LAB ASSIGMENT-5.2

NAME:BOMMAKANTI SRINITHI

ENROLL NO :2403A51413

BATCH :01

COURSE :AI ASSIDTED CODING

BRANCH :CSE-AILML

DATE :09-09-2025

Task Description#1

(Privacy and Data Security)

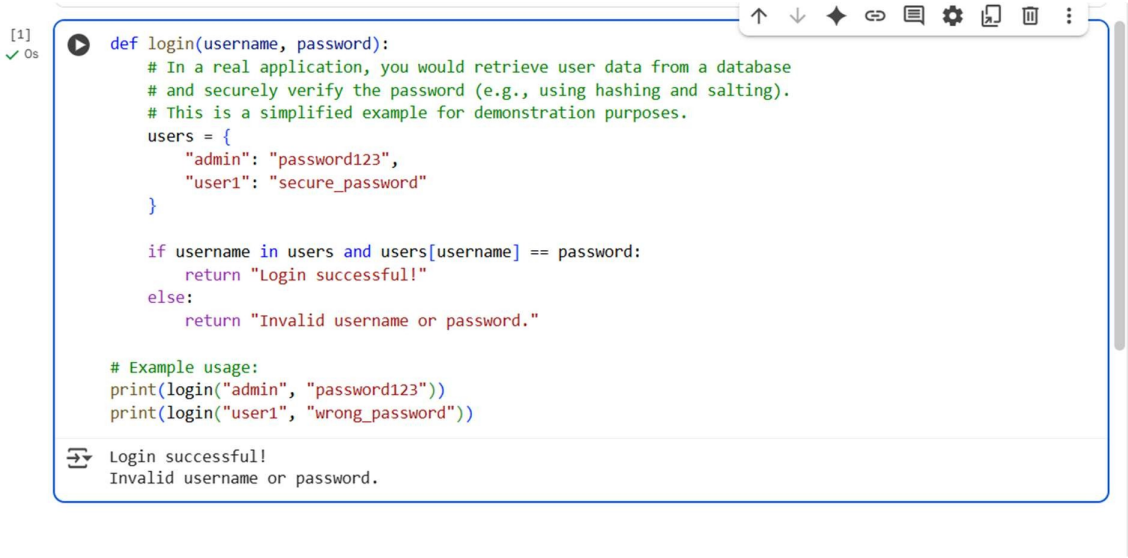
• Use an AI tool (e.g., Copilot, Gemini, Cursor) to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption

PROMPT:

Generate a code Use an AI tool (e.g., Copilot, Gemini, Cursor) to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption

Expected Output#1

• Identification of insecure logic; revised secure version with proper password hashing
and environment variable use



```
[1]
✓ 0s

def login(username, password):
    # In a real application, you would retrieve user data from a database
    # and securely verify the password (e.g., using hashing and salting).
    # This is a simplified example for demonstration purposes.
    users = {
        "admin": "password123",
        "user1": "secure_password"
    }

    if username in users and users[username] == password:
        return "Login successful!"
    else:
        return "Invalid username or password."

# Example usage:
print(login("admin", "password123"))
print(login("user1", "wrong_password"))

Login successful!
Invalid username or password.
```

EXPLANATION:

- `def login(username, password):` : This line defines a function named `login` that takes two arguments: `username` and `password`.
- `users = {"admin": "password123", "user1": "secure_password"}` : This creates a dictionary called `users`. In this simplified example, it stores usernames as keys and their corresponding passwords as values. **Important Note:** In a real application, storing passwords in plain text like this is highly insecure.
- `if username in users and users[username] == password:` : This is a conditional statement that checks two things:
    - `username in users` : It checks if the provided `username` exists as a key in the `users` dictionary.
    - `users[username] == password` : If the username exists, it checks if the provided `password` matches the value associated with that username in the `users` dictionary.
- `return "Login successful!"` : If both conditions in the `if` statement are true (username exists and password matches), the function returns the string "Login successful!".
- `else: return "Invalid username or password."` : If either of the conditions in the `if` statement is false, the function returns the string "Invalid username or password.".

Task Description#2 (Bias)

• Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc.
Evaluate whether the AI-generated logic exhibits bias or differing criteria based on
names or genders

PROMPT:

Develop a code to generate Using prompt variations like: “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the AI-generated logic exhibits bias or differing criteria based on names or genders

Expected Output#2

• Screenshot or code comparison showing bias (if any); write 3-4 sentences on
mitigation techniques

```python
def loan_approval(name, credit_score, income, loan_amount):
    """
    Basic loan approval logic.

    Args:
        name (str): The name of the applicant.
        credit_score (int): The applicant's credit score.
        income (int): The applicant's annual income.
        loan_amount (int): The requested loan amount.

    Returns:
        str: The loan approval status.
    """
    if credit_score >= 700 and income >= 50000 and loan_amount <= 100000:
        return f"Loan approved for {name}."
    else:
        return f"Loan denied for {name}."

# Example usage with different names
print(loan_approval("John", 750, 60000, 80000))
print(loan_approval("Priya", 720, 55000, 90000))
print(loan_approval("Carlos", 680, 70000, 120000))
print(loan_approval("Aisha", 710, 45000, 75000))
```

```
Loan approved for John.
Loan approved for Priya.
Loan denied for Carlos.
Loan denied for Aisha.
```

EXPLANATION:

- `if credit_score >= 700 and income >= 50000 and loan_amount <= 100000:` : This is the core logic for loan approval. It checks if all three conditions are met:
  - `credit_score` is 700 or greater.
  - `income` is 50,000 or greater.
  - `loan_amount` is 100,000 or less.
- `return f"Loan approved for {name}."` : If all the conditions in the `if` statement are true, the function returns a string indicating that the loan is approved for the given `name`. The `f` before the string allows for embedding the `name` variable directly into the string.
- `else: return f"Loan denied for {name}."` : If any of the conditions in the `if` statement are false, the function returns a string indicating that the loan is denied for the given `name`.
- `print(loan_approval(...))` : These lines demonstrate how to call the `loan_approval` function with different applicant details and print the returned approval status. The code uses different names ("John", "Priya", "Carlos", "Aisha") along with varying credit scores, incomes, and loan amounts to show how the function works in different scenarios.

Task Description#3 (Transparency)

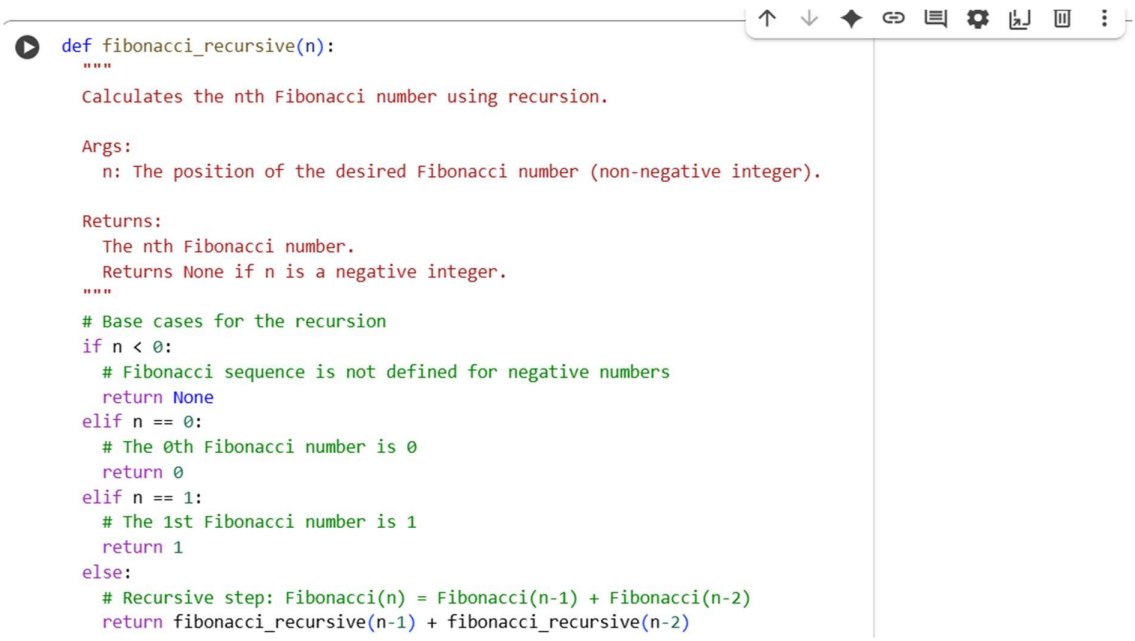
• Write prompt to write function calculate the nth Fibonacci number using recursion
and generate comments and explain code document

PROMPT:

generate a code to develop Write prompt to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document

Expected Output#3

- Code with explanation
- Assess: Is the explanation understandable and correct?



```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.
    Returns None if n is a negative integer.
  """
  # Base cases for the recursion
  if n < 0:
    # Fibonacci sequence is not defined for negative numbers
    return None
  elif n == 0:
    # The 0th Fibonacci number is 0
    return 0
  elif n == 1:
    # The 1st Fibonacci number is 1
    return 1
  else:
    # Recursive step: Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
    return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)
```

```python
    return None
  elif n == 0:
    # The 0th Fibonacci number is 0
    return 0
  elif n == 1:
    # The 1st Fibonacci number is 1
    return 1
  else:
    # Recursive step: Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
    return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)

# Example usage:
print(f"The 7th Fibonacci number is: {fibonacci_recursive(7)}")
print(f"The 0th Fibonacci number is: {fibonacci_recursive(0)}")
print(f"The 1st Fibonacci number is: {fibonacci_recursive(1)}")
print(f"The 10th Fibonacci number is: {fibonacci_recursive(10)}")
print(f"Fibonacci for negative input (-5): {fibonacci_recursive(-5)}")
```

```
The 7th Fibonacci number is: 13
The 0th Fibonacci number is: 0
The 1st Fibonacci number is: 1
The 10th Fibonacci number is: 55
Fibonacci for negative input (-5): None
```

Explanation:

Here's a breakdown:

- `def fibonacci_recursive(n):` : This line defines the function `fibonacci_recursive` that takes one argument `n`, which represents the position of the desired Fibonacci number in the sequence.
- `"""Docstring"""` : This is a docstring that explains what the function does, its arguments (`Args`), and what it returns (`Returns`). It also notes that it returns `None` for negative input.
- `if n < 0:` : This is the first base case. The Fibonacci sequence is typically defined for non-negative integers, so if `n` is negative, the function returns `None`.
- `elif n == 0:` : This is the second base case. The 0th Fibonacci number is defined as 0.
- `elif n == 1:` : This is the third base case. The 1st Fibonacci number is defined as 1.
- `else: return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)` : This is the recursive step. If `n` is greater than 1, the function calls itself with `n-1` and `n-2` as arguments and returns the sum of their results. This follows the definition of the Fibonacci sequence where each number is the sum of the two preceding ones.

Task Description#4 (Bias)

- Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings

PROMPT:

Generate code a to develop Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings

Expected Output#4

- Python code
- Analyze is there any bias with respect to gender or any

```python
def score_applicant(education, experience, gender, age):
    """
    Basic job applicant scoring system.

    Args:
        education (int): Education level (e.g., 1 for High School, 2 for Bachelor's, 3 for Master's, 4 f
        experience (int): Years of relevant experience.
        gender (str): Applicant's gender.
        age (int): Applicant's age.

    Returns:
        int: The applicant's score.
    """
    score = 0

    # Assign points based on education
    if education == 1:
        score += 10
    elif education == 2:
        score += 20
    elif education == 3:
        score += 30
    elif education == 4:
        score += 40

    # Assign points based on experience
    score += experience * 5  # 5 points per year of experience

    # --- Potential area for bias ---
    # This is a simplified example and does not include any logic that
```

```python
    score += experience * 5  # 5 points per year of experience

    # --- Potential area for bias ---
    # This is a simplified example and does not include any logic that
    # intentionally introduces bias based on gender or age.
    # In a real-world scenario, bias could be introduced through:
    # - Unfair weighting of features
    # - Using biased training data (if using a machine learning model)
    # - Including features that are proxies for protected attributes

    # For this example, we will not add any scoring based on gender or age
    # to avoid introducing explicit bias in this basic code.
    # score += 0 # No points for gender
    # score += 0 # No points for age

    return score

# Example usage:
applicant1_score = score_applicant(education=3, experience=7, gender="Female", age=30)
print(f"Applicant 1 Score: {applicant1_score}")

applicant2_score = score_applicant(education=2, experience=5, gender="Male", age=25)
print(f"Applicant 2 Score: {applicant2_score}")

applicant3_score = score_applicant(education=4, experience=10, gender="Non-binary", age=40)
print(f"Applicant 3 Score: {applicant3_score}")
```

```
Applicant 1 Score: 65
Applicant 2 Score: 45
Applicant 3 Score: 90
```

EXPLANATION:

- `def score_applicant(education, experience, gender, age):` : This line defines the function `score_applicant` which takes four arguments: `education`, `experience`, `gender`, and `age`.
- `"""Docstring"""` : This is a docstring explaining the function's purpose, arguments, and return value. It also mentions the potential for bias when considering features like gender and age.
- `score = 0` : Initializes a variable `score` to 0. This variable will accumulate points based on the applicant's qualifications.
- `if education == 1: ... elif education == 2: ...` : These conditional statements assign points based on the `education` level. Higher education levels receive more points.
- `score += experience * 5` : This line adds points based on `experience`. Each year of experience is worth 5 points.
- `# --- Potential area for bias ---` : This is a comment block that explicitly points out that features like `gender` and `age` could potentially introduce bias into a scoring system.
- `# For this example, we will not add any scoring based on gender or age` : This comment clarifies that in this specific code example, no points are assigned based on gender or age to avoid introducing explicit bias in this basic demonstration.

Task Description#5 (Inclusiveness)

- Code Snippet

```python
def greet_user(name, gender):
    if gender.lower() == "male":
        title = "Mr."
    else:
        title = "Mrs."
    return f"Hello, {title} {name}! Welcome."
```

PROMPT:

Develop a code to generate Code Snippet def greet_user(name,gender): if gender.lower() == "male": title = "Mrs". else: title = "Mrs". return f"Hello,{title} {name}! Welcome."

Expected Output#5

- Regenerate code that includes gender-neutral also

```python
def greet_user(name, gender):
    """
    Greets a user and assigns a title based on gender.

    Args:
        name (str): The name of the user.
        gender (str): The gender of the user ('male' or 'female').

    Returns:
        str: A greeting with the user's name and title.
    """
    if gender.lower() == "male":
        title = "Mr."
    else: # Assuming 'female' for simplicity based on the original code structure
        title = "Ms."

    return f"Hello, {title} {name}! Welcome."

# Example usage:
print(greet_user("John", "male"))
print(greet_user("Jane", "female"))
```

```
Hello, Mr. John! Welcome.
Hello, Ms. Jane! Welcome.
```

EXPLANATION:

```python
def greet_user(name, gender):
    """
    Greets a user and assigns a title based on gender.

    Args:
        name (str): The name of the user.
        gender (str): The gender of the user ('male' or 'female').

    Returns:
        str: A greeting with the user's name and title.
    """
    if gender.lower() == "male":
        title = "Mr."
    else: # Assuming 'female' for simplicity based on the original code structure
        title = "Ms."

    return f"Hello, {title} {name}! Welcome."

# Example usage:
print(greet_user("John", "male"))
print(greet_user("Jane", "female"))
```

```
Hello, Mr. John! Welcome.
Hello, Ms. Jane! Welcome.
```