

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))
```

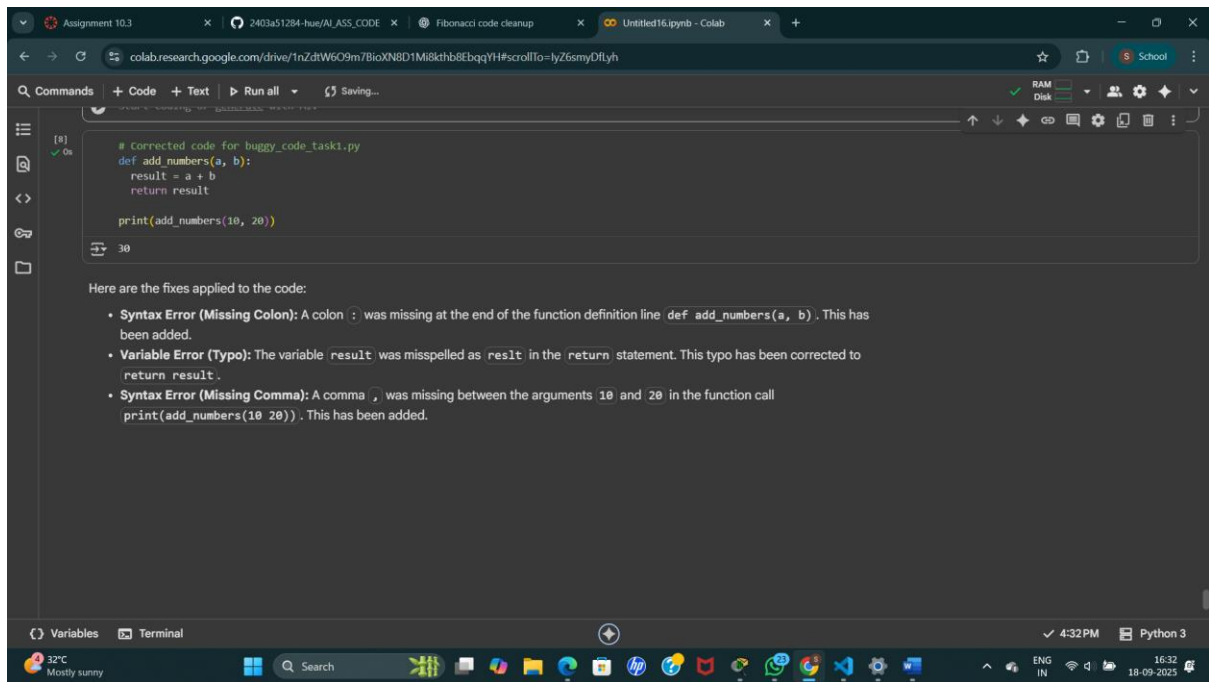
Expected Output:

- 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:



The screenshot shows a Google Colab notebook interface. The top bar includes tabs for 'Assignment 10.3', '2403a51284-hue/AI_ASS_CODE', 'Fibonacci code cleanup', and 'Untitled16.ipynb - Colab'. The address bar shows the Colab URL. The notebook has a 'Commands' bar with '+ Code', '+ Text', 'Run all', and 'Saving...'. The code editor shows a Python function `def add_numbers(a, b):` with a comment '# Corrected code for buggy_code_task1.py'. The function body is `result = a + b`, `return result`, and `print(add_numbers(10, 20))`. The output cell shows the number 30. Below the code, a message states 'Here are the fixes applied to the code:' followed by three bullet points: 'Syntax Error (Missing Colon): A colon : was missing at the end of the function definition line def add_numbers(a, b) . This has been added.', 'Variable Error (Typo): The variable result was misspelled as reslt in the return statement. This typo has been corrected to return result.', and 'Syntax Error (Missing Comma): A comma , was missing between the arguments 10 and 20 in the function call print(add_numbers(10 20)) . This has been added.'

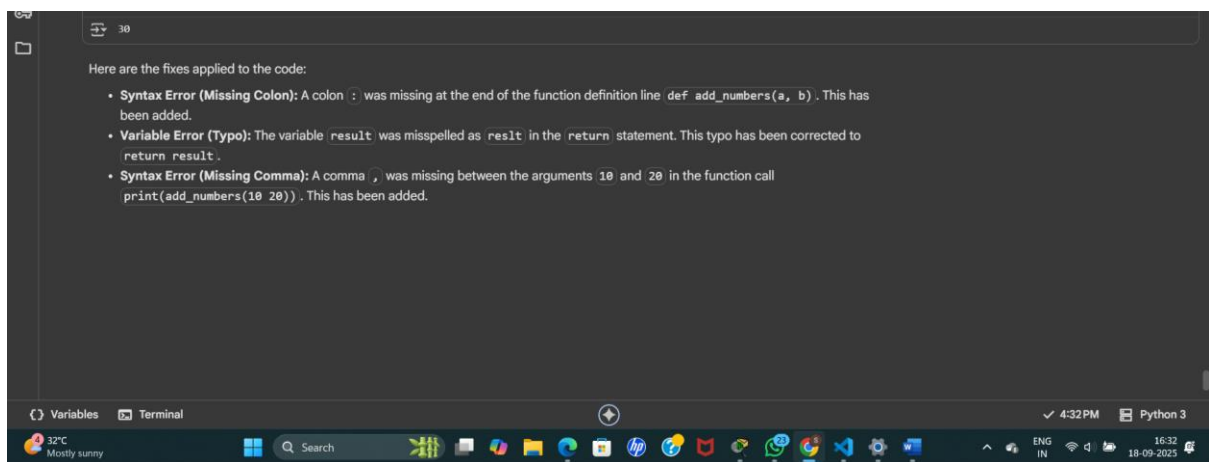
```
# Corrected code for buggy_code_task1.py
def add_numbers(a, b):
    result = a + b
    return result

print(add_numbers(10, 20))
```

Here are the fixes applied to the code:

- **Syntax Error (Missing Colon):** A colon `:` was missing at the end of the function definition line `def add_numbers(a, b)` . This has been added.
- **Variable Error (Typo):** The variable `result` was misspelled as `reslt` in the `return` statement. This typo has been corrected to `return result` .
- **Syntax Error (Missing Comma):** A comma `,` was missing between the arguments `10` and `20` in the function call `print(add_numbers(10 20))` . This has been added.

OUTPUT:



This screenshot is identical to the one above, showing the same Google Colab notebook with the corrected Python function and its output of 30. It includes the same tabs, address bar, and error correction message.

```
# Corrected code for buggy_code_task1.py
def add_numbers(a, b):
    result = a + b
    return result

print(add_numbers(10, 20))
```

Here are the fixes applied to the code:

- **Syntax Error (Missing Colon):** A colon `:` was missing at the end of the function definition line `def add_numbers(a, b)` . This has been added.
- **Variable Error (Typo):** The variable `result` was misspelled as `reslt` in the `return` statement. This typo has been corrected to `return result` .
- **Syntax Error (Missing Comma):** A comma `,` was missing between the arguments `10` and `20` in the function call `print(add_numbers(10 20))` . This has been added.

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))
```

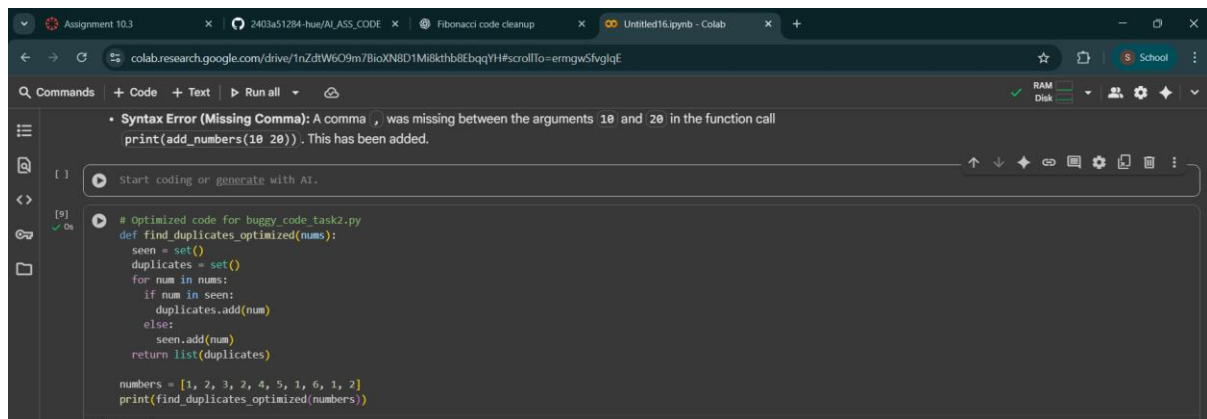
Expected Output:

- 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:



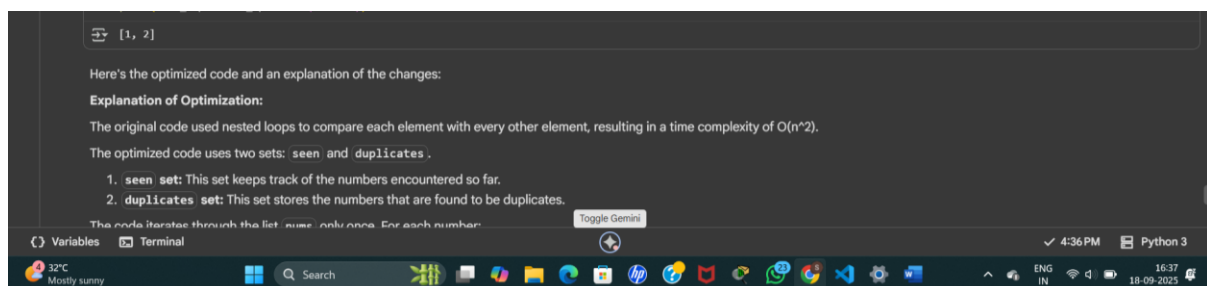
The screenshot shows a Google Colab notebook interface. At the top, there's a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all'. Below that, a message states: 'Syntax Error (Missing Comma): A comma , was missing between the arguments 10 and 20 in the function call print(add_numbers(10 20)). This has been added.' The main code cell contains the following Python code:

```
[ ] start coding or generate with AI.

# Optimized code for buggy_code_task2.py
def find_duplicates_optimized(nums):
    seen = set()
    duplicates = set()
    for num in nums:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return list(duplicates)

numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
print(find_duplicates_optimized(numbers))
```

OUTPUT:



The screenshot shows a terminal window with the output of the code execution. The output is:

```
[1, 2]
```

Below the output, there's an explanation of the changes:

Here's the optimized code and an explanation of the changes:

Explanation of Optimization:

The original code used nested loops to compare each element with every other element, resulting in a time complexity of $O(n^2)$.

The optimized code uses two sets: `seen` and `duplicates`.

1. `seen` set: This set keeps track of the numbers encountered so far.
2. `duplicates` set: This set stores the numbers that are found to be duplicates.

The code iterates through the list `nums` only once. For each number:

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:

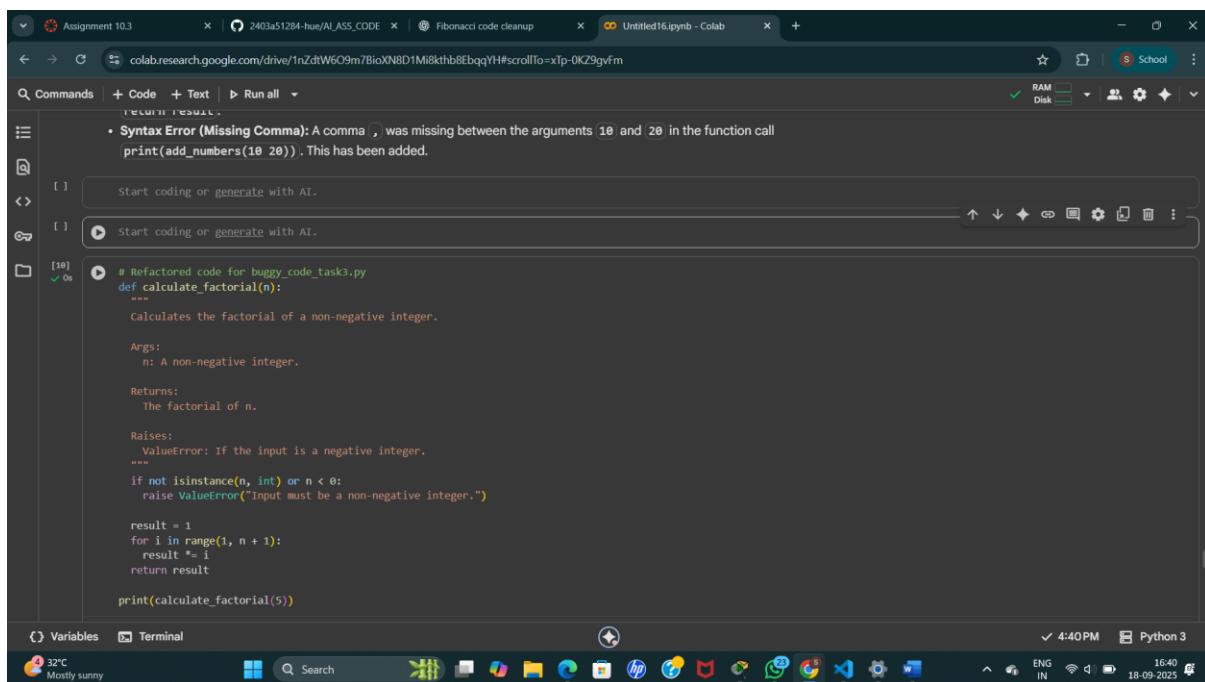
- 1 Refactor it into clean, PEP 8–compliant code.
- 2 Rename the function `c` to `calculate_factorial`.

3 Improve variable names.

4 Add a proper docstring for the function.

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

CODE:



The screenshot shows a Google Colab notebook interface. At the top, there are several tabs: 'Assignment 10.3', '2403a51284-hue/AL_ASS_CODE', 'Fibonacci code cleanup', and 'Untitled16.ipynb - Colab'. The main code cell contains a Python function named `calculate_factorial` with a docstring and a test call. The function calculates the factorial of a non-negative integer `n` and returns the result. The test call `print(calculate_factorial(5))` is at the bottom of the cell. The output of the cell is not visible in this screenshot.

```
# Refactored code for buggy_code_task3.py
def calculate_factorial(n):
    """
    Calculates the factorial of a non-negative integer.

    Args:
        n: A non-negative integer.

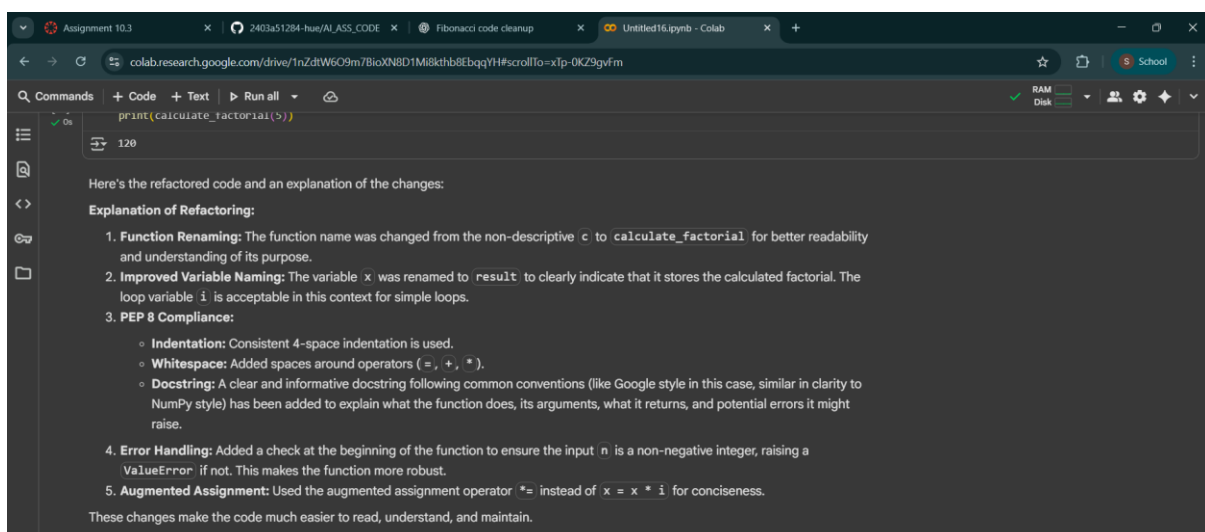
    Returns:
        The factorial of n.

    Raises:
        ValueError: If the input is a negative integer.
    """
    if not isinstance(n, int) or n < 0:
        raise ValueError("Input must be a non-negative integer.")

    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

print(calculate_factorial(5))
```

OUTPUT:



The screenshot shows the same Google Colab notebook interface. The main code cell now contains the output of the `print(calculate_factorial(5))` statement, which is the number 120. Below the code cell, there is a text block explaining the refactoring changes.

```
print(calculate_factorial(5))
```

120

Here's the refactored code and an explanation of the changes:

Explanation of Refactoring:

- Function Renaming:** The function name was changed from the non-descriptive `c` to `calculate_factorial` for better readability and understanding of its purpose.
- Improved Variable Naming:** The variable `x` was renamed to `result` to clearly indicate that it stores the calculated factorial. The loop variable `i` is acceptable in this context for simple loops.
- PEP 8 Compliance:**
 - Indentation:** Consistent 4-space indentation is used.
 - Whitespace:** Added spaces around operators (`=`, `+`, `*`).
 - Docstring:** A clear and informative docstring following common conventions (like Google style in this case, similar in clarity to NumPy style) has been added to explain what the function does, its arguments, what it returns, and potential errors it might raise.
- Error Handling:** Added a check at the beginning of the function to ensure the input `n` is a non-negative integer, raising a `ValueError` if not. This makes the function more robust.
- Augmented Assignment:** Used the augmented assignment operator `*=` instead of `x = x * i` for conciseness.

These changes make the code much easier to read, understand, and maintain.

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).

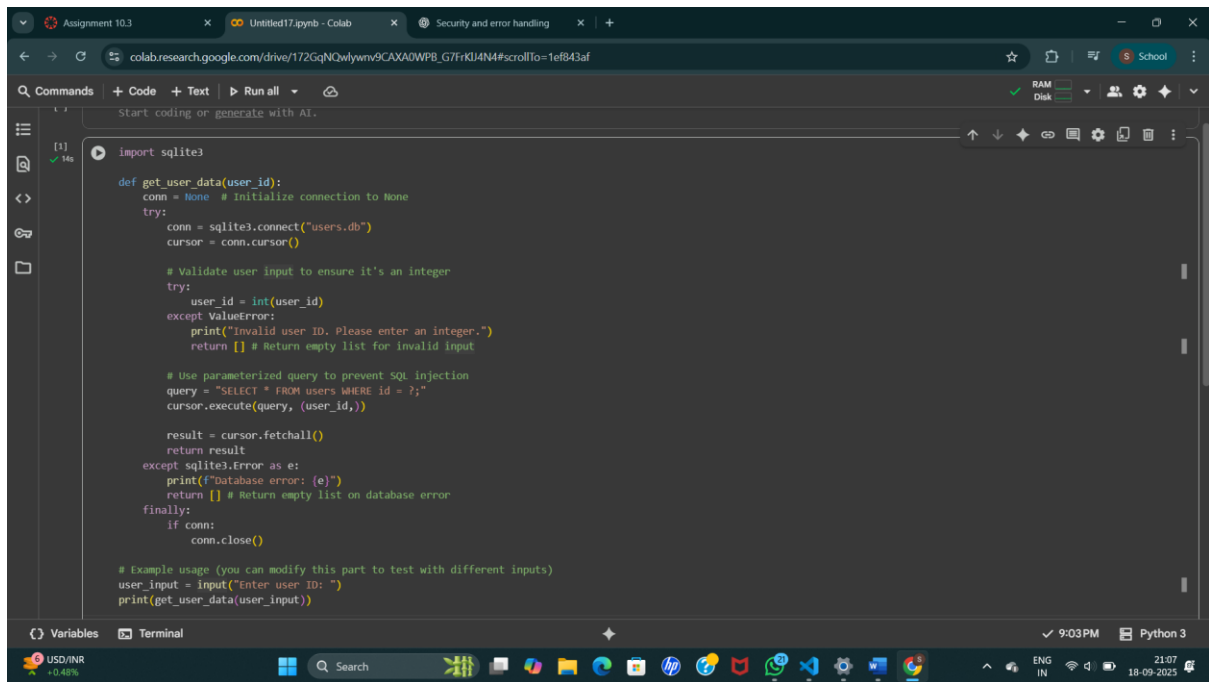
Try-except block for database errors.

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:



The image shows a Google Colab notebook interface. The top bar includes tabs for 'Assignment 10.3', 'Untitled17.ipynb - Colab', and 'Security and error handling'. The address bar shows a URL from colab.research.google.com. The left sidebar contains icons for file management and a 'Commands' search bar. The main code editor displays a Python script that uses the sqlite3 module to connect to a database named 'users.db'. The script defines a function 'get_user_data(user_id)' that validates the input, executes a SQL query, and returns the results. It includes error handling for 'ValueError' and 'sqlite3.Error'. At the bottom, there is an example usage section that prompts the user for an ID and prints the result. The bottom status bar shows '9:03 PM', 'Python 3', and system icons for USD/INR, search, and network status.

```
import sqlite3

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

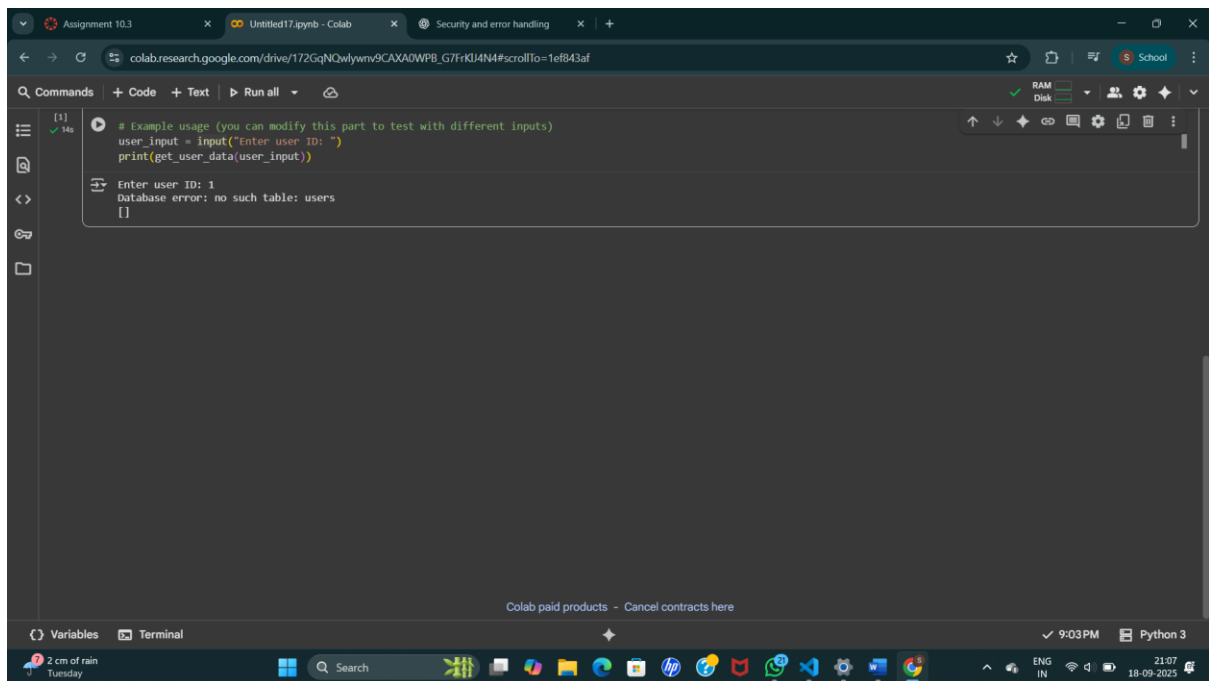
        # Validate user input to ensure it's an integer
        try:
            user_id = int(user_id)
        except ValueError:
            print("Invalid user ID. Please enter an integer.")
            return [] # Return empty list for invalid input

        # 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:



The image shows the same Google Colab notebook interface, but now displaying the output of the code. The code editor shows the example usage section. The output area below the code shows the prompt 'Enter user ID: 1' and the resulting 'Database error: no such table: users' followed by an empty list '[]'. The bottom status bar shows '9:03 PM', 'Python 3', and system icons for weather (2 cm of rain), search, and network status.

```
# 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:

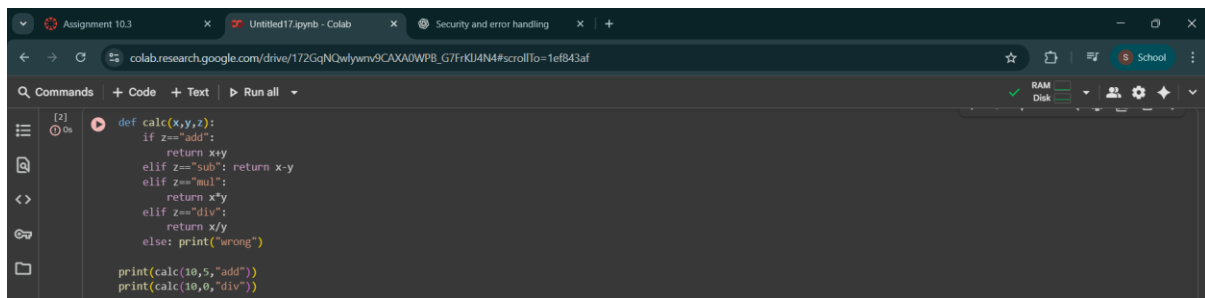
AI-generated review report should mention:

- 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:

A screenshot of a Google Colab notebook interface. The browser address bar shows a URL from colab.research.google.com. The notebook has a dark theme. On the left, there's a sidebar with icons for file explorer, search, and other tools. The main area shows a Python code cell with the following code:

```
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"))
```

OUTPUT:

