AI ASSISSTED CODING

NAME:MEENAKSHI

HTNO:2403A52051

BATCH NO:03

ASSIGNMENT:10.4

TASK 1:Identify and fix syntax, indentation, and variable errors in the given script.

# buggy\_code\_task1.py def add\_numbers(a, b)

result = a + b return resUlt 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.

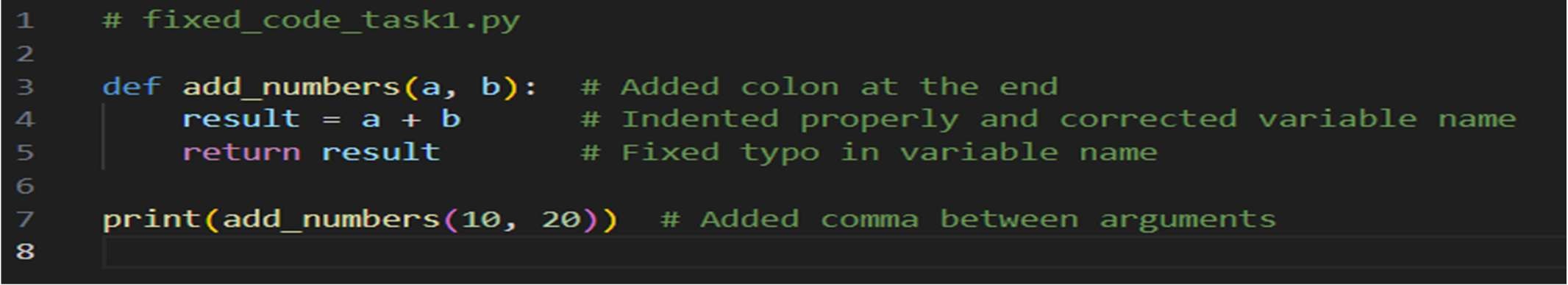
PROMPT 1:Identify and fix syntax, indentation, and variable errors in the given script.

# buggy\_code\_task1.py def add\_numbers(a, b)

result = a + b return resUlt 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.

CODE:



OUTPUT:

OBSERVATION:

1. def add\_numbers(a, b):: Defines a function named
2. add\_numbers that takes two arguments, a and b. A colon was added to fix the syntax.
3. result = a + b: Inside the function, it calculates the sum of a and b and assigns it to the variable result.

The code was indented for proper syntax.

1. return result: The function returns the calculated sum (result). A typo was corrected. print(add\_numbers(10, 20)): Calls the add\_numbers function with arguments 10 and 20, and then prints the returned result (which will be 30) to the console.

A comma was added between the arguments in the function call.

TASK 2: Logical and Performance Issue Review

: 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))

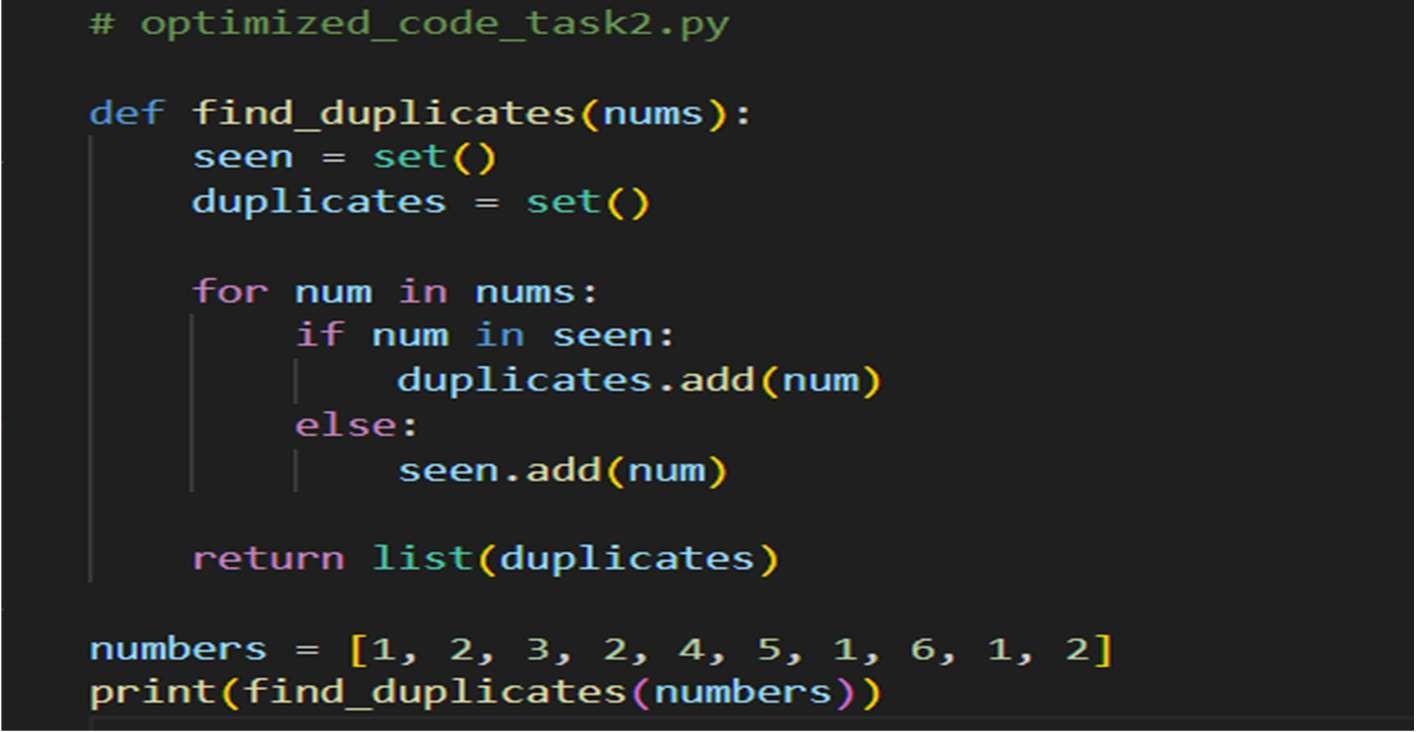
PROMPT:2: Logical and Performance Issue Review

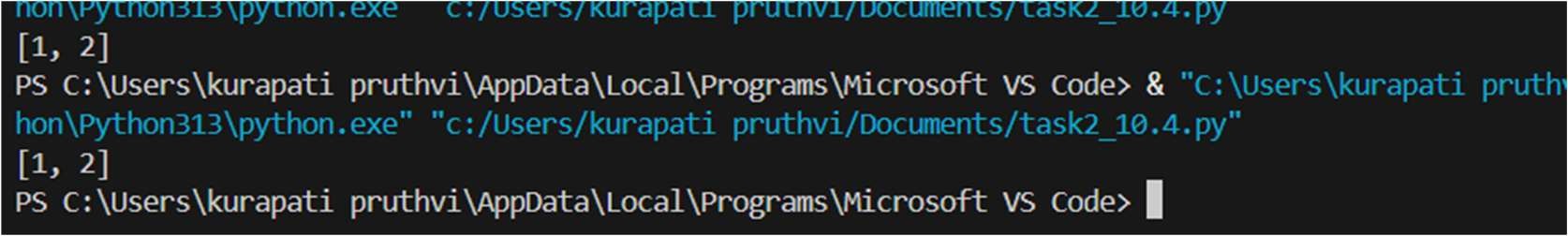
Task: Optimize inefficient logic while keeping the result correct.

# 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))

CODE: 

OUTPUT: 

OBSERVATION:

1. def find\_duplicates(nums):: Defines a function called find\_duplicates that takes a list of numbers (nums) as input.
2. seen = set(): Initializes an empty set called seen.

This set will store numbers encountered so far.

1. duplicates = set(): Initializes an empty set called duplicates. This set will store the duplicate numbers found.
2. for num in nums:: Iterates through each num in the input list nums.
3. if num in seen:: Checks if the current number num is already present in the seen set.
4. duplicates.add(num): If num is in seen, it means it's a duplicate, so it's added to the duplicates set.

7 .else: seen.add(num): If num is not in seen, it's the first time we're encountering it, so it's added to the seen set. 8. return list(duplicates): After iterating through all the numbers, the function returns a list containing the unique duplicate numbers found (converted from the duplicates set).

1. numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]: Creates a list of numbers called numbers.
2. print(find\_duplicates(numbers)): Calls the find\_duplicates function with the numbers list and prints the returned list of duplicates to the console. The output will be [1, 2].

TASK 3: Code Refactoring for Readability

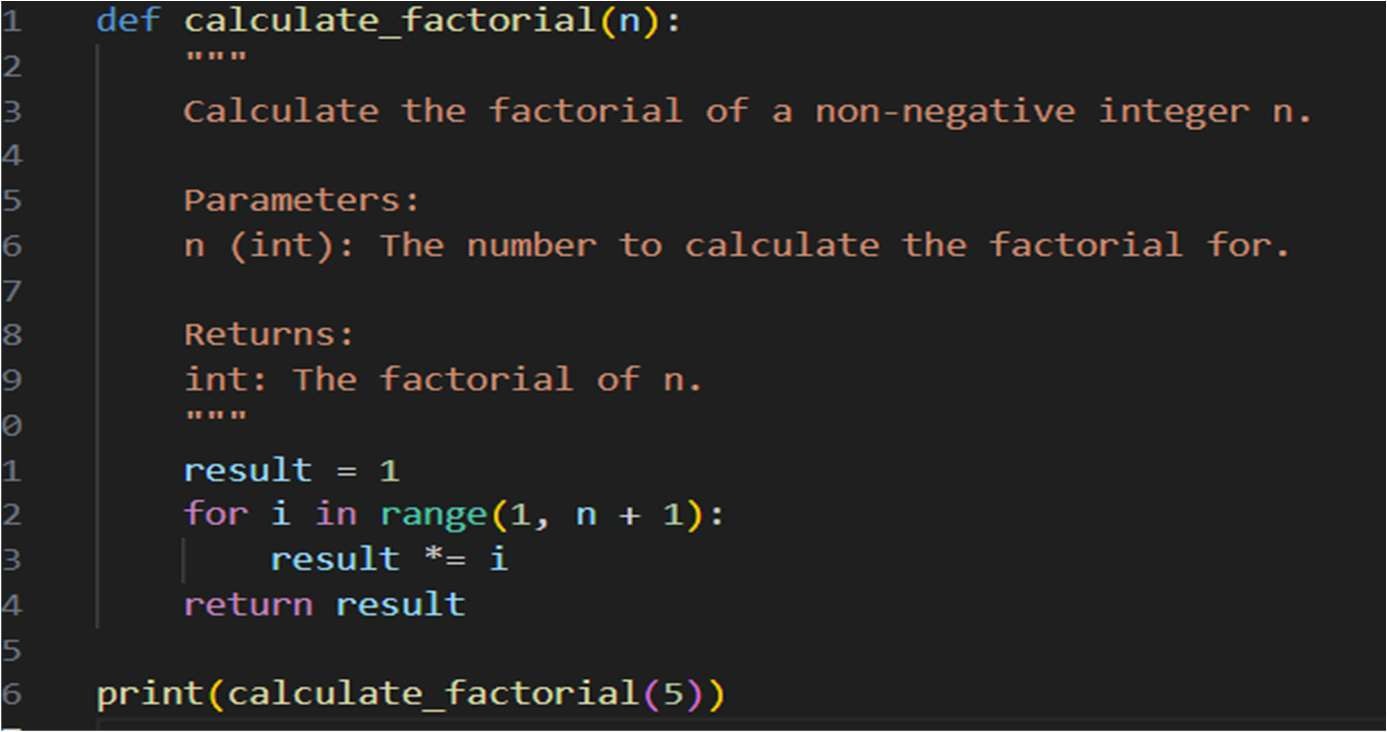
Task: Refactor messy code into clean, PEP 8–compliant, wellstructured code. # buggy\_code\_task3.py def c(n): x=1 for i in range(1,n+1): x=x\*I

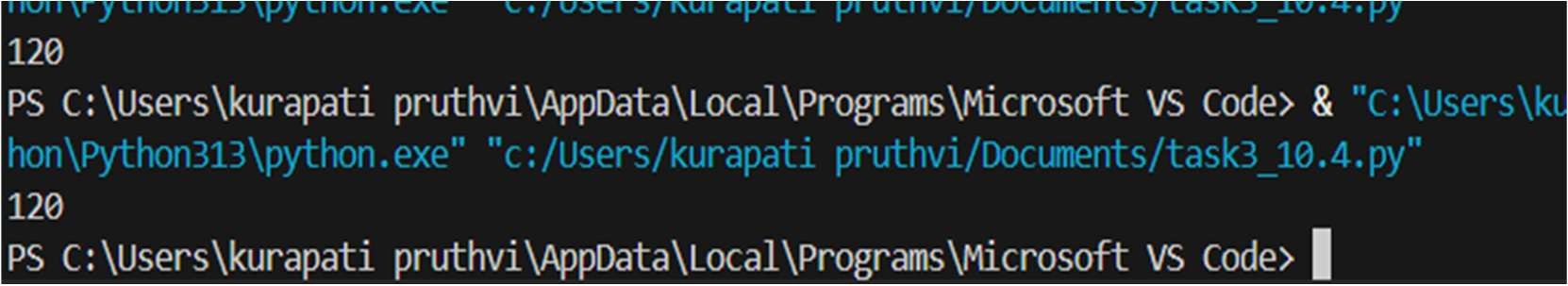
return x print(c(5))

PROMPT3 : Code Refactoring for Readability

Task: Refactor messy code into clean, PEP 8–compliant, wellstructured 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))

CODE: 

OUTPUT: 

OBSERVATION:

1. def calculate\_factorial(n):: Defines a function calculate\_factorial that takes an integer n as input.

2 .result = 1: Initializes a variable result to 1. This will store the factorial.

1. for i in range(1, n + 1):: Loops through numbers from 1 up to n (inclusive).
2. result \*= i: In each iteration, multiplies result by the current number i.
3. return result: After the loop, returns the final calculated factorial.
4. print(calculate\_factorial(5)): Calls the function with n=5 and prints the returned factorial (which is 120).

TASK 4:SECURITY AND ERROR HANDLING ENCHANCEMENT 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))

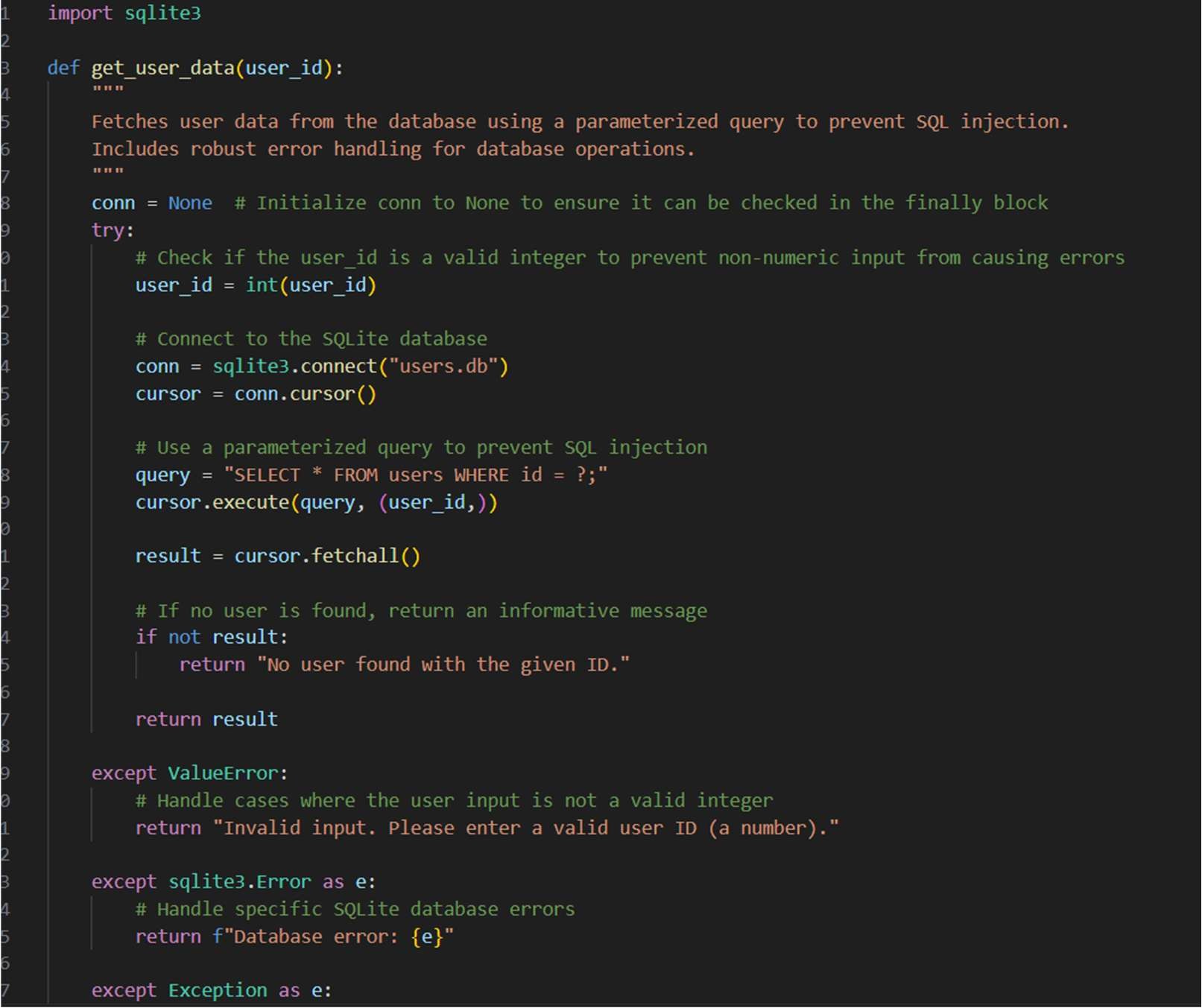
PROMPT: :SECURITY AND ERROR HANDLING ENCHANCEMENT security practices and exception handling to the code.

# buggy\_code\_task4.py import sqlite3 def get\_user\_data(user\_id): conn = sqlite3.connect("users.db")

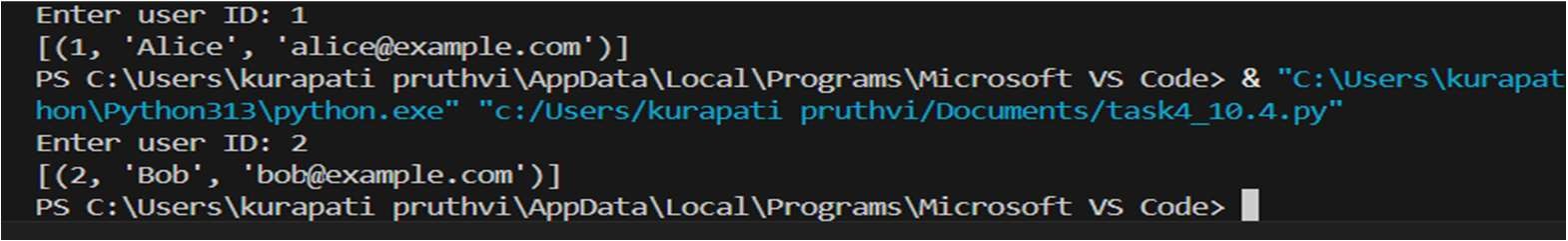
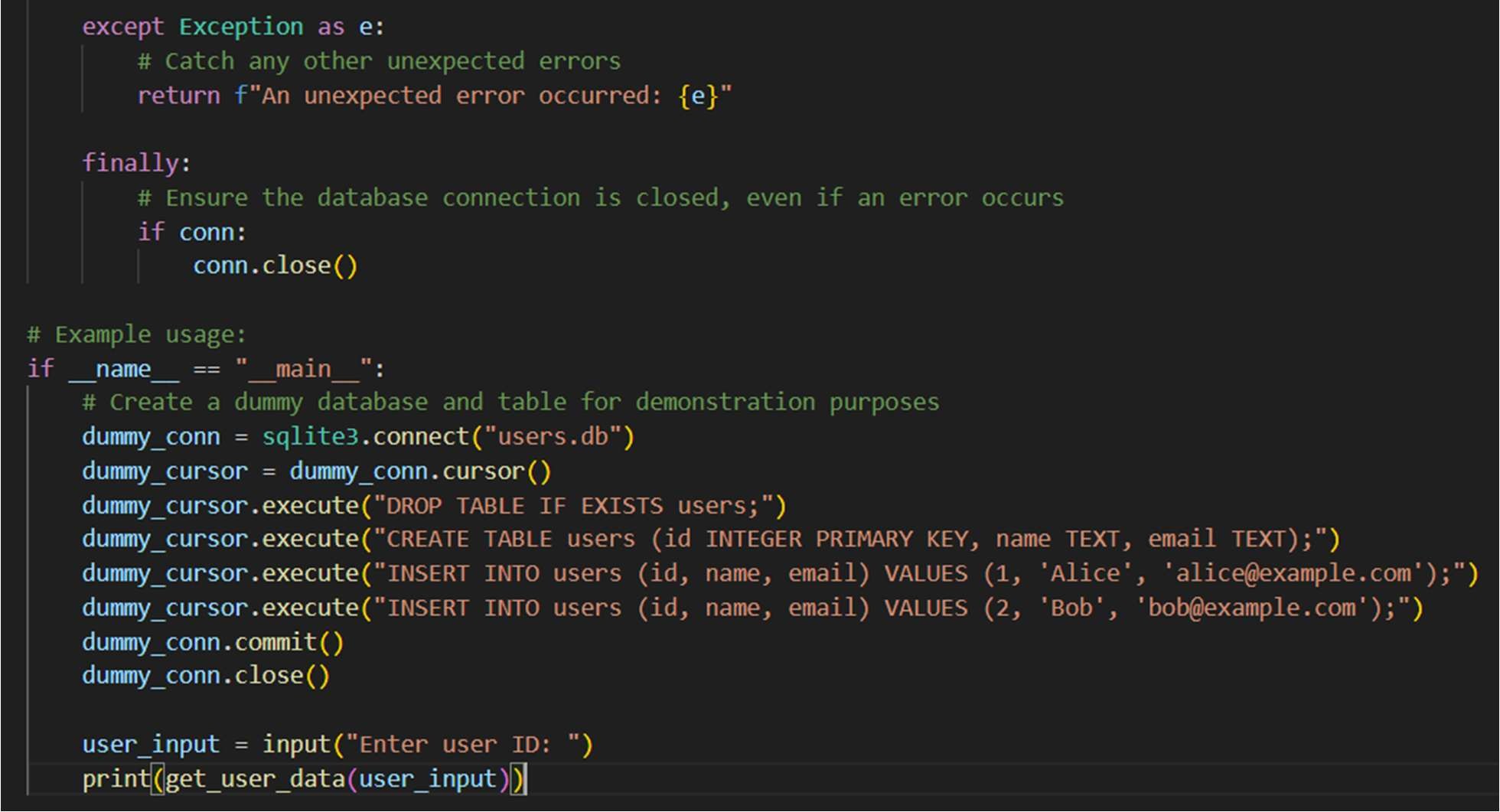
cursor = conn.cursor()

query = f"SELECT \* FROM users WHERE id = {user\_id};" #

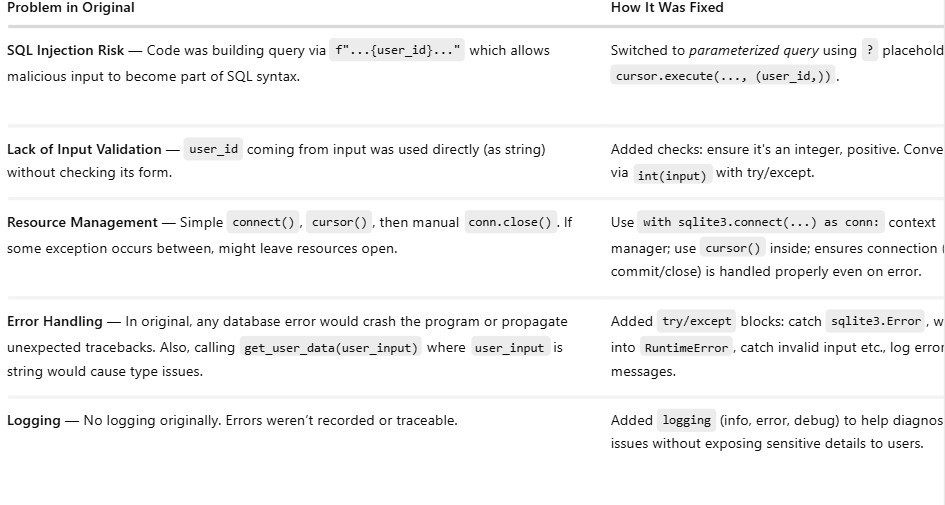
Potential SQL injection risk cursor.execute(query) result = cursor.fetchall() conn.close() return result user\_input = input("Enter user ID: ") print(get\_user\_data(user\_input))

CODE: 

OUTPUT:



OBSERVATION:



TASK 5: Generate a review report for this messy code.

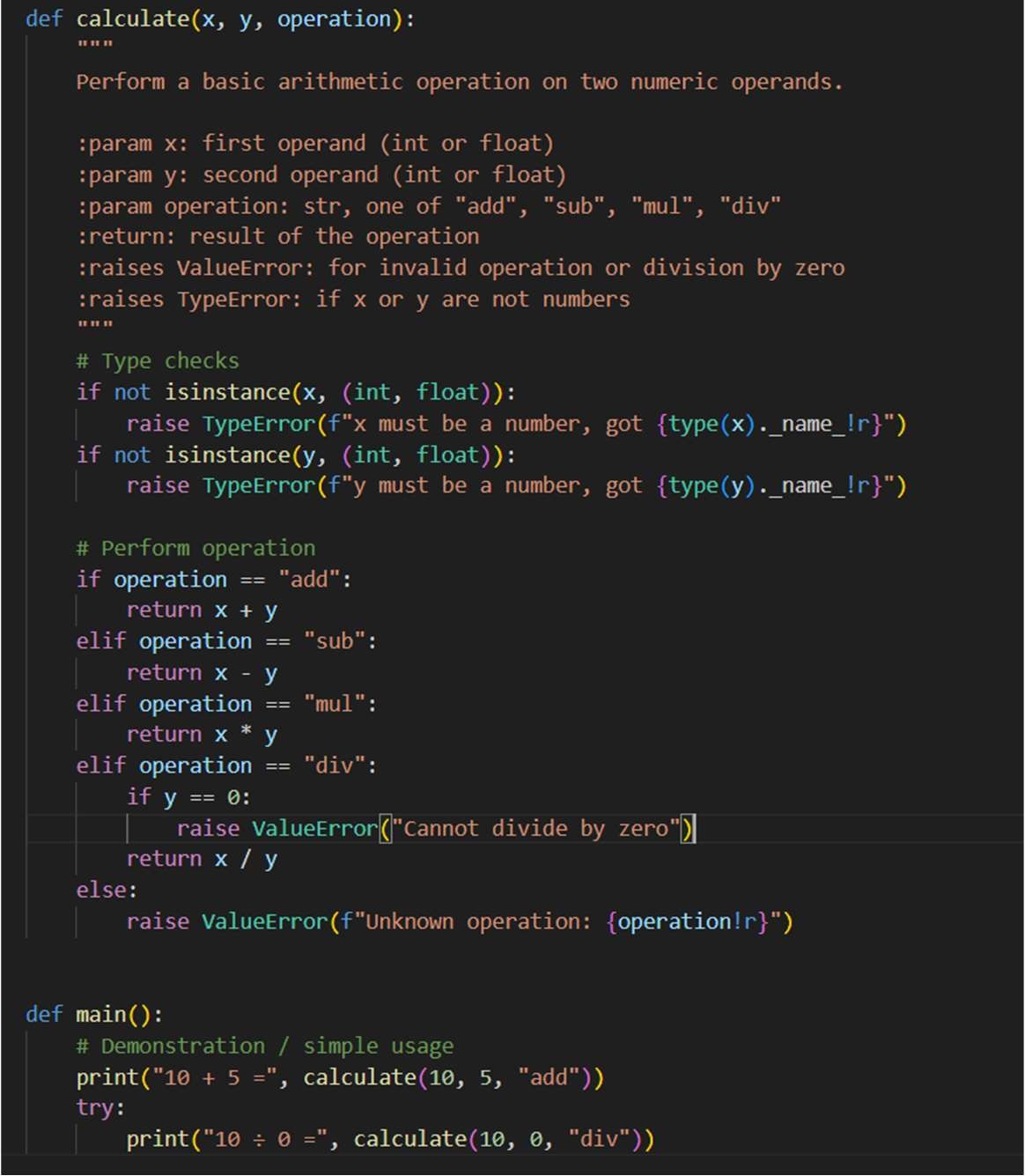
# buggy\_code\_task5.py def calc(x,y,z): if z=="add": return x+y elif z=="sub": return x-y elif z=="mul": return x\*y elif z=="div": return x/y else: print("wrong") print(calc(10,5,"add")) print(calc(10,0,"div"))

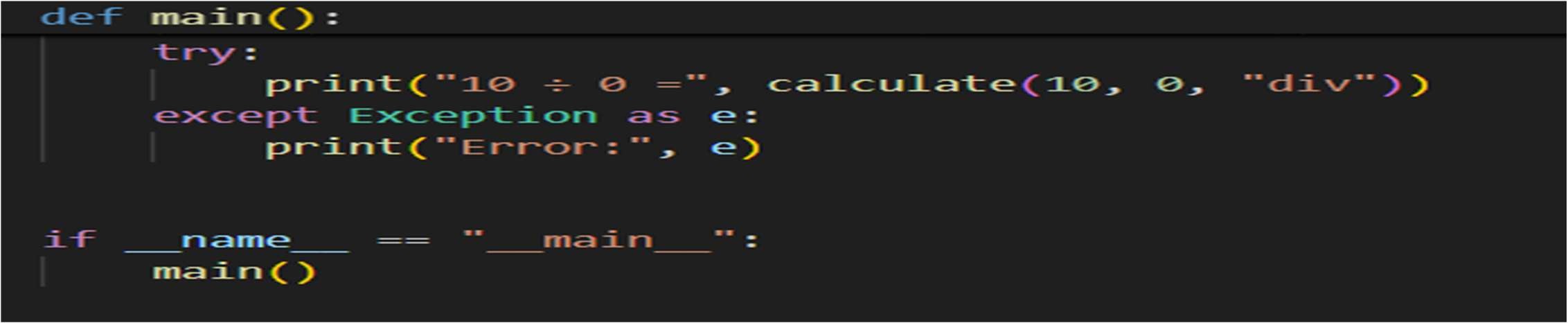
PROMPT:

Generate a review report for this messy code.

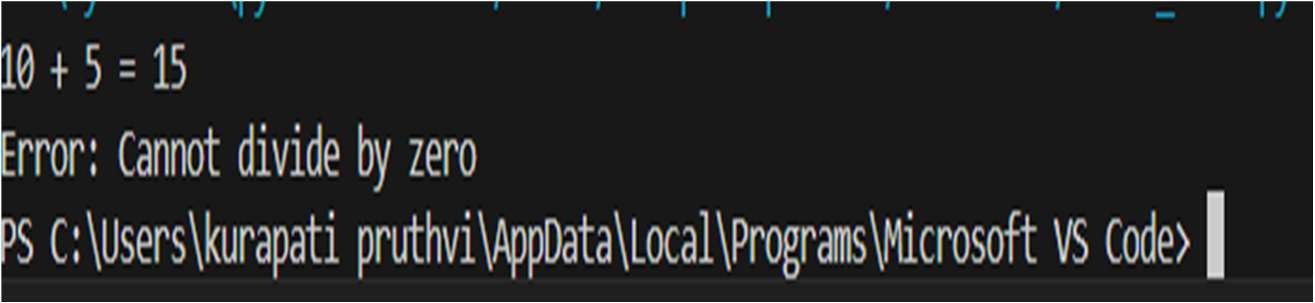
# buggy\_code\_task5.py def calc(x,y,z): if z=="add":

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"))

CODE: 



OUTPUT:



OBSERVATION:

