ASSIGNMENT:10.3

HTNO:2403A51284

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

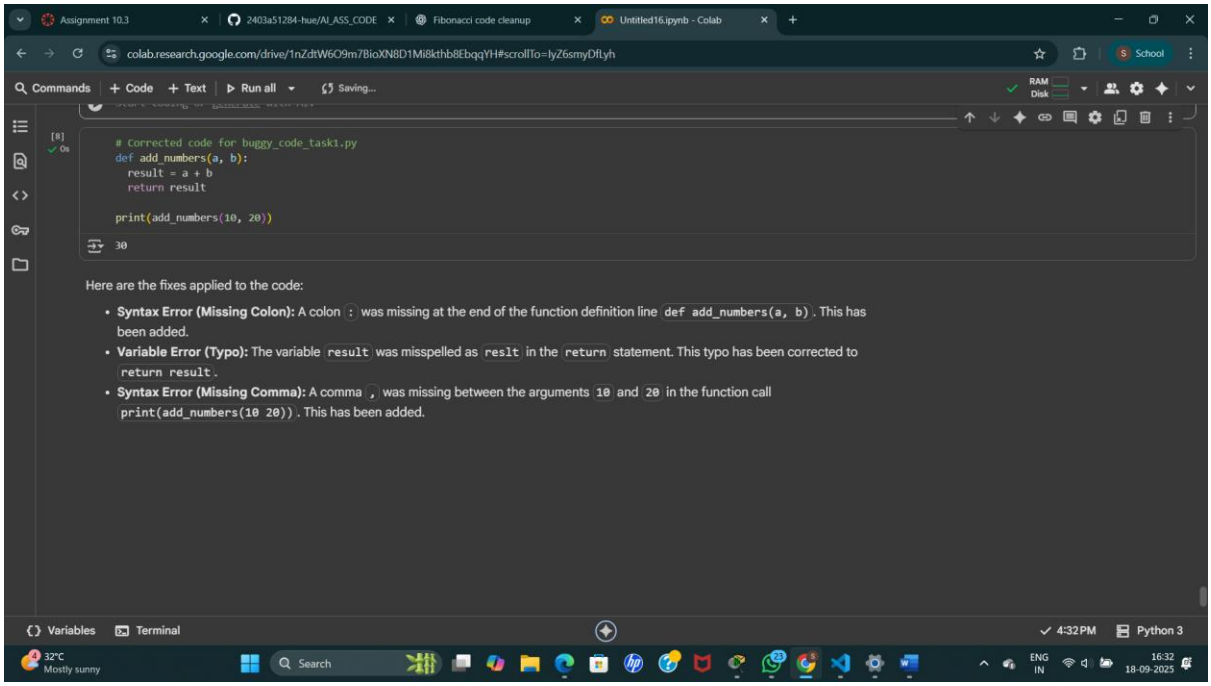
PROMPT:

1 Identify the syntax, indentation, and variable errors.

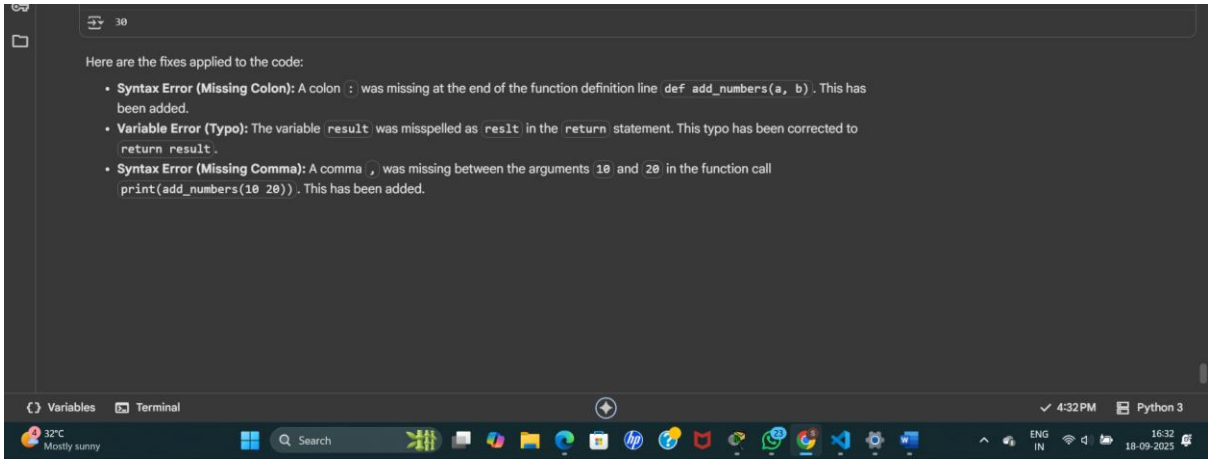
2 Correct the code so it runs properly.

3 Explain what was fixed (missing symbols, wrong variable names, incorrect function call, etc.).

CODE:



OUTPUT:



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

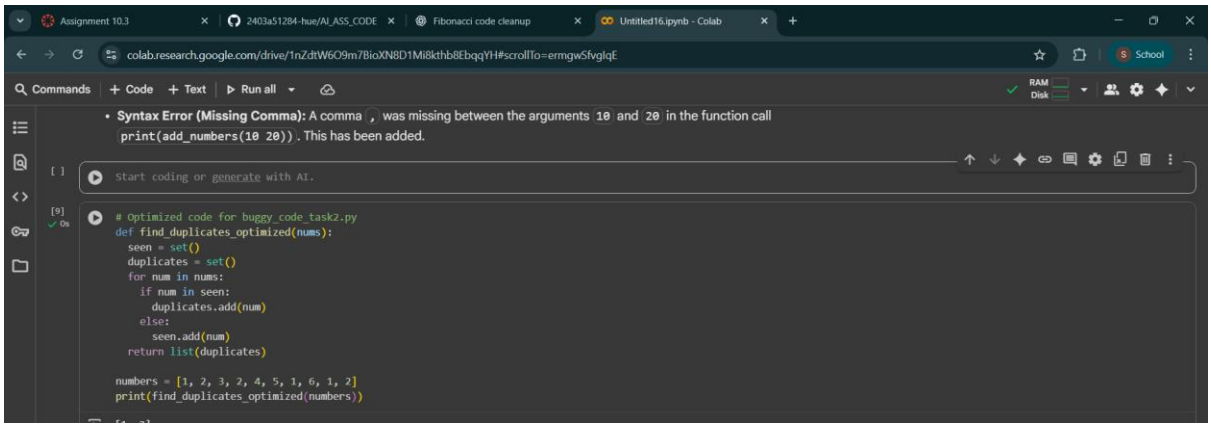
PROMPT:

1 Review the code and identify the logical and performance issues.

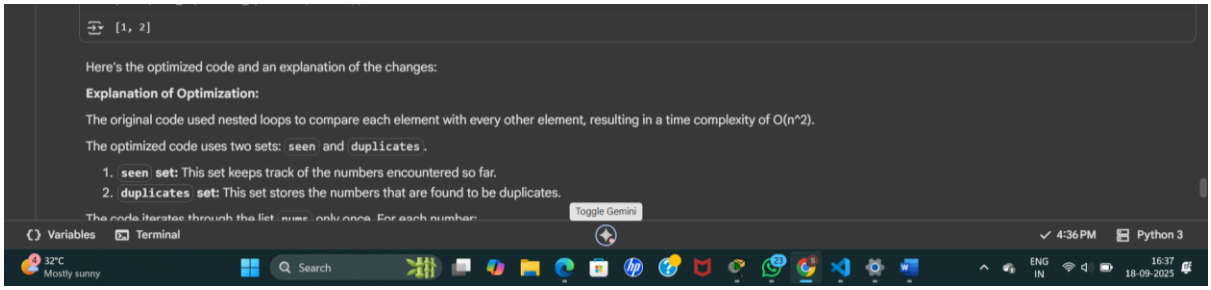
2 Optimize the code so it still produces the correct result but runs more efficiently (e.g., use sets instead of nested loops).

3 Explain the improvements made.

CODE:



OUTPUT:



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

PROMPT:

1 Refactor it into clean, PEP 8–compliant code.

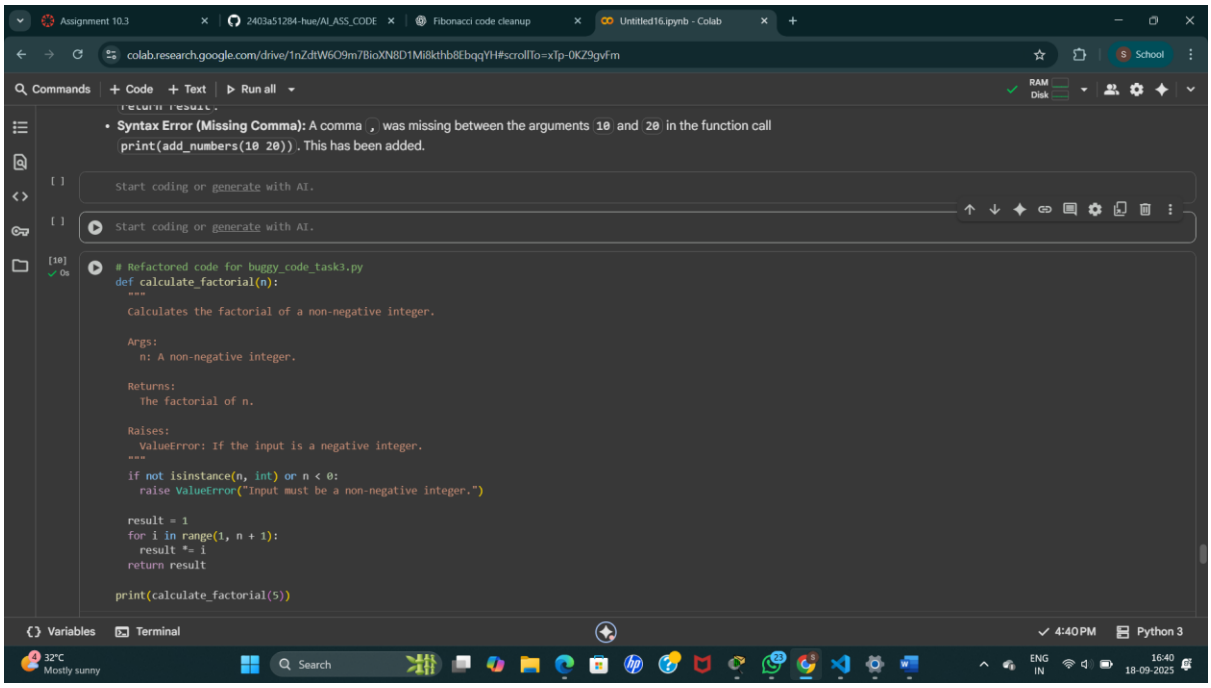
2 Rename the function c to calculate_factorial.

3 Improve variable names.

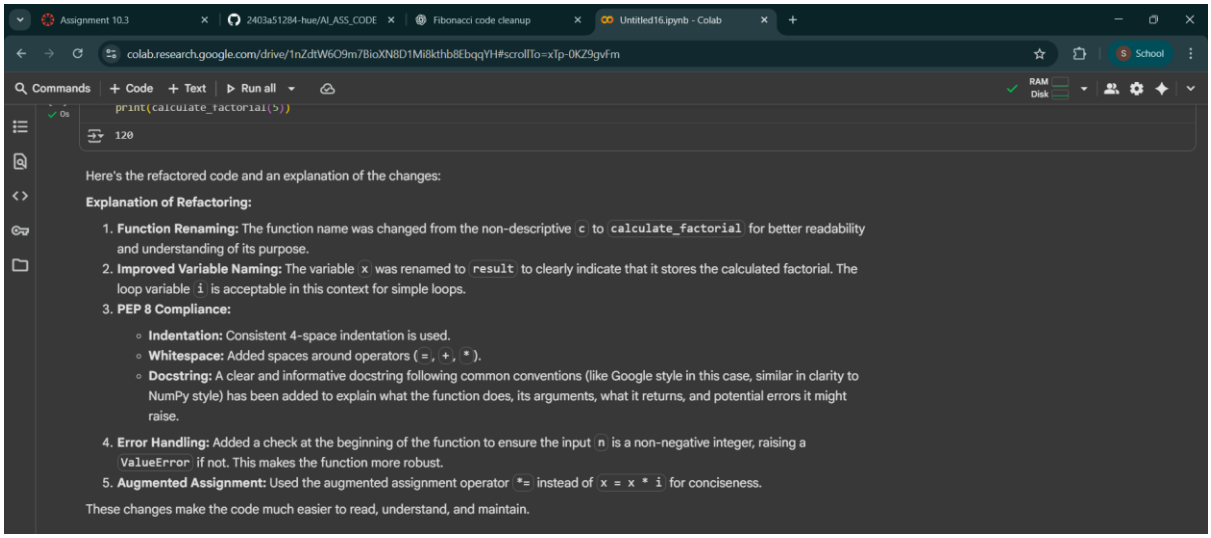
4 Add a proper docstring for the function.

5 Return a more readable, well-structured version of the code.

CODE:



OUTPUT:



Task 4: Security and Error Handling Enhancement
Task: Add security practices and exception handling to the code.

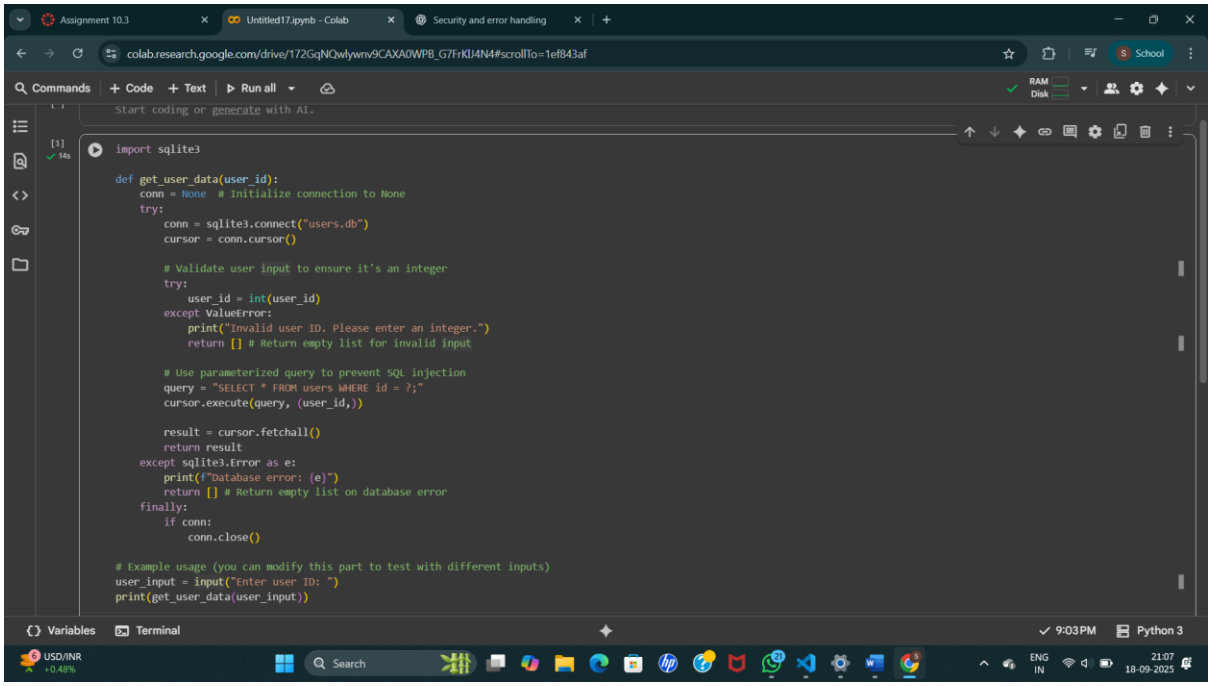
```
# buggy_code_task4.py
import sqlite3
def get_user_data(user_id):
conn = sqlite3.connect("users.db")
cursor = conn.cursor()
query = f"SELECT * FROM users WHERE id = {user_id};" #
Potential SQL injection risk
cursor.execute(query)
result = cursor.fetchall()
conn.close()
return result
user_input = input("Enter 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

PROMPT:

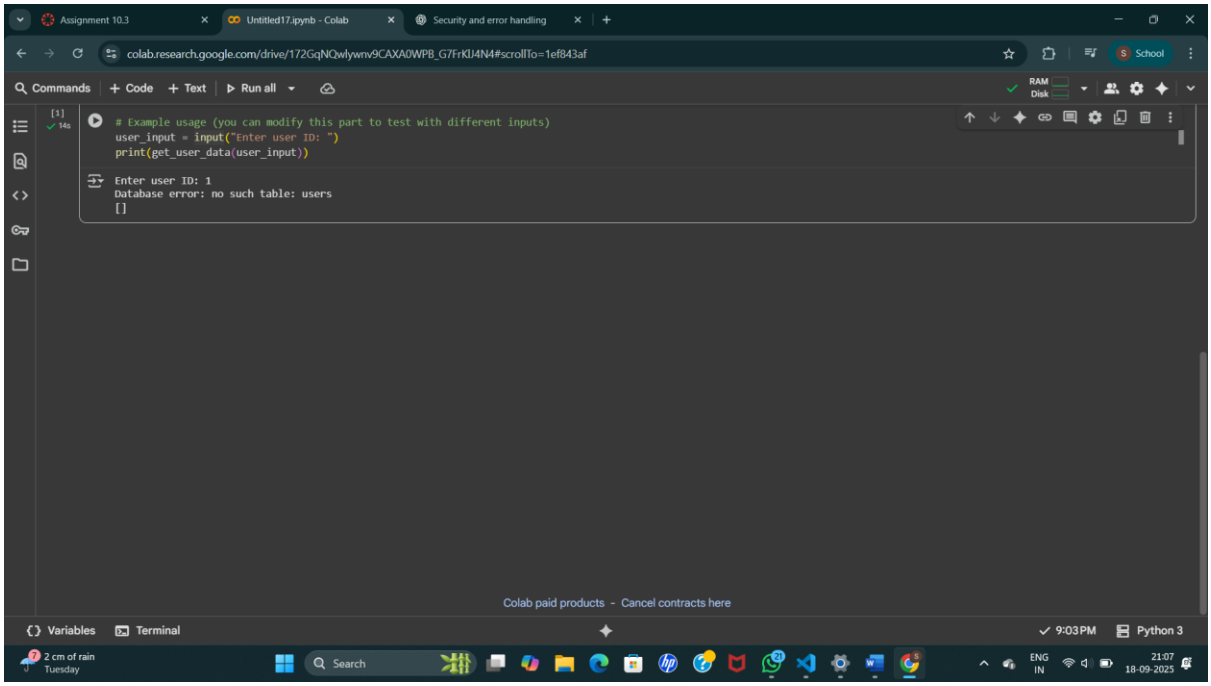
Since the error indicates that the users table does not exist, the next logical step is to create it.

CODE:



```python
import sqlite3

def get_user_data(user_id):
    conn = None  # Initialize connection to None
    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()

        # Validate user input to ensure it's an integer
        try:
            user_id = int(user_id)
        except ValueError:
            print("Invalid user ID. Please enter an integer.")
            return [] # Return empty list for invalid input

        # Use parameterized query to prevent SQL injection
        query = "SELECT * FROM users WHERE id = ?;"
        cursor.execute(query, (user_id,))

        result = cursor.fetchall()
        return result
    except sqlite3.Error as e:
        print(f"Database error: {e}")
        return [] # Return empty list on database error
    finally:
        if conn:
            conn.close()

# Example usage (you can modify this part to test with different inputs)
user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```

OUTPUT:



```python
# Example usage (you can modify this part to test with different inputs)
user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```

```
Enter user ID: 1
Database error: no such table: users
[]
```

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":

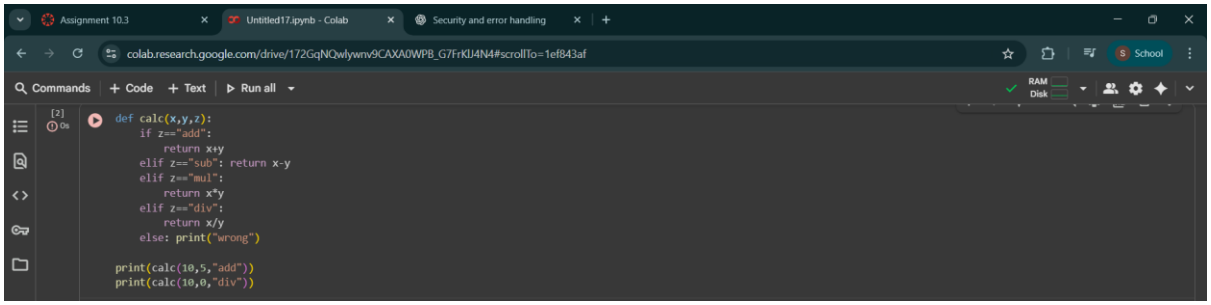
return x/y

else: print("wrong")
print(calc(10,5,"add"))
print(calc(10,0,"div"))
Expected Output:
AI-generated review report should mention:
o Missing docstrings
o Inconsistent formatting (indentation, inline return)
o Missing error handling for division by zero
o Non-descriptive function/variable names
o Suggestions for readability and PEP 8 compliance

PROMPT:

As mentioned in the code review report, the code currently has a ZeroDivisionError when attempting to divide by zero.

CODE:



OUTPUT: