**LAB ASSIGNMENT-10.4**

**Task 1: Syntax and Error Detection**

**Task:** Identify and fix syntax, indentation, and variable errors in the given script.

# buggy\_code\_task1.py

def add\_numbers(a, b)

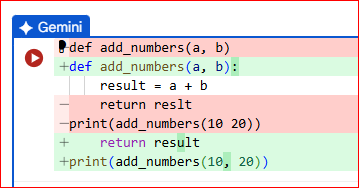
    result = a + b

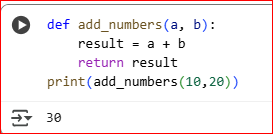
    return reslt

print(add\_numbers(10 20))

**Expected Output**:

* Corrected code with proper syntax (: after function, fixed variable name, corrected function call).
* AI should explain what was fixed.

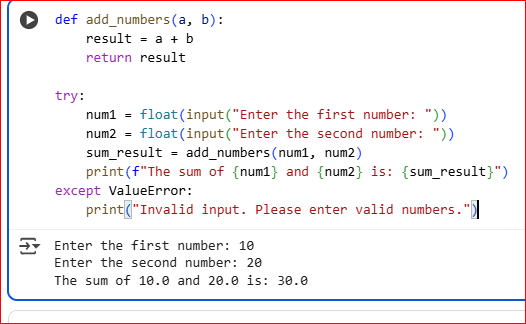




The original code had a few errors:

1. **SyntaxError: invalid syntax**: This error occurred because there was no colon (:) at the end of the def add\_numbers(a, b) line. In Python, function definitions must end with a colon.
2. **NameError: name 'reslt' is not defined**: This error happened because there was a typo in the line return reslt. The variable result was defined, but reslt was not. Python is case-sensitive and requires variable names to match exactly.
3. **SyntaxError: invalid syntax**: This error also occurred in the print statement because there was a missing comma between the two arguments add\_numbers(10, 20) and 'Result:'.

**MODIFIED CODE:**



**Task 2: Logical and Performance Issue Review**

**Task**: Optimize inefficient logic while keeping the result correct.

# buggy\_code\_task2.py

def find\_duplicates(nums):

    duplicates = []

    for i in range(len(nums)):

        for j in range(len(nums)):

            if i != j and nums[i] == nums[j] and nums[i] not in duplicates:

                duplicates.append(nums[i])

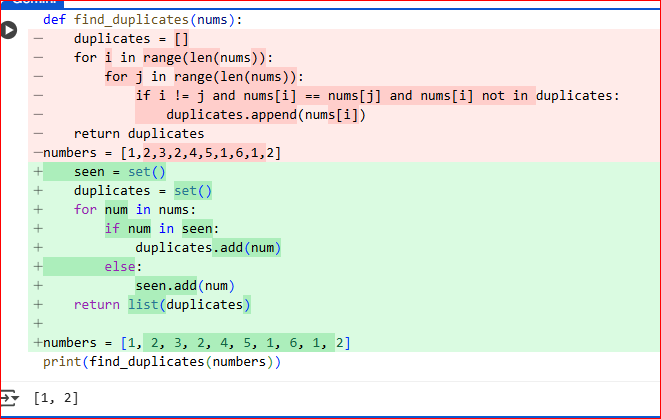
    return duplicates

numbers = [1,2,3,2,4,5,1,6,1,2]

print(find\_duplicates(numbers))

**Expected Output**:

* More efficient duplicate detection (e.g., using sets).
* AI should explain the optimization.

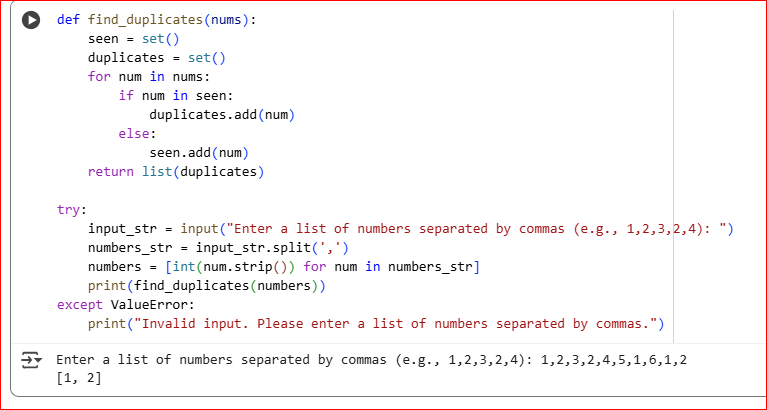


It modified the find\_duplicates function Here's how it works:

* It initialized two empty sets: seen and duplicates.
* It iterates through the input list nums.
* For each num in the list, It checks if it's already in the seen set.
  + If it is in seen, it means we've encountered this number before, so it's a duplicate. It adds it to the duplicates set.
  + If it's not in seen, it's the first time we're seeing this number, so It adds it to the seen set.
* Finally, It converts the duplicates set back to a list and return it.

Using sets for seen allows for very fast lookups (on average), making this approach more efficient for larger lists compared to nested loops or other methods that involve repeated scanning of the list.

MODIFIED CODE:



**Task 3: Code Refactoring for Readability**

**Task**: Refactor messy code into clean, PEP 8–compliant, well-structured code.

# buggy\_code\_task3.py

def c(n):

 x=1

 for i in range(1,n+1):

  x=x\*i

 return x

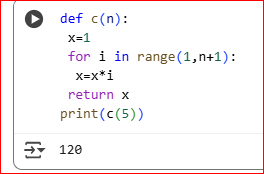
print(c(5))

Expected Output:

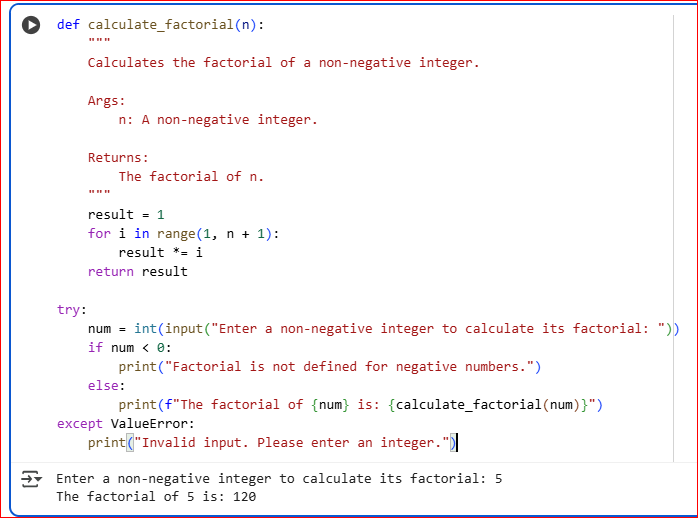
Function renamed to calculate\_factorial.

Proper indentation, variable naming, docstrings, and formatting.

AI should provide a more readable version.



MODIFIED CODE:



**Task 4: Security and Error Handling Enhancement**

**Task:** Add security practices and exception handling to the code.

# buggy\_code\_task4.py

import sqlite3

def get\_user\_data(user\_id):

    conn = sqlite3.connect("users.db")

    cursor = conn.cursor()

    query = f"SELECT \* FROM users WHERE id = {user\_id};"  # Potential SQL injection risk

    cursor.execute(query)

    result = cursor.fetchall()

    conn.close()

    return result

user\_input = input("Enter user ID: ")

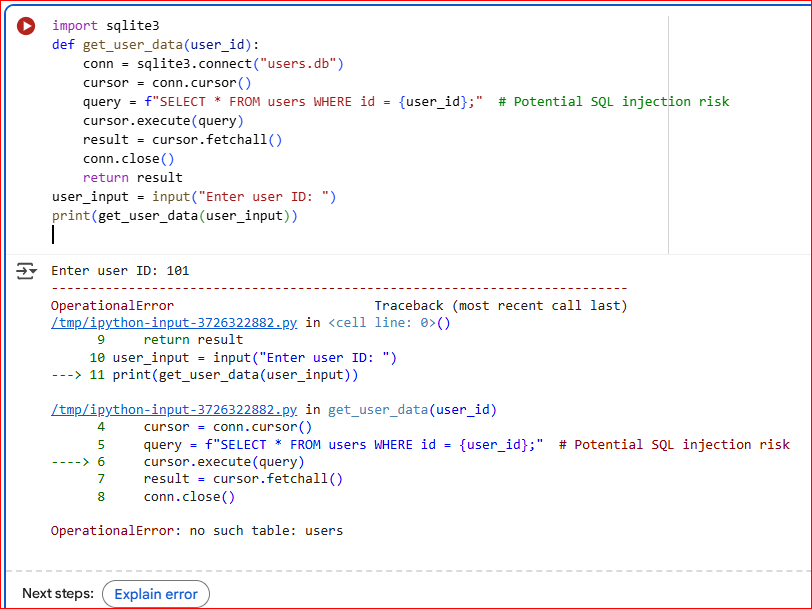
print(get\_user\_data(user\_input))

**Expected Output:**

Safe query using parameterized SQL (? placeholders).

Try-except block for database errors.

Input validation before query execution.

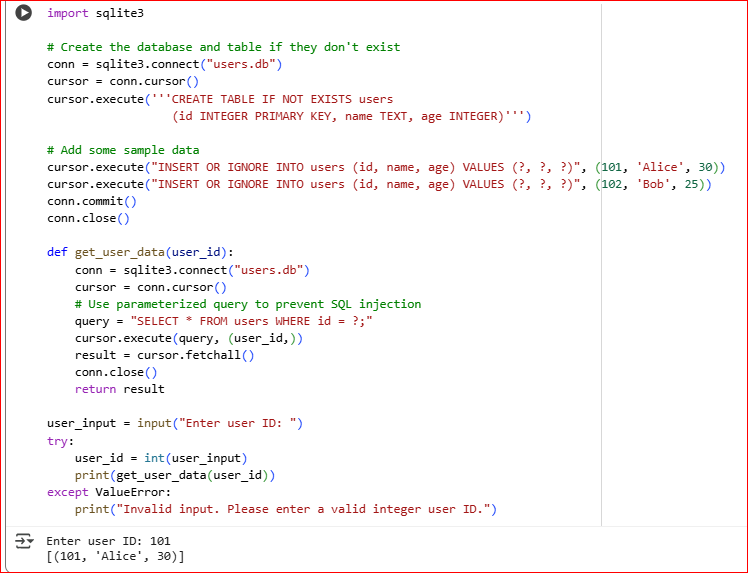
****

**CORRECT CODE:**

The error you're seeing, OperationalError: no such table: users, means that the Python script is trying to access a database table named users which doesn't exist in the users.db file.

To fix this, we need to create the database and the users table before trying to query it. I'll add code to do that. I'll also address the potential SQL injection risk in the original query by using parameterized queries, which is a safer way to pass values to SQL statements.

The error has been fixed, and the code now creates the database and table before querying it.

****

**Task 5: Automated Code Review Report Generation**

**Task**: Generate a **review report** for this messy code.

# buggy\_code\_task5.py

def calc(x,y,z):

 if z=="add":

  return x+y

 elif z=="sub": return x-y

 elif z=="mul":

  return x\*y

 elif z=="div":

  return x/y

 else: print("wrong")

print(calc(10,5,"add"))

print(calc(10,0,"div"))

**Expected Output**:

AI-generated **review report** should mention:

* Missing docstrings
* Inconsistent formatting (indentation, inline return)
* Missing error handling for division by zero
* Non-descriptive function/variable names
* Suggestions for readability and PEP 8 compliance



This code defines a function called calc that takes three arguments: x, y, and z. It performs a calculation based on the value of z:

* If z is "add", it returns the sum of x and y.
* If z is "sub", it returns the difference between x and y.
* If z is "mul", it returns the product of x and y.
* If z is "div", it returns the result of dividing x by y.
* If z is none of the above, it prints "wrong".

The code then calls the calc function twice. The first call print(calc(10, 5, "add")) will correctly return and print 15.

The second call print(calc(10, 0, "div")) is causing the error. You are trying to divide by zero, which is mathematically undefined and results in a ZeroDivisionError in Python.

The error in the code is a ZeroDivisionError. This occurs on the line return x / y when the value of y is 0, as division by zero is not allowed in Python (or mathematics). In your code, this happens when you call print(calc(10, 0, "div")).

To make this code more robust, you should add error handling (e.g., a try-except block) to catch the ZeroDivisionError when performing division and inform the user that division by zero is not allowed.

**CORRECT CODE :**

