AI ASSISSTED CODING – 10.3

NAME : S.NADHIYA

ROLLNO : 2403A510C6

BATCH : 05

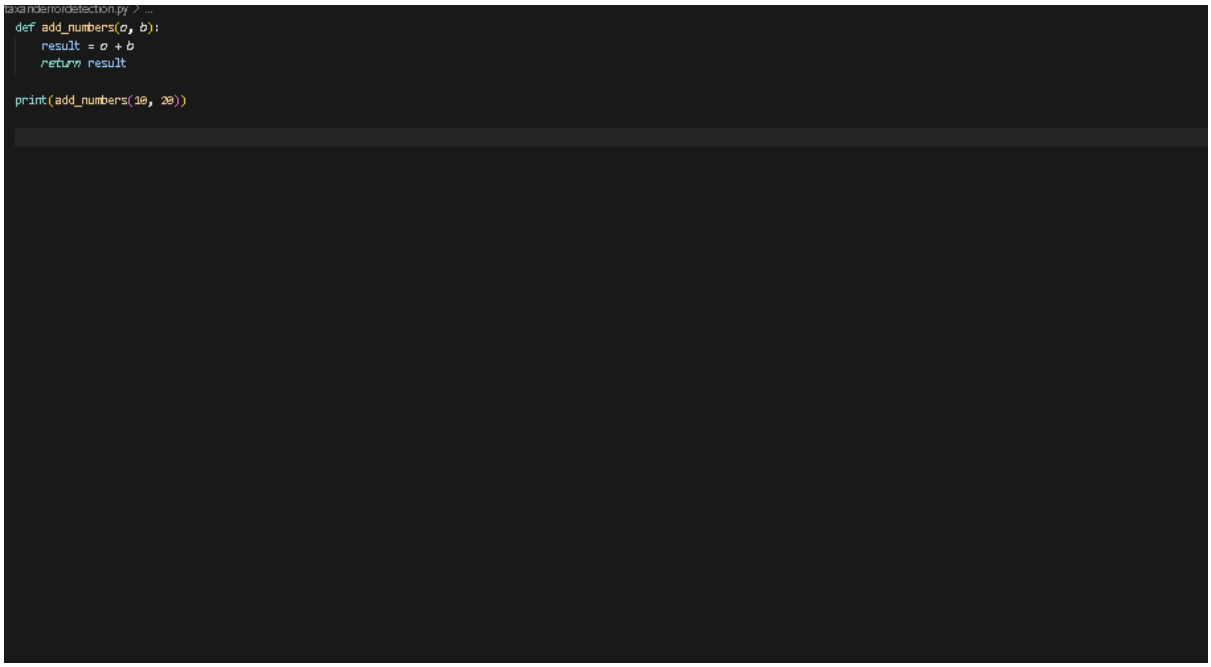
DEPT : CSE

Task – 01:

Prompt:

def add_numbers(a, b)
result = a + b
return reslt
print(add_numbers(10 20))identify and fix syntax errors, indentation issues, and variable name mistakes. Also explain what was corrected and also explain the code .

Code generated:



Explanation:

Added a colon (:) After the function definition.

Fixed indentation for the function body.

Corrected the variable name from reslt to result in the return statement.

Added a comma between 10 and 20 in the function call.

Output:

Observation:

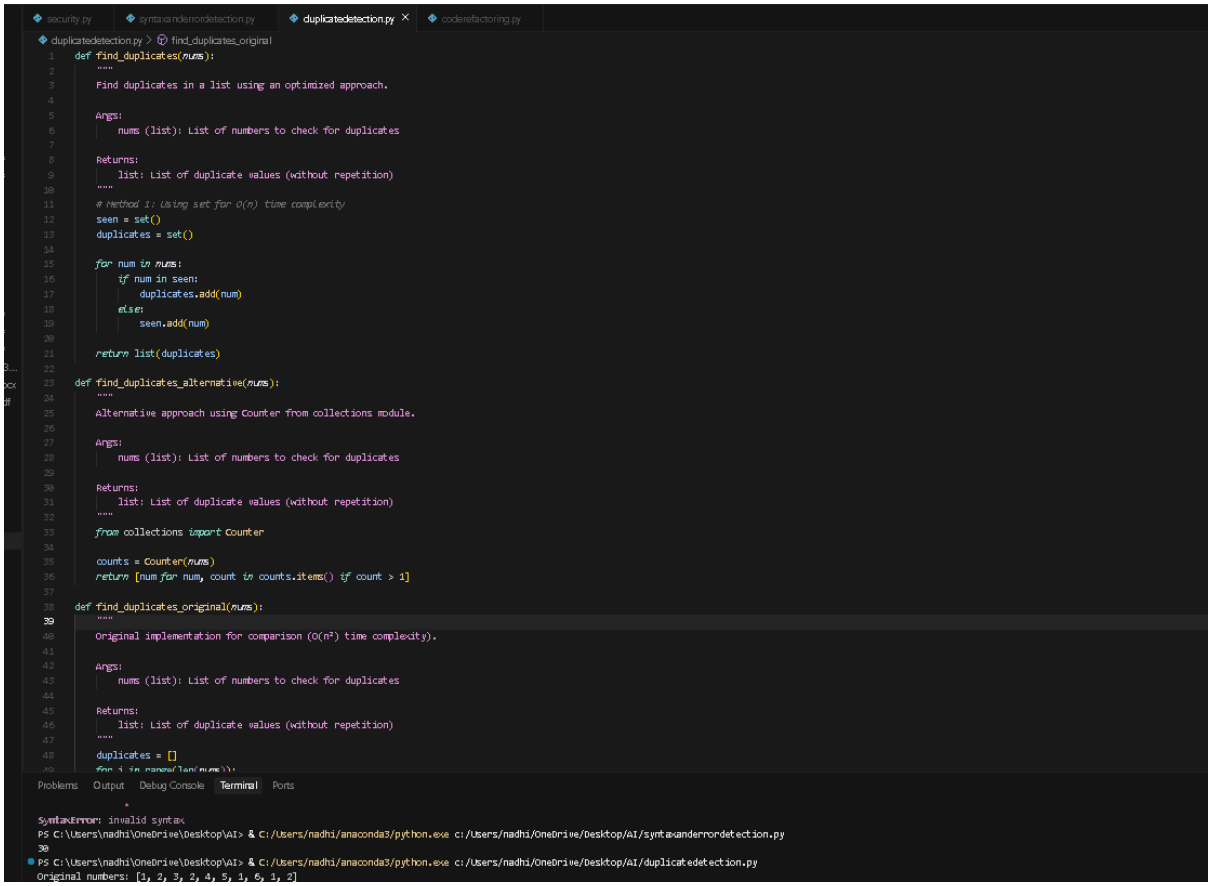
The file contains a simple Python function add_numbers() that performs basic addition and returns the result, followed by a function call that prints the output.The code appears to be a basic example for syntax and error detection, with proper Python syntax

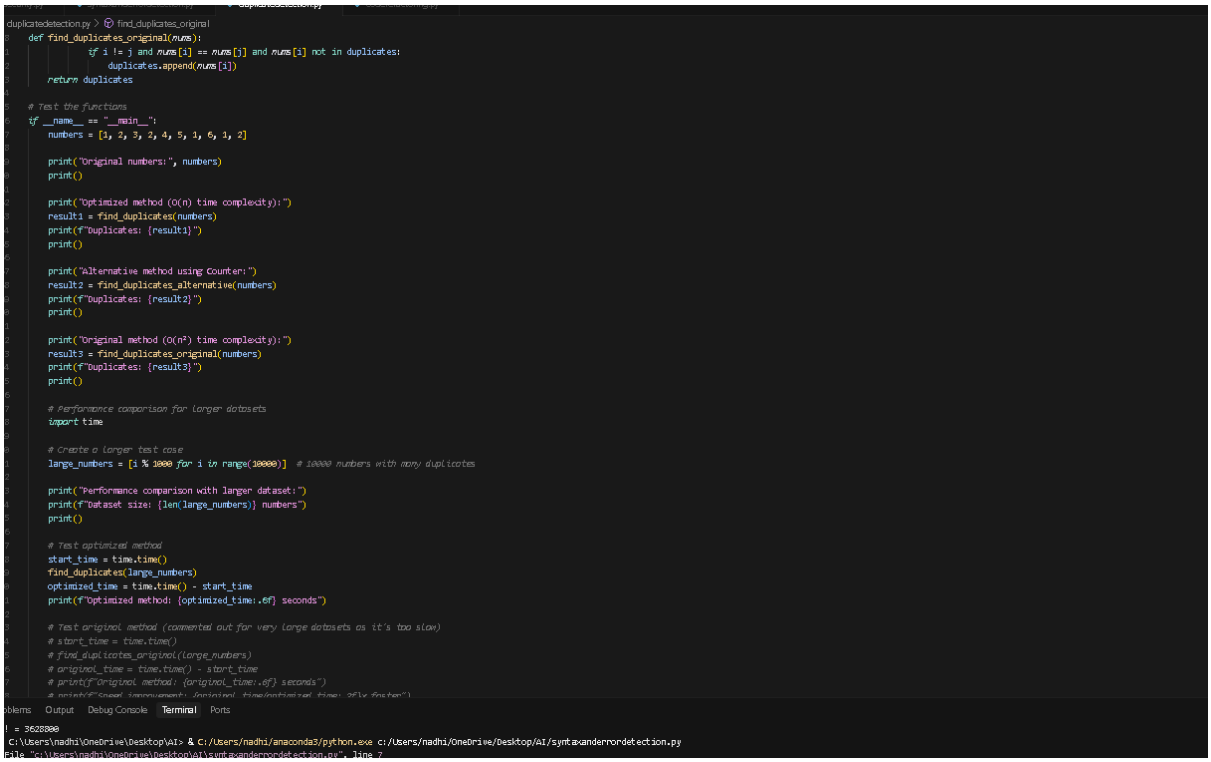
Task – 02

Prompt:

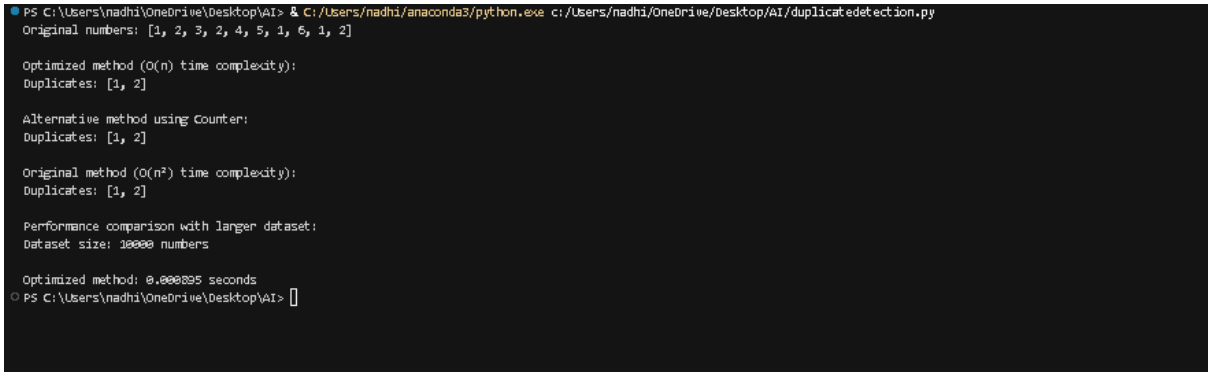
def find_duplicates(nums): duplicates = [] for i in range(len(nums)): for j in range(len(nums)): if i != j and nums[i] == nums[j] and nums[i] not in duplicates: duplicates.append(nums[i]) return duplicates numbers = [1,2,3,2,4,5,1,6,1,2] print(find_duplicates(numbers)) finds duplicates in a list using nested loops. Please optimize the logic to make it more efficient while keeping the result correct. The script is

Code generated:


security.py
syntaxanderrordetection.py
duplicatedetection.py
coderefactoring.py
duplicatedetection.py > find_duplicates_original
def find_duplicates(nums):
"""
Find duplicates in a list using an optimized approach.
Args:
nums (list): List of numbers to check for duplicates
Returns:
list: List of duplicate values (without repetition)
"""
# Method 1: Using set for O(n) time complexity
seen = set()
duplicates = set()
for num in nums:
if num in seen:
duplicates.add(num)
else:
seen.add(num)
return list(duplicates)
def find_duplicates_alternative(nums):
"""
Alternative approach using Counter from collections module.
Args:
nums (list): List of numbers to check for duplicates
Returns:
list: List of duplicate values (without repetition)
"""
from collections import Counter
counts = Counter(nums)
return [num for num, count in counts.items() if count > 1]
def find_duplicates_original(nums):
"""
Original implementation for comparison (O(n²) time complexity).
Args:
nums (list): List of numbers to check for duplicates
Returns:
list: List of duplicate values (without repetition)
"""
duplicates = []
Problems
Output
Debug Console
Terminal
Ports
SyntaxError: invalid syntax
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/syntaxanderrordetection.py
38
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/duplicatedetection.py
Original numbers: [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]


duplicatedetection.py > find_duplicates_original
def find_duplicates_original(nums):
if i != j and nums[i] == nums[j] and nums[i] not in duplicates:
duplicates.append(nums[i])
return duplicates
# Test the functions
if __name__ == "__main__":
numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
print("Original numbers:", numbers)
print()
print("Optimized method (O(n) time complexity):")
result1 = find_duplicates(numbers)
print(f"Duplicates: {result1}")
print()
print("Alternative method using Counter:")
result2 = find_duplicates_alternative(numbers)
print(f"Duplicates: {result2}")
print()
print("Original method (O(n²) time complexity):")
result3 = find_duplicates_original(numbers)
print(f"Duplicates: {result3}")
print()
# Performance comparison for larger datasets
import time
# Create a larger test case
large_numbers = [i % 1000 for i in range(10000)] # 10000 numbers with many duplicates
print("Performance comparison with larger dataset:")
print(f"Dataset size: {len(large_numbers)} numbers")
print()
# Test optimized method
start_time = time.time()
find_duplicates(large_numbers)
optimized_time = time.time() - start_time
print(f"Optimized method: {optimized_time:.6f} seconds")
# Test original method (commented out for very large datasets as it's too slow)
# start_time = time.time()
# find_duplicates_original(large_numbers)
# original_time = time.time() - start_time
# print(f"Original method: {original_time:.6f} seconds")
Problems
Output
Debug Console
Terminal
Ports
= 3628800
C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/syntaxanderrordetection.py

Output:



Observation:

The code demonstrates three different approaches to finding duplicates in a list, showcasing algorithmic optimization from $O(n^2)$ to O(n) time complexity. The file includes comprehensive testing with both small and large datasets, along with performance timing comparisons to illustrate the efficiency gains of the optimized methods.

Task – 03

Prompt:

Write a Python program that calculates the factorial of a given number.

- Define a function that takes an integer n as input.
- The function should return the factorial of n (i.e., the product of all positive integers from 1 to n).
- Ensure the code follows PEP 8 style guidelines (proper naming, indentation, and readability).
- Add a docstring to describe the function.
- Test your function by calculating the factorial of 5.

Code generated:

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

    The factorial of a non-negative integer n is the product of all positive
    integers from 1 to n. For example, 5! = 5 × 4 × 3 × 2 × 1 = 120.

    Args:
        n (int): A non-negative integer for which to calculate the factorial.

    Returns:
        int: The factorial of n.

    Raises:
        ValueError: If n is negative.

    Examples:
        >>> calculate_factorial(5)
        120
        >>> calculate_factorial(0)
        1
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    if n == 0 or n == 1:
        return 1

    result = 1
    for i in range(2, n + 1):
        result *= i

    return result


def main():
    """
```

```python
def calculate_factorial(n):
        raise ValueError("Factorial is not defined for negative numbers")

    if n == 0 or n == 1:
        return 1

    result = 1
    for i in range(2, n + 1):
        result *= i

    return result


def main():
    """
    Main function to test the factorial calculation.
    """
    # Test the function with n = 5
    test_number = 5
    factorial_result = calculate_factorial(test_number)

    print(f"The factorial of {test_number} is: {factorial_result}")

    # Additional test cases
    test_cases = [0, 1, 3, 6, 10]
    print("\nAdditional test cases:")
    for num in test_cases:
        result = calculate_factorial(num)
        print(f"{num}! = {result}")


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

Output:

```
No data found or error occurred.
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/coderefactoring.py
The factorial of 5 is: 120

Additional test cases:
0! = 1
1! = 1
3! = 6
6! = 720
10! = 3628800
PS C:\Users\nadhi\OneDrive\Desktop\AI>
```

Ctrl+K to generate a command

Observation:

The code demonstrates good refactoring practices with a well-documented factorial function that includes comprehensive docstrings, proper error handling, and clear separation of concerns between the calculation logic and testing.The implementation uses an iterative approach for factorial calculation which is more memory-efficient than recursion, and includes edge case handling for negative numbers and base cases (0 and 1)

Task – 04

Prompt:

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()

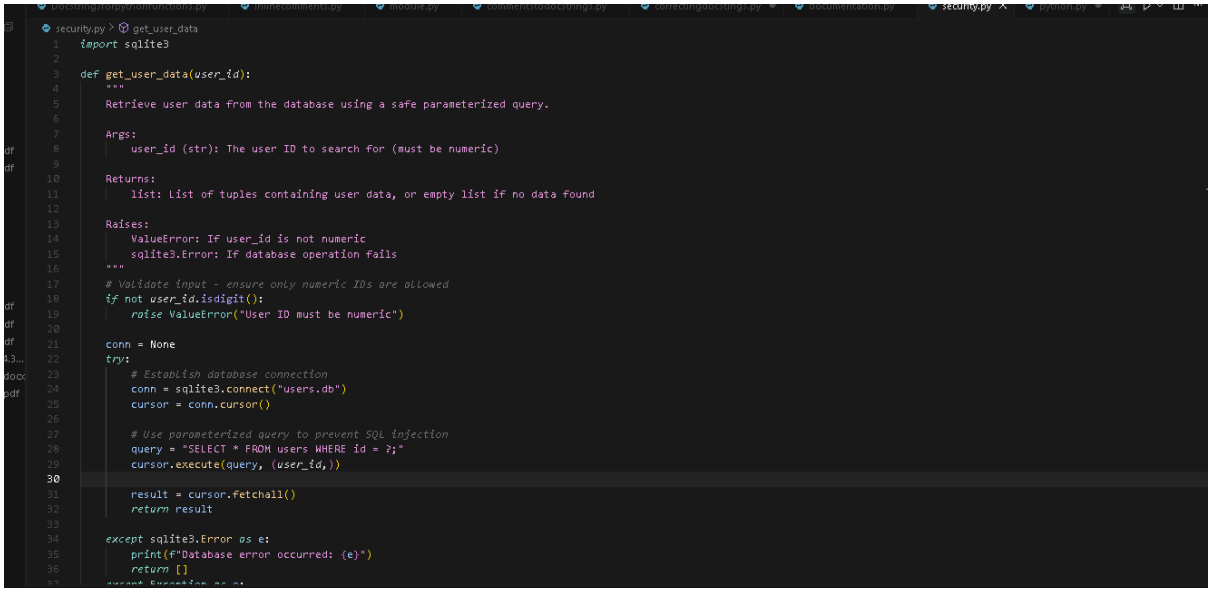
return result

user_input = input("Enter user ID: ")

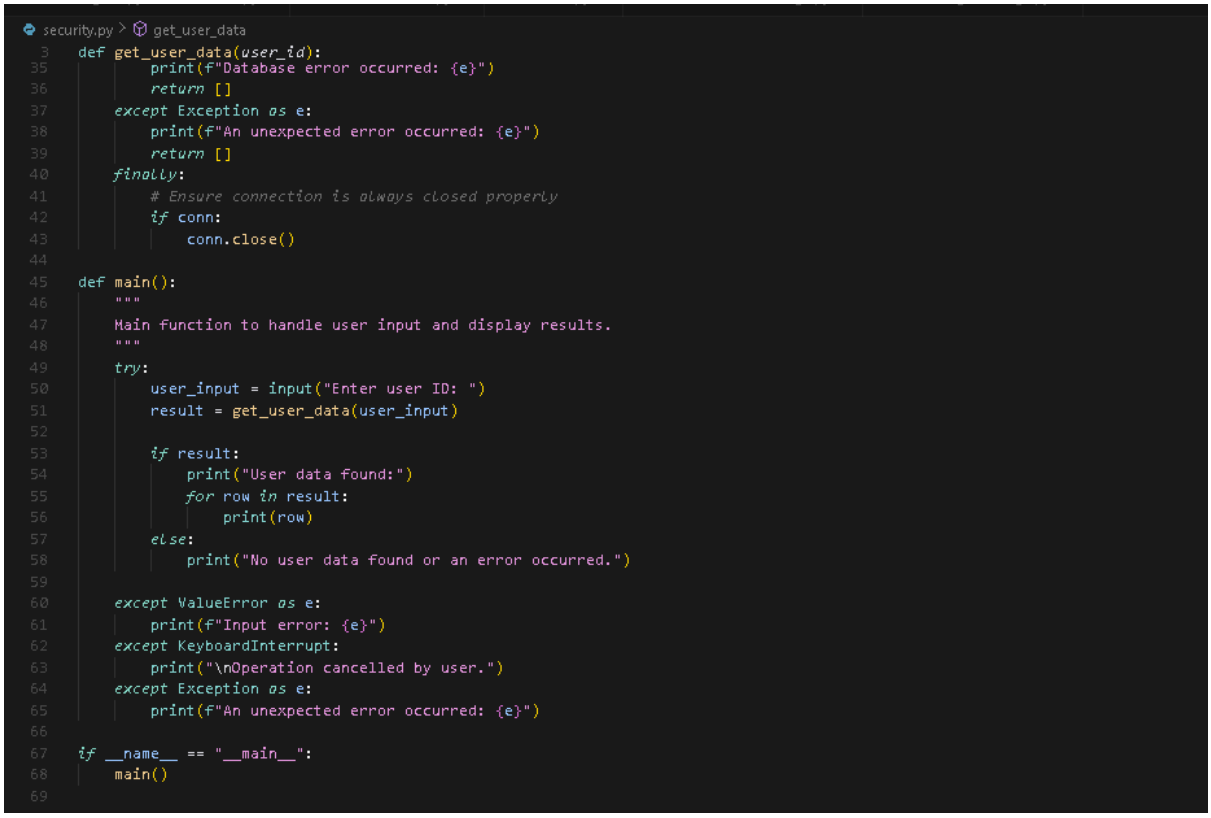
print(get_user_data(user_input))

rewrite code with the following improvements:Use parameterized SQL queries with ? placeholders.Add try-except-finally blocks to handle database errors safely.Validate user input so only numeric IDs are allowed.Ensure the connection is always closed properly.

Code generated:

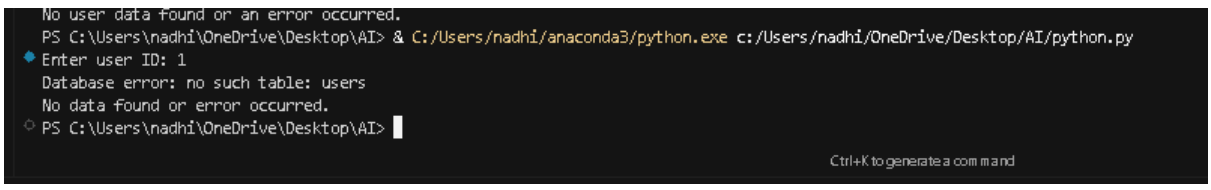


```python
import sqlite3

def get_user_data(user_id):
    """
    Retrieve user data from the database using a safe parameterized query.

    Args:
        user_id (str): The user ID to search for (must be numeric)

    Returns:
        list: List of tuples containing user data, or empty list if no data found

    Raises:
        ValueError: If user_id is not numeric
        sqlite3.Error: If database operation fails
    """
    # Validate input - ensure only numeric IDs are allowed
    if not user_id.isdigit():
        raise ValueError("User ID must be numeric")

    conn = None
    try:
        # Establish database connection
        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 sqlite3.Error as e:
        print(f"Database error occurred: {e}")
        return []
```



```python
def get_user_data(user_id):
        print(f"Database error occurred: {e}")
        return []
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return []
    finally:
        # Ensure connection is always closed properly
        if conn:
            conn.close()

def main():
    """
    Main function to handle user input and display results.
    """
    try:
        user_input = input("Enter user ID: ")
        result = get_user_data(user_input)

        if result:
            print("User data found:")
            for row in result:
                print(row)
        else:
            print("No user data found or an error occurred.")

    except ValueError as e:
        print(f"Input error: {e}")
    except KeyboardInterrupt:
        print("\nOperation cancelled by user.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

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

Output:



```
No user data found or an error occurred.
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
Enter user ID: 1
Database error: no such table: users
No data found or error occurred.
PS C:\Users\nadhi\OneDrive\Desktop\AI>
```

Observation:

Security Enhancement: The code now uses parameterized queries (? placeholders) instead of string formatting, completely eliminating the SQL injection vulnerability that existed in the original code where user input was directly interpolated into the SQL string.

Robust Error Handling: The implementation includes comprehensive try-except-finally blocks with proper connection cleanup, input validation for numeric-only user IDs, and graceful error handling that prevents the application from crashing while providing meaningful error messages to users.

Task -05:

Prompt:

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"))

generate a review report that identifies issues and suggests improvements. The review should cover:

Missing docstrings and documentation

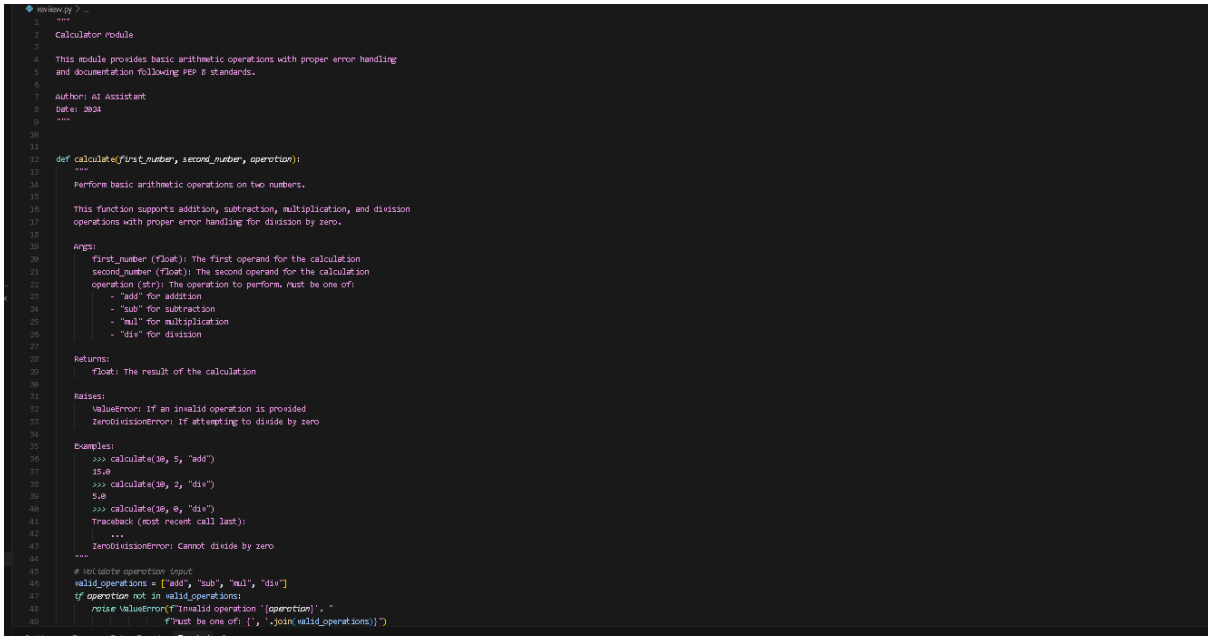
Inconsistent formatting (indentation and inline return statements)

Missing error handling for division by zero

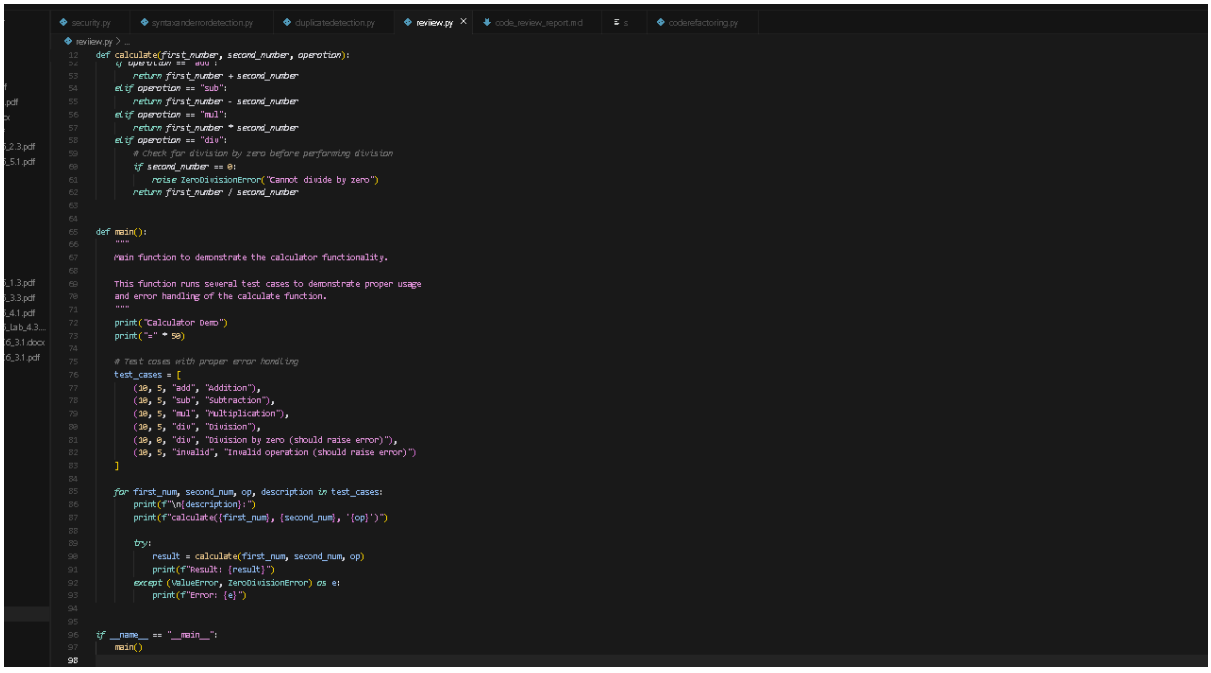
Non-descriptive function and variable names

Suggestions for readability and PEP 8 compliance

Code generated:



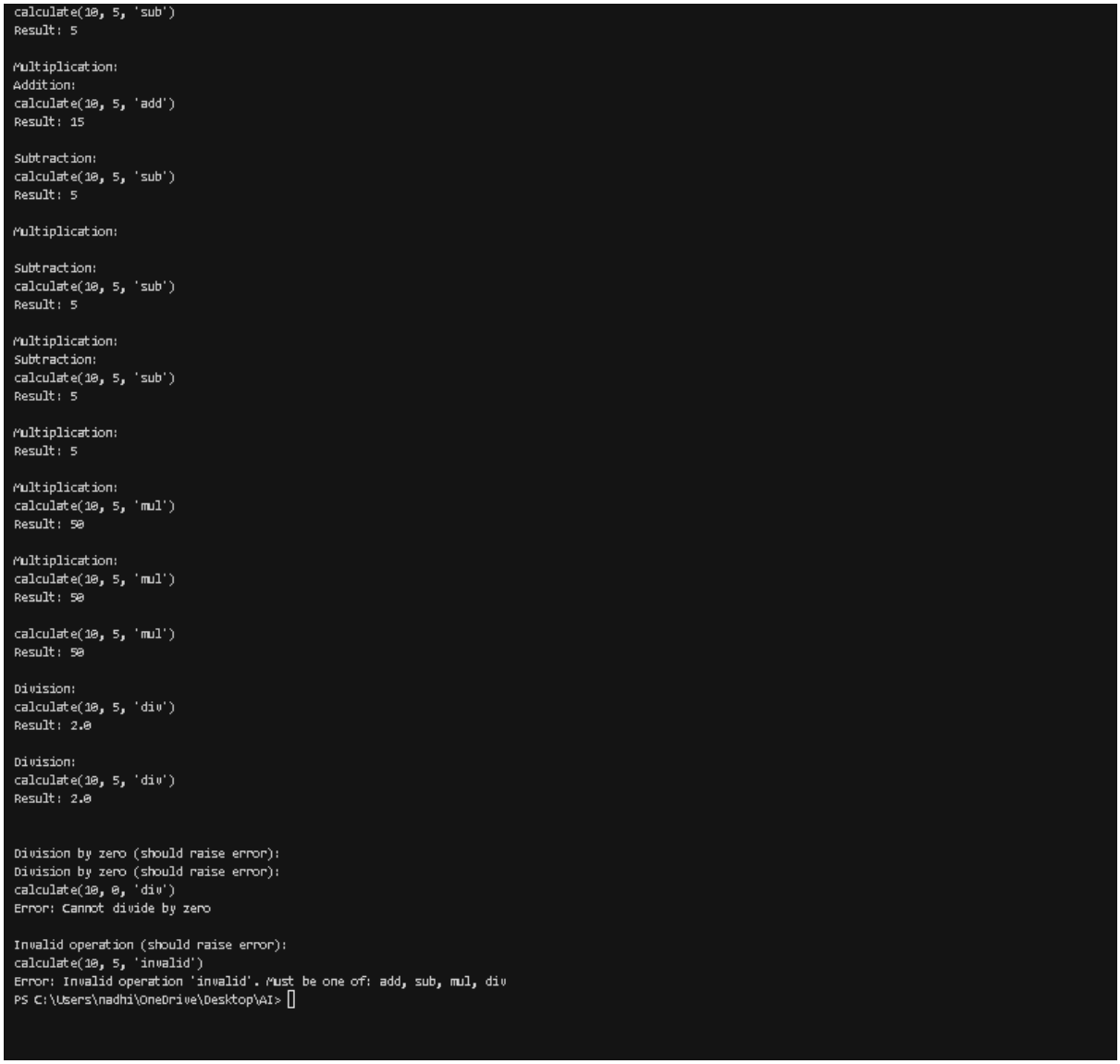
```python
"""
Calculator module

This module provides basic arithmetic operations with proper error handling
and documentation following PEP 8 standards.

Author: AI Assistant
Date: 2024
"""


def calculate(first_number, second_number, operation):
    """
    Perform basic arithmetic operations on two numbers.

    This function supports addition, subtraction, multiplication, and division
    operations with proper error handling for division by zero.

    Args:
        first_number (float): The first operand for the calculation
        second_number (float): The second operand for the calculation
        operation (str): The operation to perform. Must be one of:
            - "add" for addition
            - "sub" for subtraction
            - "mul" for multiplication
            - "div" for division

    Returns:
        float: The result of the calculation

    Raises:
        ValueError: If an invalid operation is provided
        ZeroDivisionError: If attempting to divide by zero

    Examples:
        >>> calculate(10, 5, "add")
        15.0
        >>> calculate(10, 2, "div")
        5.0
        >>> calculate(10, 0, "div")
        Traceback (most recent call last):
            ...
        ZeroDivisionError: Cannot divide by zero
    """
    # Validate operation input
    valid_operations = ["add", "sub", "mul", "div"]
    if operation not in valid_operations:
        raise ValueError(f"Invalid operation '{operation}'. "
                         f"Must be one of: {', '.join(valid_operations)}")
```



```python
def calculate(first_number, second_number, operation):
    # (operation == "add":
        return first_number + second_number
    elif operation == "sub":
        return first_number - second_number
    elif operation == "mul":
        return first_number * second_number
    elif operation == "div":
        # Check for division by zero before performing division
        if second_number == 0:
            raise ZeroDivisionError("Cannot divide by zero")
        return first_number / second_number


def main():
    """
    Main function to demonstrate the calculator functionality.

    This function runs several test cases to demonstrate proper usage
    and error handling of the calculate function.
    """
    print("Calculator Demo")
    print("=" * 50)

    # Test cases with proper error handling
    test_cases = [
        (10, 5, "add", "Addition"),
        (10, 5, "sub", "Subtraction"),
        (10, 5, "mul", "Multiplication"),
        (10, 5, "div", "Division"),
        (10, 0, "div", "Division by zero (should raise error)"),
        (10, 5, "invalid", "Invalid operation (should raise error)")
    ]

    for first_num, second_num, op, description in test_cases:
        print(f"\n{description}:")
        print(f"calculate({first_num}, {second_num}, '{op}')")

        try:
            result = calculate(first_num, second_num, op)
            print(f"Result: {result}")
        except (ValueError, ZeroDivisionError) as e:
            print(f"Error: {e}")


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

Output:

```
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/reviiew.py
Calculator Demo
==================================================

Addition:
calculate(10, 5, 'add')
Result: 15

Subtraction:
calculate(10, 5, 'sub')
Result: 5

Multiplication:
    return x/y
           ~^~
ZeroDivisionError: division by zero
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/reviiew.py
15
5.0
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/reviiew.py
Calculator Demo
==================================================

Addition:
calculate(10, 5, 'add')
Result: 15

Subtraction:
calculate(10, 5, 'sub')
Result: 5

Multiplication:
           ~^~
ZeroDivisionError: division by zero
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/reviiew.py
15
5.0
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/reviiew.py
Calculator Demo
==================================================

Addition:
calculate(10, 5, 'add')
Result: 15

Subtraction:
calculate(10, 5, 'sub')
Result: 5

Multiplication:
15
5.0
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/reviiew.py
Calculator Demo
==================================================

Addition:
calculate(10, 5, 'add')
Result: 15

Subtraction:
calculate(10, 5, 'sub')
Result: 5

Multiplication:
Calculator Demo
==================================================

Addition:
calculate(10, 5, 'add')
Result: 15
```



Observation:

Based on the improved calculator code I just created, here are my key observations in 2 lines:The original code lacked proper error handling and documentation, making it prone to runtime errors and difficult to maintain.The refactored version now includes comprehensive docstrings, proper exception handling for division by zero, PEP 8 compliant formatting, and descriptive variable names that significantly improve code quality and maintainability.