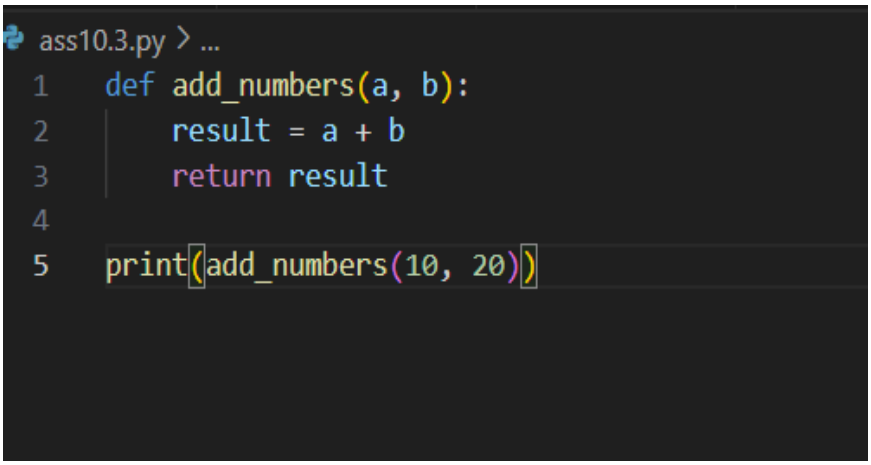
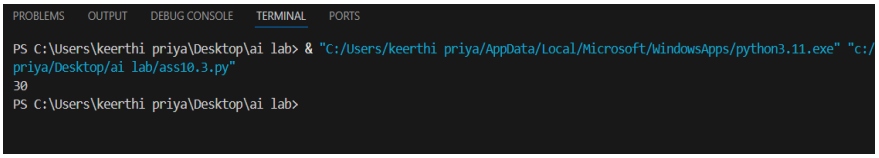
	AI ASSISTED CODING **NAME:HARINI CHANDA** **HALL TICKET NUM:2403A510E1** **ASSIGNMENT:10.3** **BATCH:05**	
1	TASK1# PROMPT: I have a Python script with syntax, indentation, and variable errors. Please identify and fix them # buggy_code_task1.py def add_numbers(a, b) result = a + b return reslt print(add_numbers(10 20)) CODE:  OUTPUT:  OBSERVATION: □ Missing Colon in Function Definition: • Original: def add_numbers(a, b) • Issue: Python function definitions require a colon (:) at the end of the def line to indicate the start of the function's code block. • Fix: def add_numbers(a, b): □ Incorrect Indentation: • Original: The lines result = a + b and return reslt were not	

properly indented under the function definition.

- Issue: Python uses indentation to define code blocks. All statements within a function must be indented consistently.
- Fix: The lines result = a + b and return result have been indented to align correctly with the function definition.

Task 2

PROMPT:

I have a Python script that finds duplicate numbers in a list, but the logic is inefficient because it uses nested loops. Please optimize the code so that it still produces the correct result but runs more efficiently.

CODE:

```
ass10.3.py > ...
1   def find_duplicates(nums):
2       seen = set()
3       duplicates = set()
4       for num in nums:
5           if num in seen:
6               duplicates.add(num)
7           else:
8               seen.add(num)
9       return list(duplicates)
10
11  numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
12  print(find_duplicates(numbers))
13
```

OUTPUT:

```
priya/Desktop/ai lab/ass10.3.py"
[1, 2]
PS C:\Users\keerthi priya\Desktop\ai lab>
```

OBSERVATION:

 The original code used two nested loops ($O(n^2)$ time complexity) to compare every element with every other element.

 In the optimized code, we use two sets (seen and duplicates) to track numbers efficiently.

- seen keeps track of elements we've already encountered.
- If a number is already in seen, it gets added to duplicates.

 This reduces the time complexity to O(n) and makes the solution much faster for large input lists.

 The output remains the same:

Task 3

PROMPT:

I have a Python script that calculates the factorial of a number, but the code is messy and not PEP 8–compliant. Please refactor it into a clean, well-structured version with:

- Proper indentation and formatting.
- A meaningful function name (calculate_factorial).
- Clear variable naming.
- A docstring explaining the function.

CODE:

ass10.3.py > ...

```python
def calculate_factorial(n):
    """
    Calculate the factorial of a given number.

    Args:
        n (int): A non-negative integer.

    Returns:
        int: The factorial of the input number.
    """
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result


print(calculate_factorial(5))

```

OUTPUT:

```
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:
priya/Desktop/ai lab/ass10.3.py"
120
PS C:\Users\keerthi priya\Desktop\ai lab>
```

OBSERVATION:

- Function name changed from c → calculate_factorial for clarity.
- Variable x renamed to result, making the purpose more descriptive.
- PEP 8 formatting applied: proper indentation, spaces around operators, blank lines for readability.
- Docstring added to explain parameters, return type, and purpose.
- Loop logic preserved but made more readable with result *= i.

Task 4:

PROMPT:

I have a Python script that fetches user data from a SQLite database. The current code is unsafe because it uses string formatting in SQL queries, which makes it vulnerable to SQL injection. Please:

- Use parameterized queries (? placeholders) instead of string concatenation.
- Add try–except blocks to handle database errors gracefully.
- Include input validation before executing the query.
- Refactor the code to follow clean practices.

CODE:

ASS10.3.4.PY > ...

```python
import sqlite3


def get_user_data(user_id):
    """
    Fetch user data from the database by user_id.

    Args:
        user_id (int): ID of the user to look up.

    Returns:
        list: A list of rows matching the user_id, or an empty list if none found.
    """
    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()
        query = "SELECT * FROM users WHERE id = ?;"
        cursor.execute(query, (user_id,))
        result = cursor.fetchall()
    except sqlite3.Error as e:
        print(f"Database error: {e}")
        result = []
    finally:
        if conn:
            conn.close()
    return result


def main():
    user_input = input("Enter user ID: ").strip()

    if not user_input.isdigit():
        print("Invalid input. Please enter a numeric user ID.")
        return

    user_id = int(user_input)
    data = get_user_data(user_id)
```

Ln 47, Col 1 Spaces: 4 UT

ASS10.3.4.PY > ...

```python
def main():
    user_id = int(user_input)
    data = get_user_data(user_id)

    if data:
        print("User Data:", data)
    else:
        print("No user found with that ID.")


if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai
 lab/ASS10.3.4.PY"
Enter user ID: 2403A510G4
Enter user ID: 2403A510G4
Invalid input. Please enter a numeric user ID.
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai
 lab/ASS10.3.4.PY"
Enter user ID: 123456
Database error: no such table: users
No user found with that ID.
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai
No user found with that ID.
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai
 lab/ASS10.3.4.PY"
Enter user ID: 101
Database error: no such table: users
No user found with that ID.
PS C:\Users\keerthi priya\Desktop\ai lab>
```

OBSERVATION:

- Exception Handling:
  - Added try–except to catch sqlite3.Error.
  - Ensures the program doesn't crash on DB errors.
- Input Validation:
  - Checked user_input.isdigit() before converting to integer.
  - Prevents invalid input like "abc" from reaching the query.
- Resource Management:
  - Used finally to close the DB connection safely.

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

PROMPT:

I have a Python script that performs basic arithmetic operations, but it is messy and not PEP 8–compliant. Please generate a review report identifying issues such as:

- Missing docstrings.
- Inconsistent formatting and indentation.
- Inline return statements without readability.
- Missing error handling (division by zero).
- Non-descriptive function and variable names.
- Suggestions for improving readability and PEP 8 compliance.

After that, provide a refactored version of the code.

CODE:

ass10.3.5.py > ...

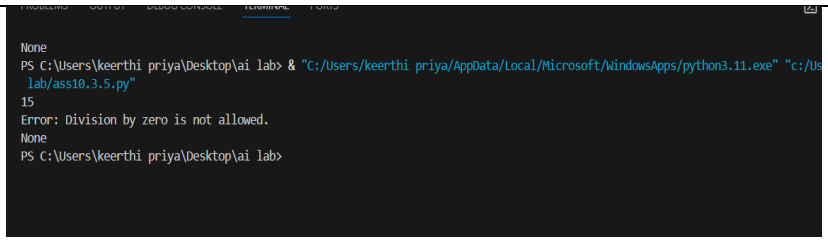
```python
def calculate(x, y, operation):
    """
    Perform basic arithmetic operations.

    Args:
        x (float): First operand.
        y (float): Second operand.
        operation (str): The operation to perform: 'add', 'sub', 'mul', or 'div'.

    Returns:
        float | None: Result of the operation, or None if invalid operation or error.
    """
    if operation == "add":
        return x + y
    elif operation == "sub":
        return x - y
    elif operation == "mul":
        return x * y
    elif operation == "div":
        if y == 0:
            print("Error: Division by zero is not allowed.")
            return None
        return x / y
    else:
        print("Error: Invalid operation.")
        return None


print(calculate(10, 5, "add"))
print(calculate(10, 0, "div"))
```

OUTPUT:



OBSERVATION:

Issues in Original Code:

1. **Missing docstrings** – The function has no explanation of purpose, arguments, or return values.
2. **Inconsistent formatting** – Mixed inline and block returns (elif z=="sub": return x-y).
3. **Division by zero** – No error handling, which can cause runtime exceptions.
4. **Non-descriptive names** – Function calc and parameter z are not descriptive; replaced with calculate and operation.
5. **PEP 8 Violations** – Missing spaces after commas, no blank lines between function and calls, inconsistent indentation.