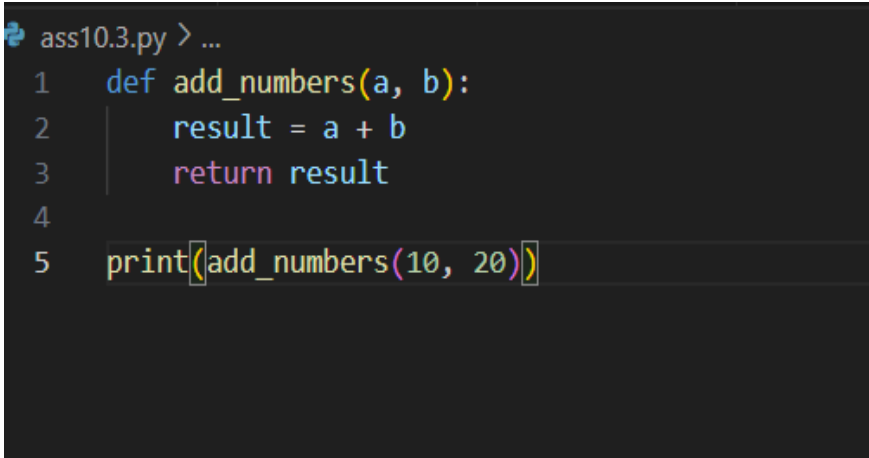
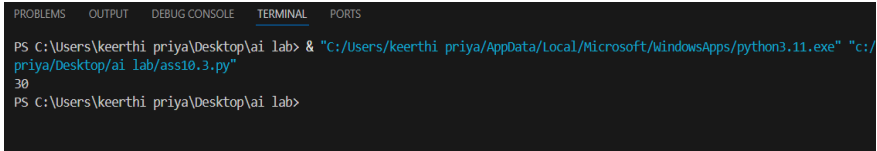


	<p style="text-align: center;">AI ASSISTED CODING</p> <p>NAME:DONTHI MEGHANA HALL TICKET NUM:2403A510D9 ASSIGNMENT:10.3 BATCH:05</p>	
1	<p>TASK1#</p> <p>PROMPT:</p> <p>I have a Python script with syntax, indentation, and variable errors. Please identify and fix them</p> <p># buggy_code_task1.py</p> <pre>def add_numbers(a, b) result = a + b return reslt print(add_numbers(10 20))</pre> <p>CODE:</p>  <p>OUTPUT:</p>  <p>OBSERVATION:</p> <ul style="list-style-type: none"><input type="checkbox"/> Missing Colon in Function Definition:<ul style="list-style-type: none">• 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):<input type="checkbox"/> Incorrect Indentation:<ul style="list-style-type: none">• 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 > ...
1  def calculate_factorial(n):
2      """
3      Calculate the factorial of a given number.
4
5      Args:
6      |   n (int): A non-negative integer.
7
8      Returns:
9      |   int: The factorial of the input number.
10     """
11     result = 1
12     for i in range(1, n + 1):
13         result *= i
14     return result
15
16
17 print(calculate_factorial(5))
18
```

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.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 > ...
1  import sqlite3
2
3
4  def get_user_data(user_id):
5      """
6      Fetch user data from the database by user_id.
7
8      Args:
9      |   user_id (int): ID of the user to look up.
10
11      Returns:
12      |   list: A list of rows matching the user_id, or an empty list if none found.
13      """
14      try:
15          conn = sqlite3.connect("users.db")
16          cursor = conn.cursor()
17          query = "SELECT * FROM users WHERE id = ?;"
18          cursor.execute(query, (user_id,))
19          result = cursor.fetchall()
20      except sqlite3.Error as e:
21          print(f"Database error: {e}")
22          result = []
23      finally:
24          if conn:
25              conn.close()
26      return result
27
28
29  def main():
30      user_input = input("Enter user ID: ").strip()
31
32      if not user_input.isdigit():
33          print("Invalid input. Please enter a numeric user ID.")
34          return
35
36      user_id = int(user_input)
37      data = get_user_data(user_id)

```

Ln 47, Col 1 Spaces: 4 UT

```

ASS10.3.4.PY > ...
29  def main():
30
31
32
33
34
35
36      user_id = int(user_input)
37      data = get_user_data(user_id)
38
39      if data:
40          print("User Data:", data)
41      else:
42          print("No user found with that ID.")
43
44
45  if __name__ == "__main__":
46      main()
47

```

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: 2403AS10G4
Enter user ID: 2403AS10G4
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
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 > ...
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:

```
None
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "C:/Users/
lab/ass10.3.5.py"
15
Error: Division by zero is not allowed.
None
PS C:\Users\keerthi priya\Desktop\ai lab>
```

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.
-