

# AI ASSISTED CODING

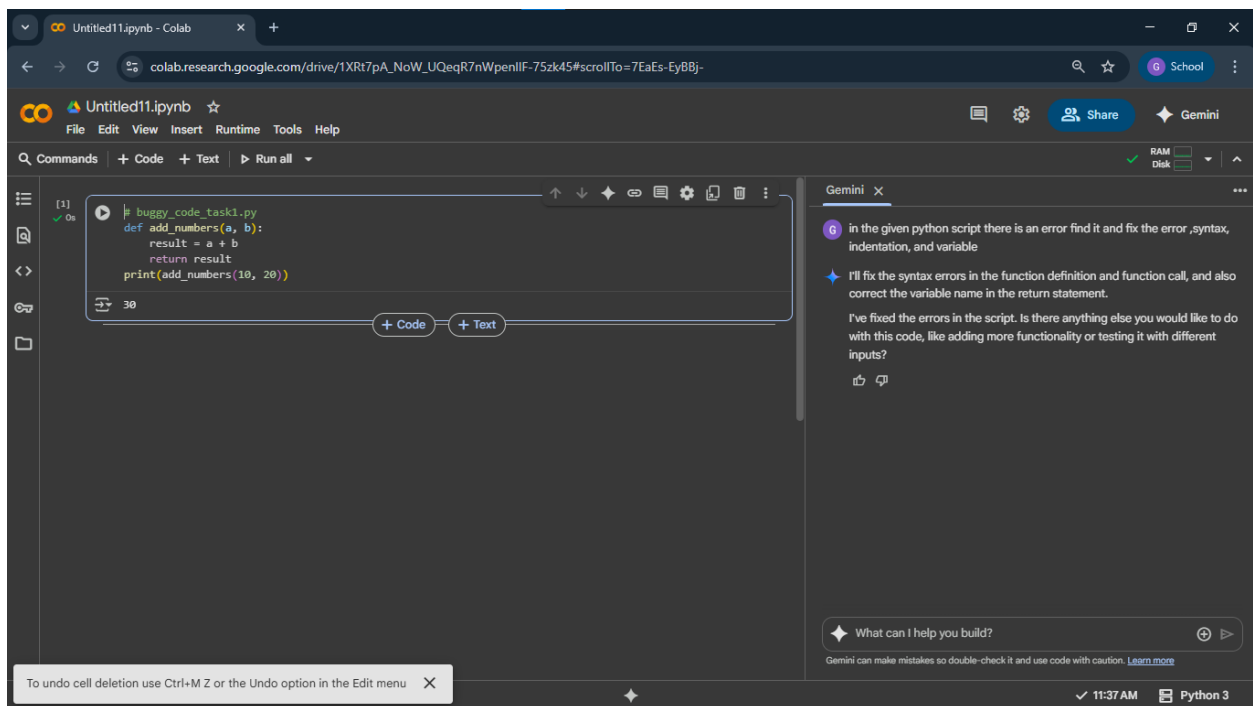
Name:- Kartheek

Rollno:- 2403A53003

Assignment:- 10.4

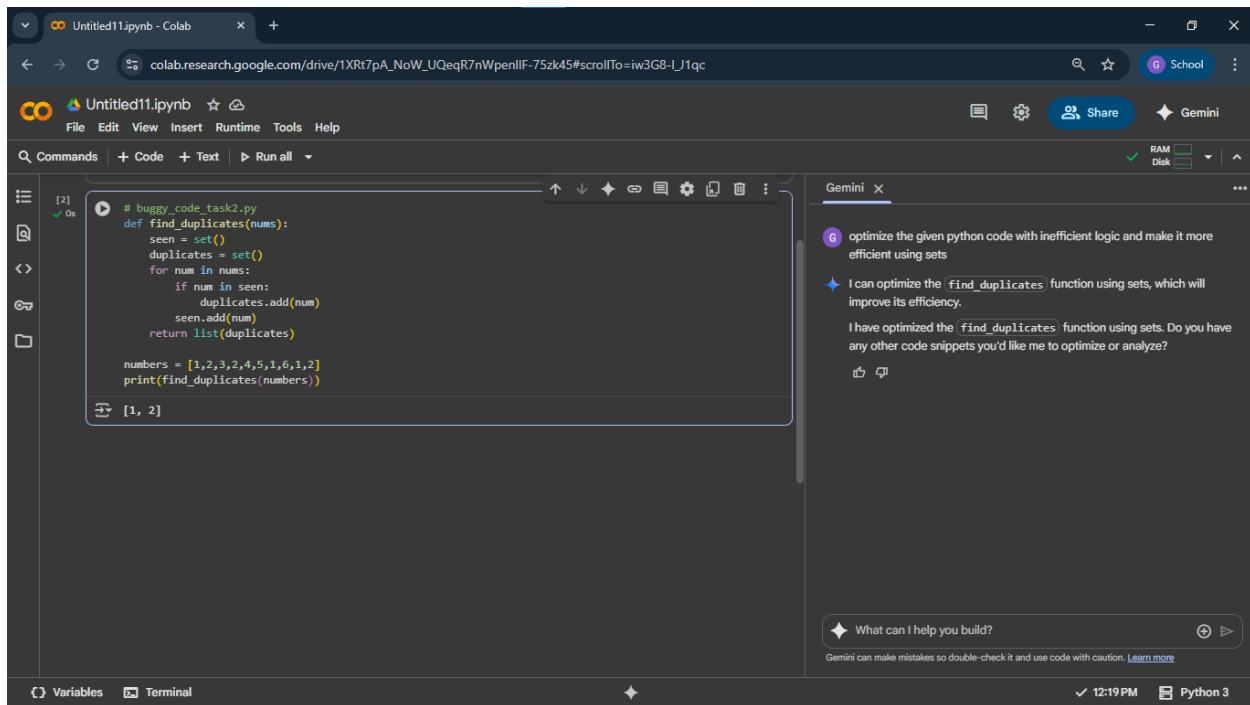
## Task 1: Syntax and Error Detection

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



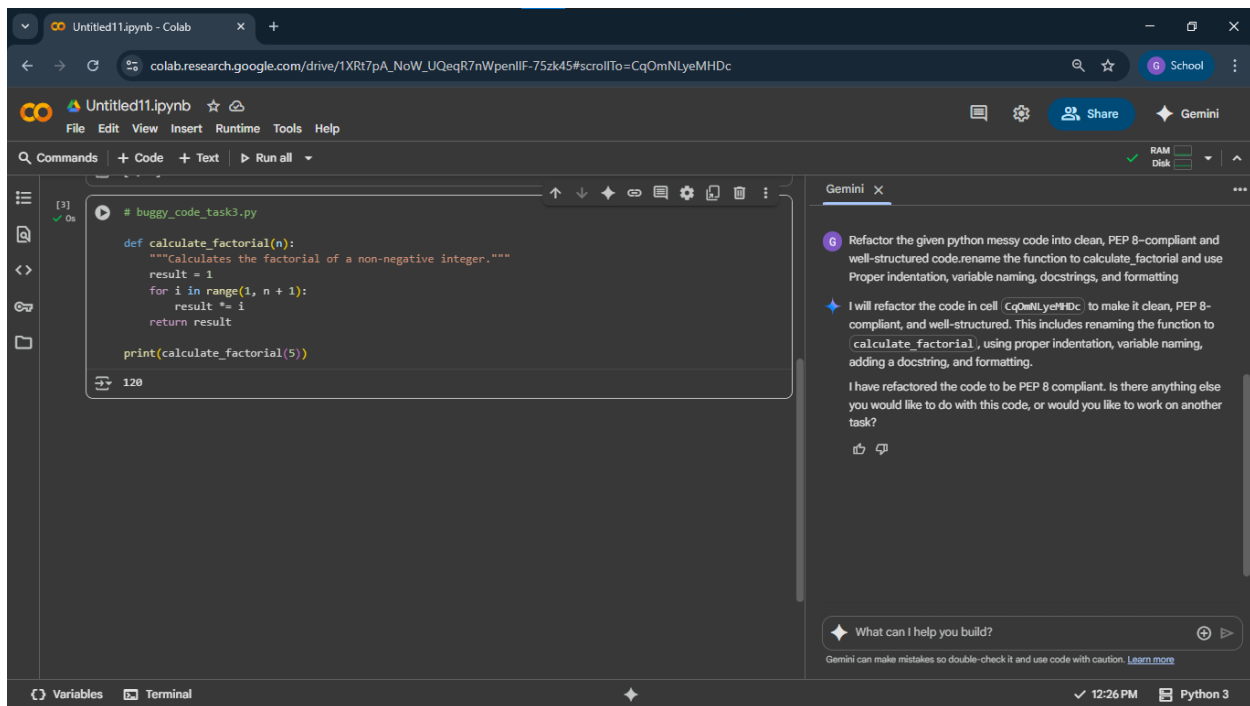
## Task 2: Logical and Performance Issue Review

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



### Task 3: Code Refactoring for Readability

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



### Task 4: Security and Error Handling Enhancement

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

Untitled11.ipynb - Colab

colab.research.google.com/drive/1XRt7pA\_NoW\_UQeqR7nWpenlIF-75zk45#scrollTo=fMHdFk0QO1Vt

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```

except sqlite3.Error as e:
    print(f"Database error: {e}")
    return None
finally:
    if conn:
        conn.close()

# Example usage:
# Assuming 'users' table exists with an 'id' column
# You might need to create the table and add some data for testing
# Example of creating table and adding data:
conn = sqlite3.connect("users.db")
cursor = conn.cursor()
cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)")
cursor.execute("INSERT INTO users (id, name) VALUES (?, ?)", (1, "Alice"))
cursor.execute("INSERT INTO users (id, name) VALUES (?, ?)", (2, "Bob"))
conn.commit()
conn.close()

user_input = input("Enter user ID: ")
print(get_user_data(user_input))

```

Enter user ID: 3003  
Database error: no such table: users  
None

Gemini

compliant, and well-structured. This includes renaming the function to `calculate_factorial`, using proper indentation, variable naming, adding a docstring, and formatting.

I have refactored the code to be PEP 8 compliant. Is there anything else you would like to do with this code, or would you like to work on another task?

6 Add security practices and exception handling using parameterized SQL and Try-except block for database errors. Input validation before query execution. to the given python code

I will add security practices and exception handling to the database code in cell `fMHdFk0QO1Vt`. This includes input validation, using parameterized SQL to prevent injection, and adding a `try-except` block for database errors.

I've added security practices and exception handling to the code, including input validation and parameterized SQL. Would you like to test this code with some sample data, or is there anything else I can help you with?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal 12:38 PM Python 3

Untitled11.ipynb - Colab

colab.research.google.com/drive/1XRt7pA\_NoW\_UQeqR7nWpenlIF-75zk45#scrollTo=fMHdFk0QO1Vt

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```

# buggy_code_task4.py
import sqlite3

def get_user_data(user_id):
    """
    Retrieves user data from the database using a user ID.

    Args:
        user_id: The ID of the user to retrieve data for.

    Returns:
        A list of tuples containing the user data, or None if an error occurs.
    """
    conn = None # Initialize connection to None
    try:
        # Input validation: Ensure user_id is an integer
        user_id = int(user_id)
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()
        # Use parameterized query to prevent SQL injection
        query = "SELECT * FROM users WHERE id = ?;"
        cursor.execute(query, (user_id,))
        result = cursor.fetchall()
        return result
    except ValueError:
        print("Error: Invalid user ID. Please enter an integer.")
        return None
    except sqlite3.Error as e:
        print(f"Database error: {e}")
        return None
    finally:
        if conn:
            conn.close()

```

Gemini

compliant, and well-structured. This includes renaming the function to `calculate_factorial`, using proper indentation, variable naming, adding a docstring, and formatting.

I have refactored the code to be PEP 8 compliant. Is there anything else you would like to do with this code, or would you like to work on another task?

6 Add security practices and exception handling using parameterized SQL and Try-except block for database errors. Input validation before query execution. to the given python code

I will add security practices and exception handling to the database code in cell `fMHdFk0QO1Vt`. This includes input validation, using parameterized SQL to prevent injection, and adding a `try-except` block for database errors.

I've added security practices and exception handling to the code, including input validation and parameterized SQL. Would you like to test this code with some sample data, or is there anything else I can help you with?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal 12:38 PM Python 3

## Task 5: Automated Code Review Report Generation

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

Untitled11.ipynb - Colab

colab.research.google.com/drive/1XRt7pA\_NoW\_UQeqR7nWpenlIF-75zk45#scrollTo=tSkkGmDKQ2ya

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

Code Review Report for `buggy_code_task5.py`

Here is a review of the provided Python code, focusing on the requested areas:

Issues Identified:

1. Missing Docstrings: The function `calc` lacks a docstring explaining its purpose, arguments, and what it returns.

2. Inconsistent Formatting:

o Indentation: The indentation is inconsistent, particularly in the `elif` and `else` blocks. PEP 8 recommends consistent 4-space indentation.

o Inline Return: The `return x-y` in the first `elif` is on the same line as the condition, which is less readable than placing the return statement on the next line with proper indentation.

3. Missing Error Handling for Division by Zero: The code does not handle the case where the `div` operation is performed with `y` being 0, which will result in a `ZeroDivisionError`.

4. Non-descriptive Function/Variable Names:

o The function name `calc` is very generic. A more descriptive name like `perform_operation` or `calculate_result` would better convey its purpose.

o The variable names `x`, `y`, and `z` are not descriptive. They should be renamed to reflect what they represent (e.g., `num1`, `num2`, `operation`).

Suggestions for Readability and PEP 8 Compliance:

1. Add a Docstring: Include a clear and concise docstring for the `calc` function.

2. Consistent Indentation and Formatting: Apply consistent 4-space indentation throughout the function.

Gemini

6 Generate a review report for the given python messy code and 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

I will generate a review report for the code in cell `tSkkGmDKQ2ya`, highlighting the missing docstrings, inconsistent formatting, missing error handling for division by zero, non-descriptive names, and provide suggestions for readability and PEP 8 compliance.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal 12:38 PM Python 3

Untitled11.ipynb - Colab

colab.research.google.com/drive/1XRt7pA\_NoW\_UQeqR7nWpenlIF-75zk45#scrollTo=tSkkGmDKQ2ya

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

o Indentation: The indentation is inconsistent, particularly in the `elif` and `else` blocks. PEP 8 recommends consistent 4-space indentation.

o Inline Return: The `return x-y` in the first `elif` is on the same line as the condition, which is less readable than placing the return statement on the next line with proper indentation.

3. Missing Error Handling for Division by Zero: The code does not handle the case where the `div` operation is performed with `y` being 0, which will result in a `ZeroDivisionError`.

4. Non-descriptive Function/Variable Names:

o The function name `calc` is very generic. A more descriptive name like `perform_operation` or `calculate_result` would better convey its purpose.

o The variable names `x`, `y`, and `z` are not descriptive. They should be renamed to reflect what they represent (e.g., `num1`, `num2`, `operation`).

Suggestions for Readability and PEP 8 Compliance:

1. Add a Docstring: Include a clear and concise docstring for the `calc` function.

2. Consistent Indentation and Formatting: Apply consistent 4-space indentation throughout the function. Place return statements on a new line after the operation.

3. Implement Error Handling: Add a `try-except` block or an explicit check for division by zero when `z` is "div" to handle this potential error gracefully.

4. Use Descriptive Names: Rename the function and variables to be more descriptive and improve code readability.

Refactored Code (Incorporating suggestions):

Gemini

6 Generate a review report for the given python messy code and 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

I will generate a review report for the code in cell `tSkkGmDKQ2ya`, highlighting the missing docstrings, inconsistent formatting, missing error handling for division by zero, non-descriptive names, and provide suggestions for readability and PEP 8 compliance.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal 12:38 PM Python 3