ASSIGNMENT – 10

NAME: G. OMKAR

HT.NO: 2403A52039

BATCH: 03

Task-1:

Syntax and Error Detection

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

```
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: Identify and fix syntax, indentation, and variable errors in the

given script.

def add_numbers(a, b)

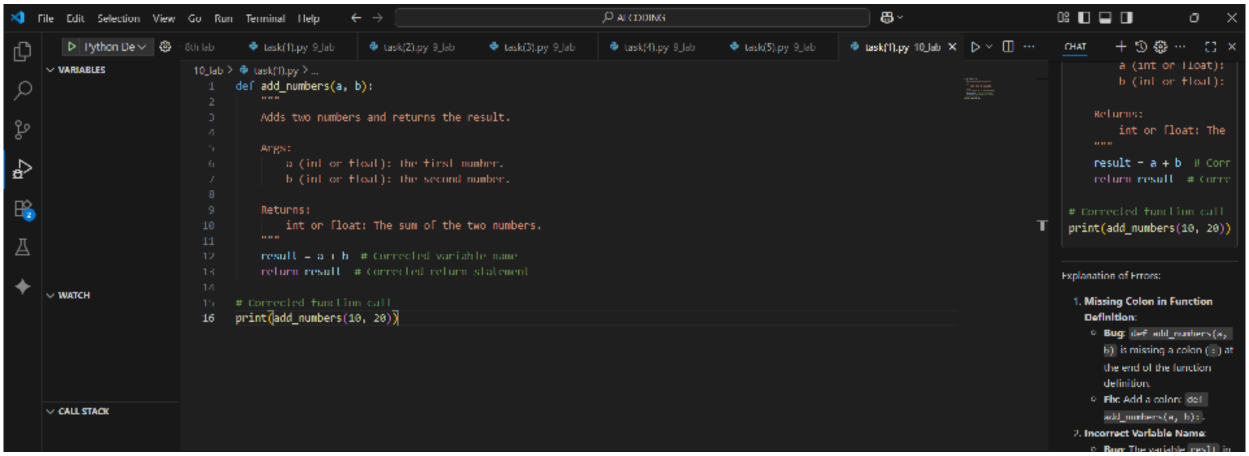
result = a + b

return reslt

print(add_numbers(10 20)).

even expain the errors.

Code:

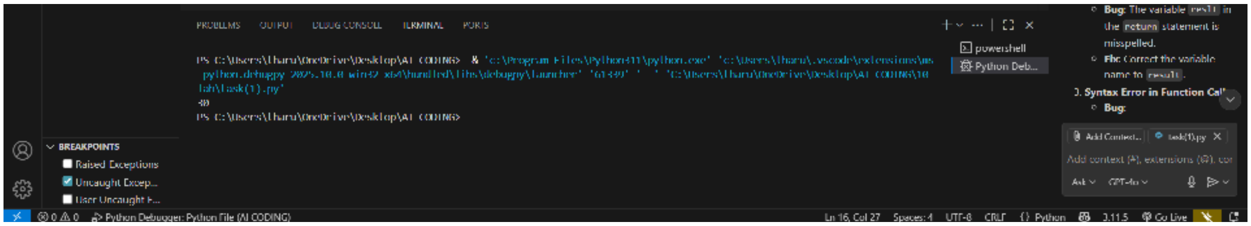


```
def add_numbers(a, b):
    """
    Adds two numbers and returns the result.

    Args:
        a (int or float): the first number.
        b (int or float): the second number.

    Returns:
        int or float: The sum of the two numbers.
    """
    result = a + b  # Corrected variable name
    return result  # Corrected return statement

# Corrected function call
print(add_numbers(10, 20))
```

Explanation of Errors:

1. **Missing Colon in Function Definition:**
   - **Bug:** `def add_numbers(a, b)` is missing a colon (`:`) at the end of the function definition.
   - **Fix:** Add a colon: `def add_numbers(a, b):`.
2. **Incorrect Variable Name:**
   - **Bug:** The variable `reslt` in

OP:



```
PS C:\Users\Charu\OneDrive\Desktop\AI CODING>  & 'c:\Program Files\Python311\python.exe' 'c:\Users\Charu\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '61339' '--' 'C:\Users\Charu\OneDrive\Desktop\AI CODING\10_lab\task(1).py'
30
PS C:\Users\Charu\OneDrive\Desktop\AI CODING>
```

   - **Bug:** The variable `reslt` in the `return` statement is misspelled.
   - **Fix:** Correct the variable name to `result`.
3. **Syntax Error in Function Call**
   - **Bug:**

Observation: It has corrected the errors and even explained it.Bug: def add_numbers(a, b) is missing a colon (:) at the end of the function definition. Fix: Add a colon: def add_numbers(a, b):. Bug: The variable reslt in the return statement is misspelled. Fix: Correct the variable name to result. Bug: print(add_numbers(10

20)) is missing a comma between 10 and 20. Fix: Add a comma: print(add_numbers(10, 20)). Bug: The print statement has an extra closing parenthesis: print(add_numbers(10, 20)). Fix: Remove the extra parenthesis.

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: Optimize inefficient logic while keeping the result correct.

def find_duplicates(nums):

duplicates = []

for i in range(len(nums)):

for j in range(len(nums)):

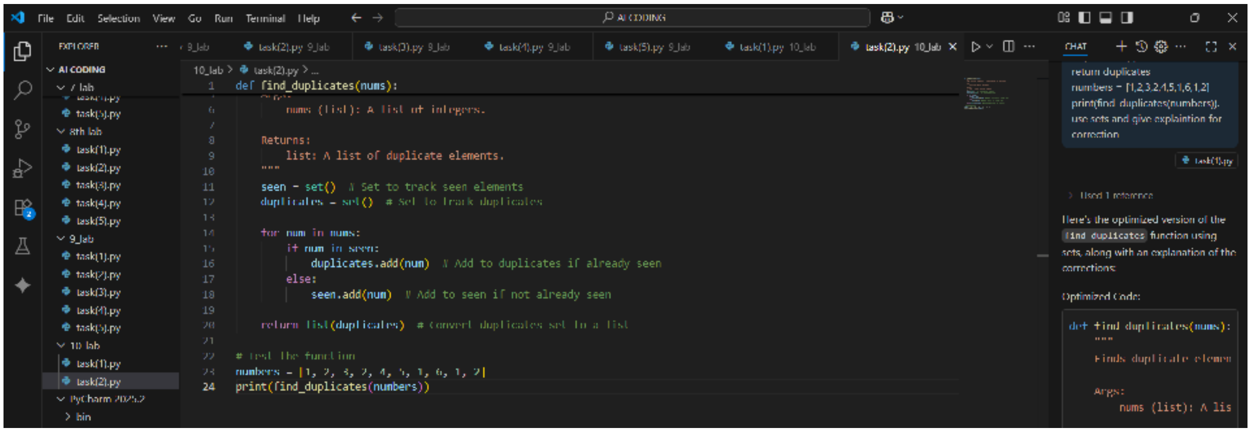
if i!= j and nums[i] == nums[j] and nums[i] not in duplicates:

duplicates.append(nums[i])

return duplicates numbers = [1,2,3,2,4,5,1,6,1,2].

print(find_duplicates(numbers)). use sets and give explaintion for correction

Code:



```python
def find_duplicates(nums):
        nums (list): A list of integers.

    Returns:
        list: A list of duplicate elements.
    """
    seen = set()  # Set to track seen elements
    duplicates = set()  # Set to track duplicates

    for num in nums:
        if num in seen:
            duplicates.add(num)  # Add to duplicates if already seen
        else:
            seen.add(num)  # Add to seen if not already seen

    return list(duplicates)  # Convert duplicates set to a list

# Test the function
numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
print(find_duplicates(numbers))
```

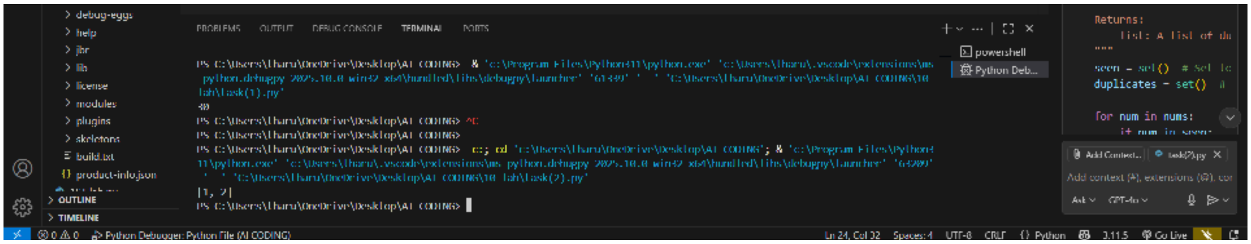
return duplicates
numbers = [1,2,3,2,4,5,1,6,1,2]
print(find_duplicates(numbers)).
use sets and give explaintion for correction

Here's the optimized version of the `find_duplicates` function using sets, along with an explanation of the corrections:

Optimized Code:

```python
def find_duplicates(nums):
    """
    Finds duplicate elemen

    Args:
        nums (list): A lis
```

OP:



```
PS C:\Users\Tharu\OneDrive\Desktop\AI CODING>  & 'c:\Program Files\Python311\python.exe' 'c:\Users\Tharu\.vscode\extensions\ms
 python.debugpy 2025.10.0 win32 x64\bundled\libs\debugpy\launcher' '61489' '  ' 'C:\Users\Tharu\OneDrive\Desktop\AI CODING\10
lab\Task(1).py'
40
PS C:\Users\Tharu\OneDrive\Desktop\AI CODING> ^C
PS C:\Users\Tharu\OneDrive\Desktop\AI CODING>
PS C:\Users\Tharu\OneDrive\Desktop\AI CODING>  c:; cd 'c:\Users\Tharu\OneDrive\Desktop\AI CODING'; & 'c:\Program Files\Python3
11\python.exe' 'c:\Users\Tharu\.vscode\extensions\ms python.debugpy 2025.10.0 win32 x64\bundled\libs\debugpy\launcher' '64209'
 '  ' 'C:\Users\Tharu\OneDrive\Desktop\AI CODING\10 lab\Task(2).py'
[1, 2]
PS C:\Users\Tharu\OneDrive\Desktop\AI CODING> 
```

```
    Returns:
        list: A list of du
    """
    seen = set()  # Set to
    duplicates = set()  #

    for num in nums:
        if num in seen:
```

Observation:

Bug: The original code uses nested loops (for i in range(len(nums)) and for j in range(len(nums))), resulting in an O(n²) time complexity. Fix: Replaced the nested loops with a single loop and used sets to track seen elements and duplicates, reducing the time complexity to O(n). Bug: The original code uses a list (duplicates) to store duplicates, which requires checking if an element is already in the list (nums[i] not in duplicates), making it inefficient. Fix: Used a set (duplicates) to store duplicates, as set lookups are faster (average O(1) time complexity). Bug: The original code checks i != j and compares every pair of elements, which is unnecessary. Fix: Simplified the logic by using a single loop. If an element is already in the seen set, it is added to the duplicates set. Fix: Converted the duplicates set to a list before returning, as the output format is expected to be a list.

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

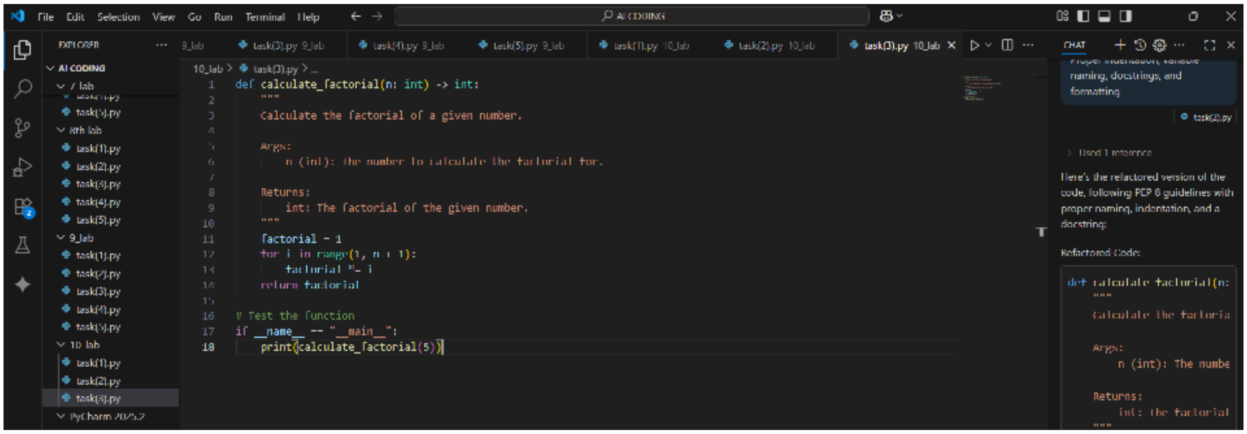
```
def c(n):
x=1
for i in range(1,n+1):
x=x*i
```

return x

print(c(5))

Function renamed to calculate_factorial. Proper indentation, variable naming, docstrings, and formatting

Code:



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

    Args:
        n (int): The number to calculate the factorial for.

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

# Test the function
if __name__ == "__main__":
    print(calculate_factorial(5))
```

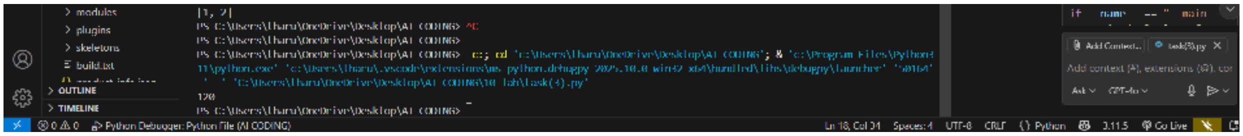
naming, docstrings, and formatting

Here's the refactored version of the code, following PEP 8 guidelines with proper naming, indentation, and a docstring:

Refactored Code:

```python
def calculate_factorial(n:
    """
    Calculate the factoria

    Args:
        n (int): The numbe

    Returns:
        int: The factorial
    """
```

OP:



```
[1, 2]
PS C:\Users\Charu\OneDrive\Desktop\AI CODING> ^C
PS C:\Users\Charu\OneDrive\Desktop\AI CODING>
PS C:\Users\Charu\OneDrive\Desktop\AI CODING>  c:; cd 'c:\Users\Charu\OneDrive\Desktop\AI CODING'; & 'c:\Program Files\Python3
11\python.exe' 'c:\Users\Charu\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '50164'
 '--' 'C:\Users\Charu\OneDrive\Desktop\AI CODING\10_lab\task(4).py'
120
PS C:\Users\Charu\OneDrive\Desktop\AI CODING>
```

Observation: Renamed c to calculate_factorial to make the function name descriptive and meaningful.
Renamed x to factorial for clarity. Added a Google-style docstring to describe the function's purpose, arguments, and return value. Ensured proper indentation and spacing.

Added type hints for the function argument and return value. Wrapped the test call in an if __name__ == "__main__": block to ensure the function is only executed when the script is run directly.

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: Add security practices and exception handling to the code.

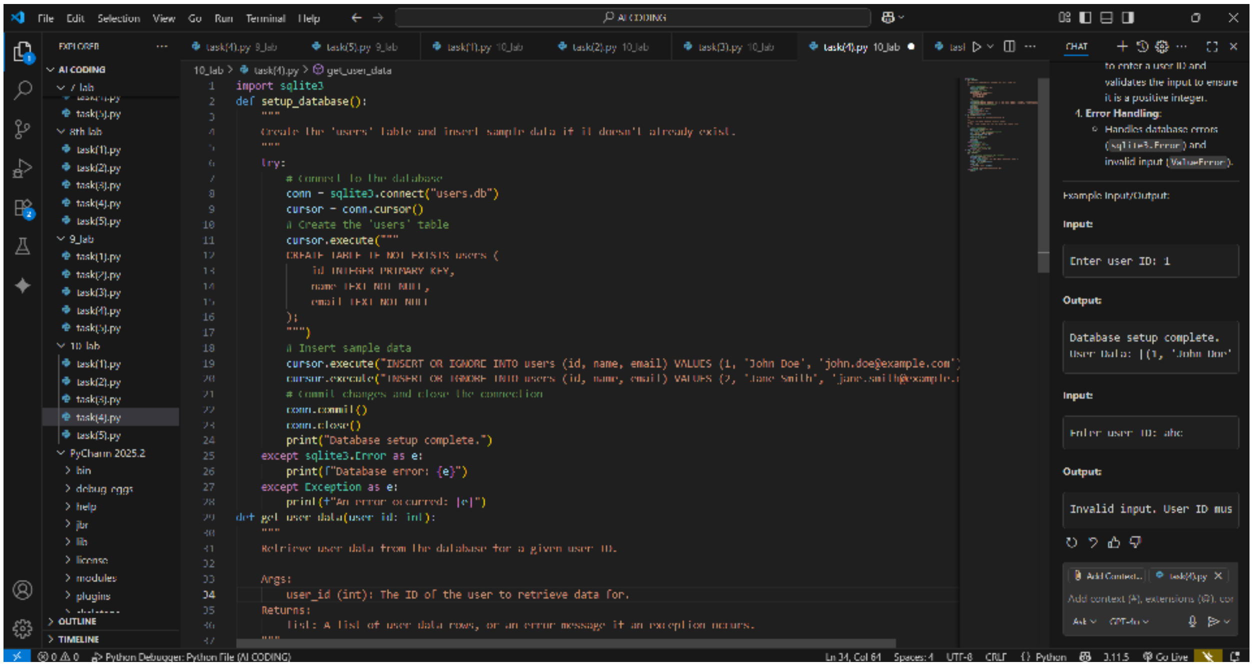
```
import sqlite3

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)).

Safe query using parameterized SQL (? placeholders). Try-except block for database errors. Input validation before query execution.

Code:



```python
import sqlite3
def setup_database():
    """
    Create the 'users' table and insert sample data if it doesn't already exist.
    """
    try:
        # Connect to the database
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()
        # Create the 'users' table
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY,
            name TEXT NOT NULL,
            email TEXT NOT NULL
        );
        """)
        # Insert sample data
        cursor.execute("INSERT OR IGNORE INTO users (id, name, email) VALUES (1, 'John Doe', 'john.doe@example.com'
        cursor.execute("INSERT OR IGNORE INTO users (id, name, email) VALUES (2, 'Jane Smith', 'jane.smith@example.
        # Commit changes and close the connection
        conn.commit()
        conn.close()
        print("Database setup complete.")
    except sqlite3.Error as e:
        print(f"Database error: {e}")
    except Exception as e:
        print(f"An error occurred: {e}")
def get_user_data(user_id: int):
    """
    Retrieve user data from the database for a given user ID.

    Args:
        user_id (int): The ID of the user to retrieve data for.
    Returns:
        list: A list of user data rows, or an error message if an exception occurs.
```

to enter a user ID and validates the input to ensure it is a positive integer.

4. **Error Handling:**
    - Handles database errors (`sqlite3.Error`) and invalid input (`ValueError`).

Example Input/Output:

Input:

```
Enter user ID: 1
```

Output:

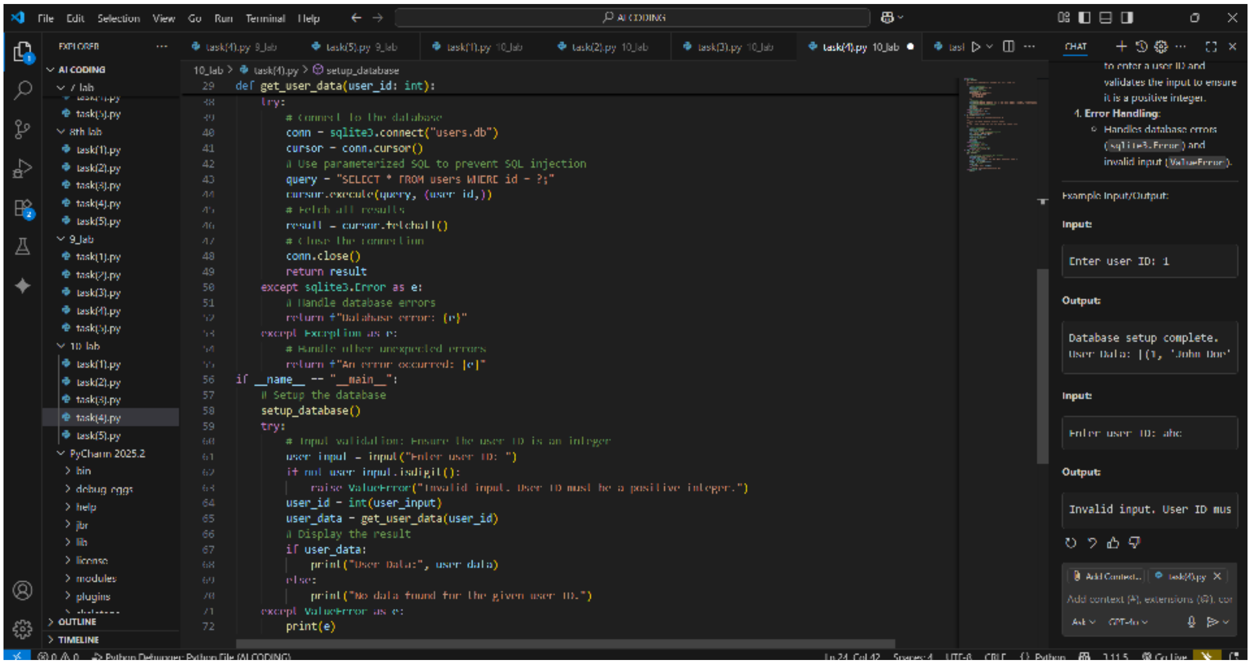
```
Database setup complete.
User Data: [(1, 'John Doe'
```

Input:

```
Enter user ID: abc
```

Output:

```
Invalid input. User ID mus
```



```python
def get_user_data(user_id: int):
    try:
        # Connect to the database
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()
        # Use parameterized SQL to prevent SQL injection
        query = "SELECT * FROM users WHERE id = ?;"
        cursor.execute(query, (user_id,))
        # Fetch all results
        result = cursor.fetchall()
        # Close the connection
        conn.close()
        return result
    except sqlite3.Error as e:
        # Handle database errors
        return f"Database error: {e}"
    except Exception as e:
        # Handle other unexpected errors
        return f"An error occurred: {e}"
if __name__ == "__main__":
    # Setup the database
    setup_database()
    try:
        # Input validation: Ensure the user ID is an integer
        user_input = input("Enter user ID: ")
        if not user_input.isdigit():
            raise ValueError("Invalid input. User ID must be a positive integer.")
        user_id = int(user_input)
        user_data = get_user_data(user_id)
        # Display the result
        if user_data:
            print("User Data:", user_data)
        else:
            print("No data found for the given user ID.")
    except ValueError as e:
        print(e)
```

to enter a user ID and validates the input to ensure it is a positive integer.

4. **Error Handling:**
   - Handles database errors (`sqlite3.Error`) and invalid input (`ValueError`).

Example Input/Output:

Input:

```
Enter user ID: 1
```

Output:

```
Database setup complete.
User Data: [(1, 'John Doe'
```

Input:

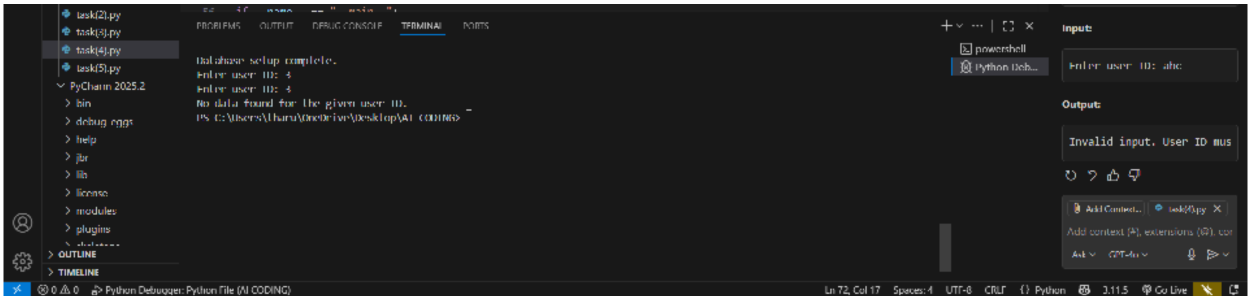
```
Enter user ID: abc
```

Output:

```
Invalid input. User ID mus
```

File Edit Selection View Go Run Terminal Help
AI CODING
EXPLORER
task(4).py 9_lab
task(5).py 9_lab
task(1).py 10_lab
task(2).py 10_lab
task(3).py 10_lab
task(4).py 10_lab
CHAT
Add Context...
Ask
GPT-4o
Python Debugger: Python File (AI CODING)
Ln 24, Col 42 Spaces: 4 UTF-8 CRLF Python 3.11.5 Go Live

OP:



```
Database setup complete.
Enter user ID: 3
Enter user ID: 3
No data found for the given user ID.
PS C:\Users\Charu\OneDrive\Desktop\AI CODING>
```

Input:

```
Enter user ID: abc
```

Output:

```
Invalid input. User ID mus
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
Ln 72, Col 17 Spaces: 4 UTF-8 CRLF Python 3.11.5 Go Live

Observation:

This Python script is designed to manage a simple SQLite database for user data. It includes two main functions: `setup_database` and `get_user_data`. The `setup_database` function ensures that the database (users.db) is properly initialized. It creates a `users` table if it doesn't already exist and inserts sample data for testing purposes. This function uses the `CREATE TABLE IF NOT EXISTS` SQL statement to avoid duplicate table creation and the `INSERT OR IGNORE` statement to prevent duplicate entries. The `get_user_data` function retrieves user data for a given user ID using parameterized SQL queries (`?`

placeholders) to prevent SQL injection attacks. It connects to the database, executes the query, fetches the results, and closes the connection. The script also includes robust error handling to manage database-related errors (`sqlite3.Error`) and unexpected exceptions. In the `if __name__ == "__main__":` block, the script first sets up the database and then prompts the user to enter a user ID. The input is validated to ensure it is a positive integer, and if valid, the `get_user_data` function is called to fetch and display the user data. If the user ID is invalid or no data is found, appropriate error messages are displayed. This script demonstrates best practices for database management, including input validation, parameterized queries, and proper resource handling.

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

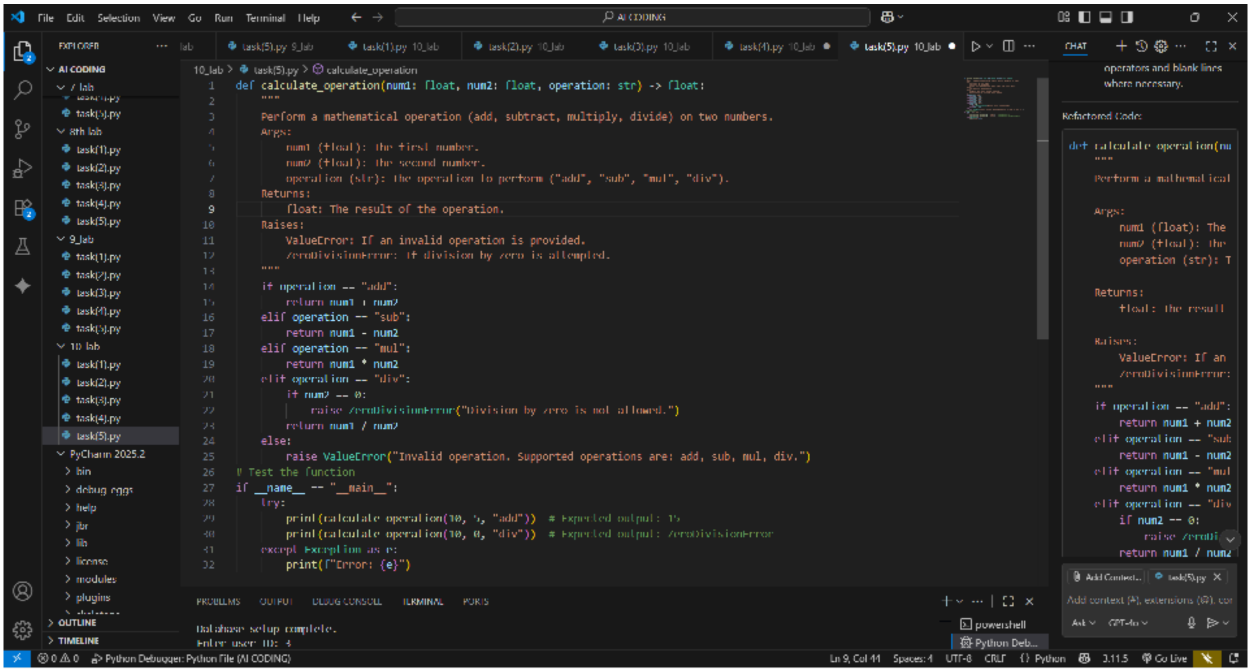
Prompt: Generate a review report for this messy code.

```
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"))
```

include: 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

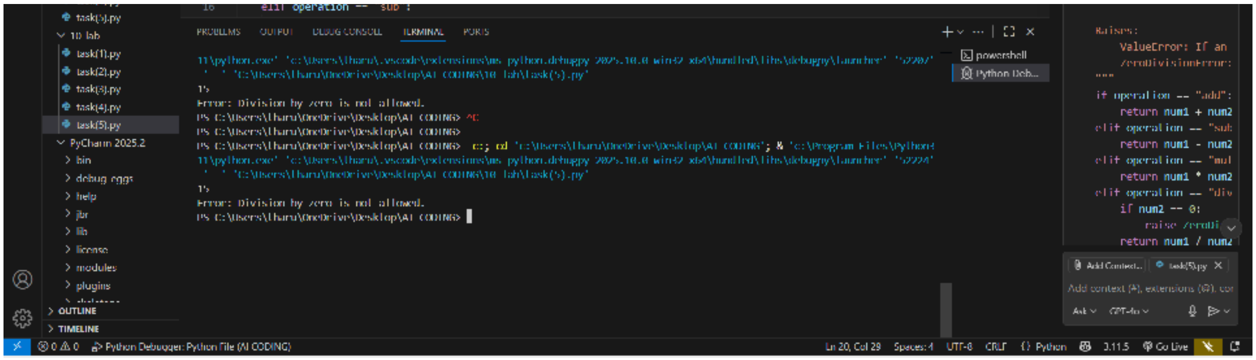
Code:



```python
def calculate_operation(num1: float, num2: float, operation: str) -> float:
    """
    Perform a mathematical operation (add, subtract, multiply, divide) on two numbers.
    Args:
        num1 (float): The first number.
        num2 (float): The second number.
        operation (str): The operation to perform ("add", "sub", "mul", "div").
    Returns:
        float: The result of the operation.
    Raises:
        ValueError: If an invalid operation is provided.
        ZeroDivisionError: If division by zero is attempted.
    """
    if operation == "add":
        return num1 + num2
    elif operation == "sub":
        return num1 - num2
    elif operation == "mul":
        return num1 * num2
    elif operation == "div":
        if num2 == 0:
            raise ZeroDivisionError("Division by zero is not allowed.")
        return num1 / num2
    else:
        raise ValueError("Invalid operation. Supported operations are: add, sub, mul, div.")
# Test the function
if __name__ == "__main__":
    try:
        print(calculate_operation(10, 5, "add"))  # Expected output: 15
        print(calculate_operation(10, 0, "div"))  # Expected output: ZeroDivisionError
    except Exception as e:
        print(f"Error: {e}")
```

```
Database setup complete.
Enter user ID: 3
```

OP:



```
11\python.exe' 'c:\Users\tharu\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '52207' '--' 'C:\Users\tharu\OneDrive\Desktop\AI CODING\10 lab\task(5).py'
15
Error: Division by zero is not allowed.
PS C:\Users\tharu\OneDrive\Desktop\AI CODING> ^C
PS C:\Users\tharu\OneDrive\Desktop\AI CODING>
PS C:\Users\tharu\OneDrive\Desktop\AI CODING>  c:; cd 'c:\Users\tharu\OneDrive\Desktop\AI CODING'; & 'c:\Program Files\Python3
11\python.exe' 'c:\Users\tharu\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '52224' '--' 'C:\Users\tharu\OneDrive\Desktop\AI CODING\10 lab\task(5).py'
15
Error: Division by zero is not allowed.
PS C:\Users\tharu\OneDrive\Desktop\AI CODING>
```

Observation: A detailed Google-style docstring was added to describe the function's purpose, parameters, return values, and exceptions. Indentation and formatting were made consistent. Removed inline returns for better readability. Added a check for division by zero and raised a ZeroDivisionError with a clear message. Renamed calc to calculate_operation and variables x, y, and z to num1, num2, and operation. Added spaces around operators and blank lines where necessary