ASSIGNMENT-10.4

TASK-1

CODE:

A white background with black text

AI-generated content may be incorrect.

OUTPUT:

A white background with a black border

AI-generated content may be incorrect.

EXPLANATION:

A close up of a text

AI-generated content may be incorrect.

TASK-2

CODE:

A screenshot of a computer code

AI-generated content may be incorrect.

OUTPUT;

A number in a box

AI-generated content may be incorrect.

EXPLANATION:

A screenshot of a computer

AI-generated content may be incorrect.

TASK-3

CODE:

A screenshot of a computer code

AI-generated content may be incorrect.

OUTPUT:



EXPLANATION:

**Changes Made:**

1. **Function Name:** The function name was changed from a non-descriptive c to a clear and understandable calculate\_factorial. This makes the code's purpose immediately obvious.
2. **Variable Names:** The variable names were changed from x to result and n to number for better readability and clarity.
3. **Indentation:** Proper indentation was added to clearly define the function block and the loop within it. This adheres to PEP 8 guidelines and makes the code structure easy to follow.
4. **Docstring:** A docstring """Calculates the factorial of a non-negative integer.""" was added to explain what the function does. This is good practice for code documentation.
5. **Example Usage:** Added an example usage section to demonstrate how to call the function and print the result in a user-friendly format using an f-string.

**How the code works:**

* The calculate\_factorial(number) function takes a non-negative integer number as input.
* It initializes a variable result to 1. This is because the factorial of 0 is 1, and it serves as the base case for the multiplication.
* It then iterates through a for loop from 1 up to and including the input number.
* In each iteration, it multiplies the current result by the loop counter i and updates the result.
* Finally, after the loop finishes, the function returns the calculated result, which is the factorial of the input number.
* The example usage sets input\_number to 5, calls the function, stores the result in factorial\_result, and then prints the result using an f-string for formatted output.

These changes make the code much easier to read, understand, and maintain, while still producing the correct factorial calculation.

TASK-4:

CODE:

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

OUTPUT:

A white background with black text

AI-generated content may be incorrect.

EXPLANATION:

**Security Practice (Preventing SQL Injection):**

* **Parameterized Query:** The most significant security improvement is in the cursor.execute(query, (user\_id,)) line. Instead of directly embedding the user\_id variable into the SQL query string using an f-string (which was the vulnerability in the original code), we now use a placeholder ? in the query string. The actual user\_id is passed as a second argument to the execute method in a tuple (user\_id,). SQLite3 and other database connectors are designed to handle these parameterized queries securely. They ensure that the value passed as an argument is treated purely as data and not as executable SQL code, effectively preventing SQL injection attacks.

**Exception Handling:**

* **try...except...finally block:** The database interaction code is wrapped in a try...except...finally block. This is a standard Python construct for handling potential errors.
  + **try:** The code that might raise an exception (connecting to the database, executing the query, fetching results) is placed inside the try block.
  + **except sqlite3.Error as e::** This block specifically catches exceptions that are subclasses of sqlite3.Error. These are errors related to the SQLite database operations themselves (e.g., table not found, syntax errors in the query). If a sqlite3.Error occurs, it prints an informative error message including the specific error e.
  + **except Exception as e::** This is a broader exception handler that catches any other type of Exception that might occur within the try block. This provides a fallback for unexpected errors.
  + **finally::** The code inside the finally block is *always* executed, regardless of whether an exception occurred or not. In this case, if conn: conn.close() ensures that the database connection is closed properly. This is crucial for releasing resources and preventing potential issues like database locks.
* **Return None on Error:** In both except blocks, the function returns None. This signals to the calling code that the database operation failed to retrieve the user data.

**Input Validation:**

* **if user\_input.isdigit()::** Before attempting to use the user\_input in the database query, the code now includes basic input validation to check if the input consists only of digits. This helps prevent errors if the user enters non-numeric input and adds another layer of robustness.

By incorporating these security and exception handling practices, the code becomes more robust, secure, and less prone to crashes due to unexpected inputs or database issues.

TASK-5

CODE:

A screenshot of a computer code

AI-generated content may be incorrect.

OUTPUT:

A close-up of black text

AI-generated content may be incorrect.

EXPLANATION:

**Code Review Report Analysis:**

The original code had a few areas that could be improved for readability, maintainability, and robustness:

1. **Poor Naming:** The function name calc and the parameter names x, y, and z are not descriptive. It's hard to tell at a glance what the function does or what the parameters represent.
2. **Inconsistent Formatting and Indentation:** The indentation is inconsistent, especially in the elif z=="sub": return x-y line, which makes the code harder to read and understand the code structure. While Python is flexible with single-line statements, consistent indentation is crucial for readability and following PEP 8.
3. **Lack of Error Handling for Division by Zero:** The code does not handle the case where the operation is "div" and the second number (y) is zero. This would result in a ZeroDivisionError and crash the program.
4. **Handling of Invalid Operations:** For invalid operations, the code just prints "wrong". It would be better to provide a more informative error message and potentially return a specific value (like None) or raise an exception to indicate that the operation failed.
5. **No Docstring:** The function lacks a docstring, which is a good practice to explain what the function does, its parameters, and what it returns.

**Improved Version Explanation:**

The improved code addresses these points:

# Improved version (Illustrative)  
def perform\_operation(num1, num2, operation):  
    """Performs a basic arithmetic operation based on the provided string."""  
    if operation == "add":  
        return num1 + num2  
    elif operation == "sub":  
        return num1 - num2  
    elif operation == "mul":  
        return num1 \* num2  
    elif operation == "div":  
        if num2 == 0:  
            print("Error: Division by zero.")  
            return None  # Or raise a ValueError  
        return num1 / num2  
    else:  
        print(f"Error: Invalid operation '{operation}'.")  
        return None # Or raise a ValueError  
  
# Example usage:  
print(perform\_operation(10, 5, "add"))  
print(perform\_operation(10, 0, "div"))  
print(perform\_operation(10, 5, "mod")) # Example of invalid operation

1. **Clearer Naming:** The function is renamed to perform\_operation, and parameters are named num1, num2, and operation. This makes the code's purpose and parameters much clearer.
2. **Consistent Indentation:** The indentation is consistent throughout the function, following standard Python practices and PEP 8.
3. **Division by Zero Handling:** An if num2 == 0: check is added within the "div" block. If num2 is 0, it prints an error message and returns None (or you could raise a ValueError for more explicit error handling).
4. **Improved Invalid Operation Handling:** The else block now prints a more informative error message using an f-string, indicating which operation was invalid. It also returns None to signify that the operation was not successful.
5. **Docstring Added:** A docstring """Performs a basic arithmetic operation based on the provided string.""" is included to explain the function's purpose.
6. **Example Usage:** The example usage demonstrates calling the function with different operations, including the division by zero case and an invalid operation, to show how the error handling works.

In summary, the improved version is more readable, maintainable, and robust due to better naming, consistent formatting, and proper error handling.

\* \* \* \* \*