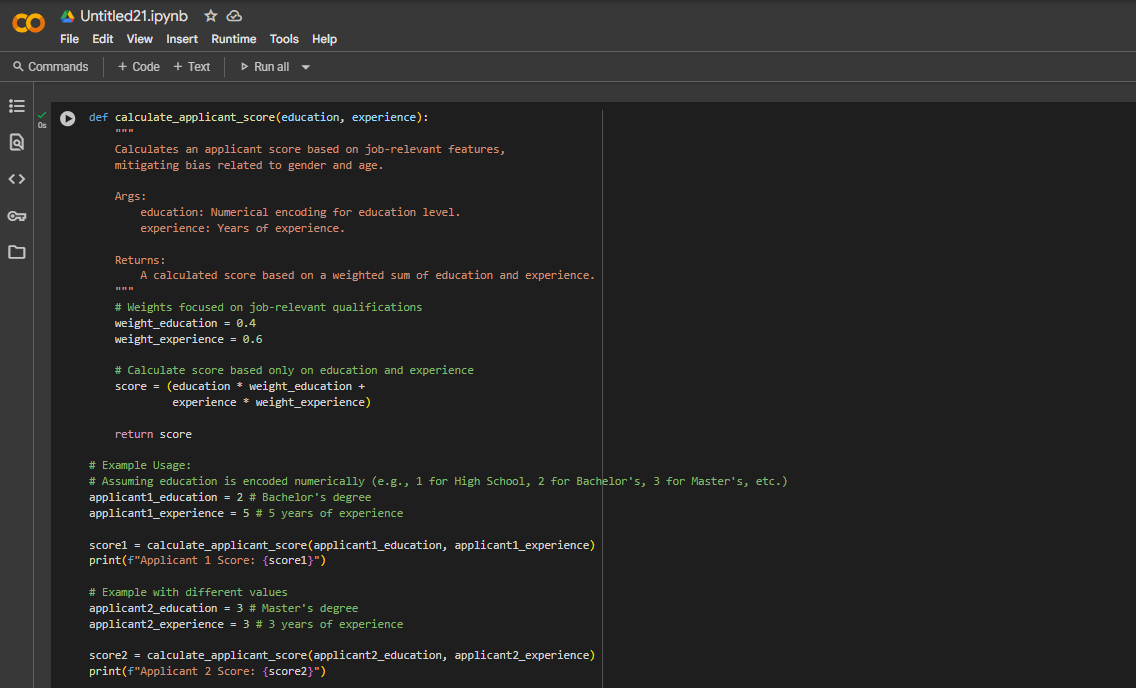
# This block catches the ValueError if it is raised.

# print(f"Error: {e}")

# If a ValueError is caught, this line prints an error message including the exception details.

Task:4

Code:



Output:

A screenshot of a computer

AI-generated content may be incorrect.

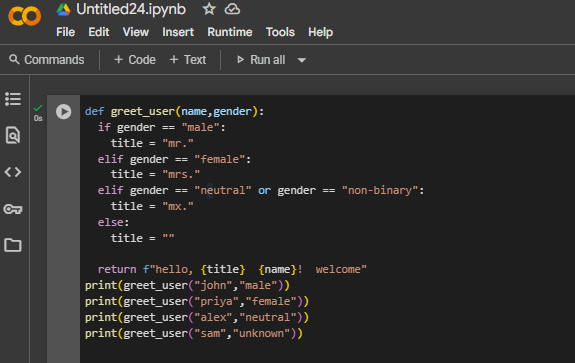
Explanation:

1. def calculate\_applicant\_score(education, experience):: This line defines a Python function named calculate\_applicant\_score that takes two arguments: education and experience.
2. """ ... """: This is a docstring, which provides a description of what the function does, its arguments (Args), and what it returns (Returns). This helps explain the function's purpose.
3. # Weights focused on job-relevant qualifications: This is a comment explaining the purpose of the next two lines.
4. weight\_education = 0.4: This line sets a variable weight\_education to 0.4. This value represents how much importance or weight is given to the education feature in the scoring.
5. weight\_experience = 0.6: This line sets a variable weight\_experience to 0.6. This value represents how much importance or weight is given to the experience feature in the scoring.
6. # Calculate score based only on education and experience: This is a comment explaining the purpose of the next line.
7. score = (education \* weight\_education + experience \* weight\_experience): This is the core of the scoring logic. It calculates the score by multiplying the education value by its weight (weight\_education) and adding it to the experience value multiplied by its weight (weight\_experience). This is a simple weighted sum.
8. return score: This line returns the calculated score as the output of the function.
9. # Example Usage:: This is a comment indicating that the following lines demonstrate how to use the function.
10. # Assuming education is encoded numerically (e.g., 1 for High School, 2 for Bachelor's, 3 for Master's, etc.): This is a comment explaining the numerical encoding assumption for the education feature in the examples.
11. applicant1\_education = 2 # Bachelor's degree: This line creates a variable applicant1\_education and assigns it the value 2, representing a Bachelor's degree for the first example applicant. The comment clarifies what the value 2 represents.
12. applicant1\_experience = 5 # 5 years of experience: This line creates a variable applicant1\_experience and assigns it the value 5, representing 5 years of experience for the first example applicant. The comment clarifies what the value 5 represents.
13. score1 = calculate\_applicant\_score(applicant1\_education, applicant1\_experience): This line calls the calculate\_applicant\_score function with the education and experience values for the first applicant and stores the returned score in the score1 variable.
14. print(f"Applicant 1 Score: {score1}"): This line prints the calculated score for Applicant 1 to the console. The f"" syntax is a formatted string literal (f-string) that allows you to embed the value of score1 directly within the string.
15. # Example with different values: This is a comment indicating a second example with different values.
16. applicant2\_education = 3 # Master's degree: This line creates a variable applicant2\_education and assigns it the value 3, representing a Master's degree for the second example applicant.
17. applicant2\_experience = 3 # 3 years of experience: This line creates a variable applicant2\_experience and assigns it the value 3, representing 3 years of experience for the second example applicant.
18. score2 = calculate\_applicant\_score(applicant2\_education, applicant2\_experience): This line calls the calculate\_applicant\_score function with the education and experience values for the second applicant and stores the returned score in the score2 variable.
19. print(f"Applicant 2 Score: {score2}"): This line prints the calculated score for Applicant 2 to the console.

This code defines a function to calculate a score based on education and experience, and then provides two examples of how to use that function with different applicant profiles.

Task:5

Code:



Output:

A screenshot of a computer

AI-generated content may be incorrect.

Explanation:

Sure, I can explain this code line by line:

def greet\_user(name,gender):

This line defines a function named greet\_user that takes two arguments: name and gender.

  if gender == "male":  
    title = "mr."

This is an if statement that checks if the value of the gender argument is equal to the string "male". If it is, it assigns the string "mr." to the variable title.

  elif gender == "female":  
    title = "mrs."

This is an elif (else if) statement. If the previous if condition was false, this line checks if the value of the gender argument is equal to the string "female". If it is, it assigns the string "mrs." to the variable title.

  elif gender == "neutral" or gender == "non-binary":  
    title = "mx."

This is another elif statement. If the previous conditions were false, this line checks if the value of the gender argument is either "neutral" or "non-binary". If it is one of these, it assigns the string "mx." to the variable title.

  else:  
    title = ""

This is an else statement. If none of the previous if or elif conditions were true, this line assigns an empty string "" to the variable title.

  return f"hello, {title}  {name}!  welcome"

This line returns a formatted string. It uses an f-string to embed the values of the title and name variables into the string "hello, [title] [name]! welcome".

print(greet\_user("john","male"))  
print(greet\_user("priya","female"))  
print(greet\_user("alex","neutral"))  
print(greet\_user("sam","unknown"))

These lines call the greet\_user function with different name and gender arguments and print the returned string to the console.

Let me know if you have any more questions!