**ASSIGNMENT – 10.4**

**NAME :** SAI CHARAN .P

**ROLL:NO :** 2403A52343

**BATCH :** AI 13

**TASK-1: Syntax and Error Detection**

**Wrong code:**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Corrected code:**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Error Explanation:**

The original code had the following errors:

1. SyntaxError: expected ':': This error occurred because the function definition line def add\_numbers(a, b) was missing a colon : at the end. In Python, function definitions must end with a colon.
2. Typo in return statement: The code had return reslt instead of return result. The variable result was correctly assigned the sum of a and b, but the function attempted to return a variable named reslt, which was not defined.
3. Missing comma in function call: The function was called as print(add\_numbers(10 20)). In Python, arguments in a function call must be separated by commas. It should have been print(add\_numbers(10, 20)).

**TASK-2: Logical and Performance Issue Review**

**Given Code:**

A screenshot of a computer code

AI-generated content may be incorrect.

**Efficient and improvised code:**

A screenshot of a computer code

AI-generated content may be incorrect.

**Explanation of optimization:**

**Explanation of Optimizations in find\_duplicates\_efficient**

1. **Use of Sets for Fast Lookup:**
   * The function uses two sets: seen and duplicates.
   * Sets in Python provide **O(1) average time complexity** for membership tests (in operations).
   * This is much faster than searching through a list (which is O(n)).
2. **Single Pass Through the List:**
   * The function loops through the input list nums only **once**.
   * During this single pass, it checks whether the current number has been seen before.
   * This reduces the time complexity from **O(n²)** in the original nested loop approach to **O(n)**.
3. **Avoiding Redundant Checks:**
   * When a number is encountered for the first time, it is added to the seen set.
   * When a number is encountered again (meaning it’s a duplicate), it is added to the duplicates set.
   * This avoids repeatedly scanning or appending the same duplicate multiple times.
4. **Efficient Data Structure Usage:**
   * Using sets for both seen and duplicates ensures that duplicates are stored uniquely without additional checks.
   * Conversion of the duplicates set to a list at the end allows the function to return the result in the expected list format.

**TASK-3: Code Refactoring for Readability**

**Given Code:**

A screenshot of a computer

AI-generated content may be incorrect.

**Clean and refactored Code:**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Explanation:**

Explanation of Refactoring Changes

1. Function Renaming:
   * The original function was named c, which is unclear.
   * Renamed to calculate\_factorial to clearly describe its purpose.
2. Proper Indentation:
   * The original code had inconsistent and improper indentation.
   * Fixed indentation to follow Python’s standard (4 spaces per level), improving readability and preventing syntax errors.
3. Meaningful Variable Names:
   * Changed variable x to result to better represent what it stores — the ongoing factorial calculation.
   * Changed parameter n to number for clarity.
4. Docstring Added:
   * Added a docstring explaining what the function does, its parameter, and its return value.
   * This improves maintainability and helps other programmers understand the function quickly.
5. PEP 8 Compliance:
   * Followed naming conventions and spacing guidelines from PEP 8, the Python style guide.
   * Ensured spaces around operators (result \*= i) and after commas.
6. Code Readability:
   * The refactored code is easier to read and understand.
   * Clear structure and comments reduce cognitive load and improve maintainability.

**TASK-4: Security and Error Handling Enhancement**

**Given Code:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Corrected 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.

A screenshot of a computer code

AI-generated content may be incorrect.

**Explanation:**

**What changed and why (security & reliability)**

1. **Parameterized SQL (? placeholders):**
   * The query uses cursor.execute(sql, (user\_id,)) so user input never gets concatenated into SQL text. This prevents SQL injection.
2. **Input validation and type checking:**
   * The prompt code ensures the user\_id is a non‑negative integer before converting and sending it to the DB. Rejecting non‑numeric input early reduces error surface.
3. **Context manager for connection (with sqlite3.connect(...) as conn):**
   * Ensures the connection is committed/closed automatically even if an exception occurs, avoiding resource leaks.
4. **Error handling with try/except:**
   * Catches sqlite3.Error exceptions and returns None on failure (and prints/logs a message). This prevents uncaught exceptions and allows the caller to handle the error state.
5. **Row factory and predictable return shape:**
   * conn.row\_factory = sqlite3.Row allows column-name access; results are converted to plain dict objects, which are easier to consume outside the function.
6. **Clear return contract:**
   * Function returns List[Dict] on success, empty list when no rows match, and None on DB error — easier for callers to handle.
7. **No sensitive info leakage:**
   * The error message printed is generic. In production you’d log detailed errors to a secure log instead of printing them to STDOUT.

**TASK-5: Automated Code Review Report Generation**

**Given Code:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Corrected Code:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Explanation:**

Code Review :-

1. Function Name Improved

* Original: calc
* Corrected: calculate
* Reason: The original name was too short and vague. The new name clearly describes the function's purpose, improving code readability and clarity.

2. Descriptive Variable Names

* Original: x, y, z
* Corrected: num1, num2, operation
* Reason: Meaningful variable names make the code easier to understand and maintain. It’s clear now which variables are numbers and which one is the operation type.

3. Added a Function Docstring

* Change: A multi-line docstring was added at the start of the function.
* Reason: The docstring explains what the function does, its parameters, return value, and possible exceptions. This is essential for documentation and helps future developers understand the code quickly.

4. Proper Indentation and Formatting (PEP 8 Compliance)

* Original: Mixed indentation and inconsistent line formatting (e.g., inline return statements).
* Corrected: Consistent 4-space indentation, and all conditional blocks use separate lines for clarity.
* Reason: Following PEP 8 (Python's style guide) ensures readability and consistency across Python codebases.

5. Error Handling for Division by Zero

* Problem: The original code did not handle division by zero, which would crash the program.
* Fix: A check was added for num2 == 0 before division, and it raises a ZeroDivisionError if true.
* Reason: Prevents runtime errors and makes the function safer to use.

6. Handling Invalid Operations

* Original Behavior: If the operation was unknown, the function printed "wrong" and returned None.
* Corrected Behavior: The function now raises a ValueError with a descriptive message.
* Reason: This is a more robust way to handle invalid input and ensures the caller can handle the error appropriately.

7. Try-Except Block in Main Code

* Added: A try block around function calls, with except blocks for ValueError and ZeroDivisionError.
* Reason: Prevents the program from crashing on errors and provides a user-friendly error message instead.