Assingment-10

Task-1:

**TSyntax and Error Detection**

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

# buggy\_code\_task1.py

def add\_numbers(a, b)

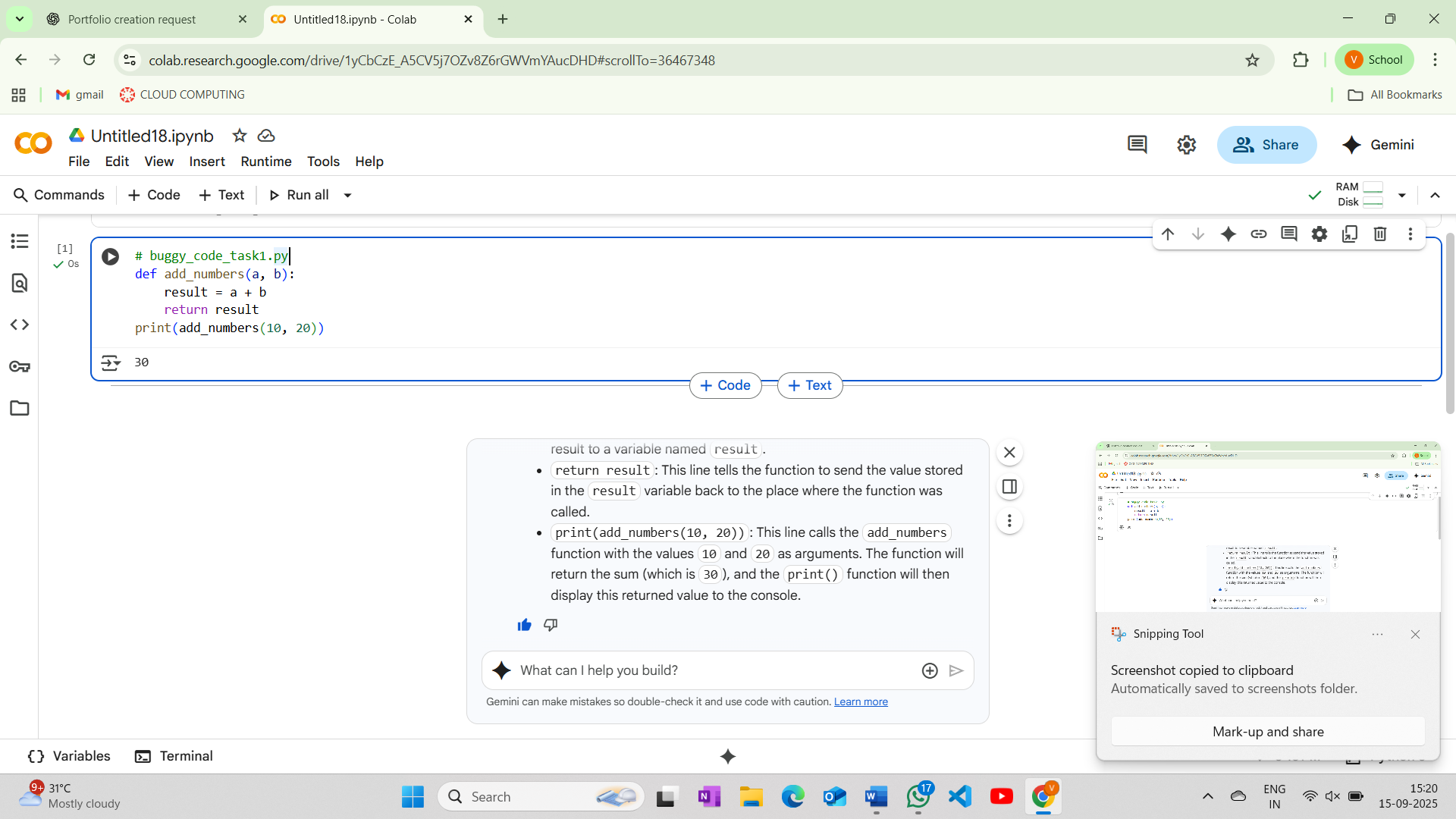
result = a + b

return reslt

print(add\_numbers(10 20))

**Expected Output**:

* Corrected code with proper syntax (: after function, fixed variable name, corrected function call).



Explaination:

* def add\_numbers(a, b):: This line defines a function named add\_numbers that takes two arguments, a and b. The colon at the end of the line is essential syntax for starting a code block (like the function's body).
* result = a + b: Inside the add\_numbers function, this line calculates the sum of the two arguments a and b and assigns the result to a variable named result.
* return result: This line tells the function to send the value stored in the result variable back to the place where the function was called.
* print(add\_numbers(10, 20)): This line calls the add\_numbers function with the values 10 and 20 as arguments. The function will return the sum (which is 30), and the print() function will then display this returned value to the console.

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

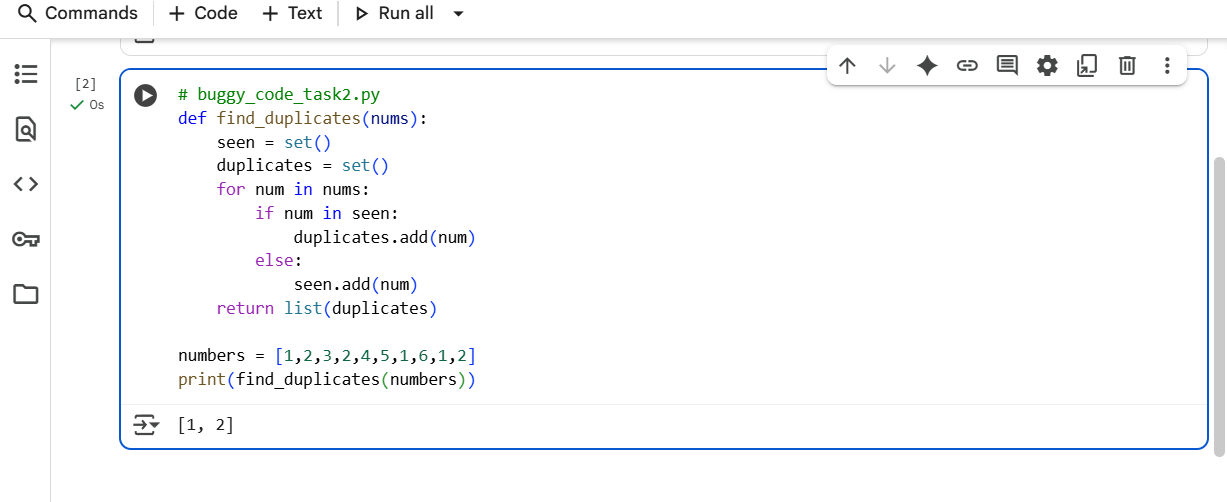
numbers = [1,2,3,2,4,5,1,6,1,2]

print(find\_duplicates(numbers))

**Expected Output**:

* More efficient duplicate detection (e.g., using sets).

AI should explain the optimization



Explaination:

* def find\_duplicates(nums):: This line defines a function called find\_duplicates that takes one argument, a list named nums.
* seen = set(): This line initializes an empty set called seen. A set is an unordered collection of unique elements. We'll use this set to keep track of the numbers we have encountered so far in the list.
* duplicates = set(): This line initializes another empty set called duplicates. This set will store the numbers that appear more than once in the list.
* for num in nums:: This line starts a for loop that iterates through each element in the input list nums. In each iteration, the current element is assigned to the variable num.
* if num in seen:: Inside the loop, this line checks if the current number (num) is already present in the seen set.
* duplicates.add(num): If the if condition is true (meaning the number has been seen before), this line adds the current number to the duplicates set. Since sets only store unique elements, if a number is already in the duplicates set, adding it again has no effect.
* else:: If the if condition is false (meaning the number has not been seen before), the code in the else block is executed.
* seen.add(num): This line adds the current number to the seen set, marking that we have now encountered this number.
* return list(duplicates): After the loop finishes, this line converts the duplicates set into a list using list() and returns this list. Converting to a list is necessary because the original problem likely expects a list as output.
* numbers = [1,2,3,2,4,5,1,6,1,2]: This line creates a list of integers called numbers.
* print(find\_duplicates(numbers)): This line calls the find\_duplicates function with the numbers list as input and prints the returned list of duplicate numbers to the console

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

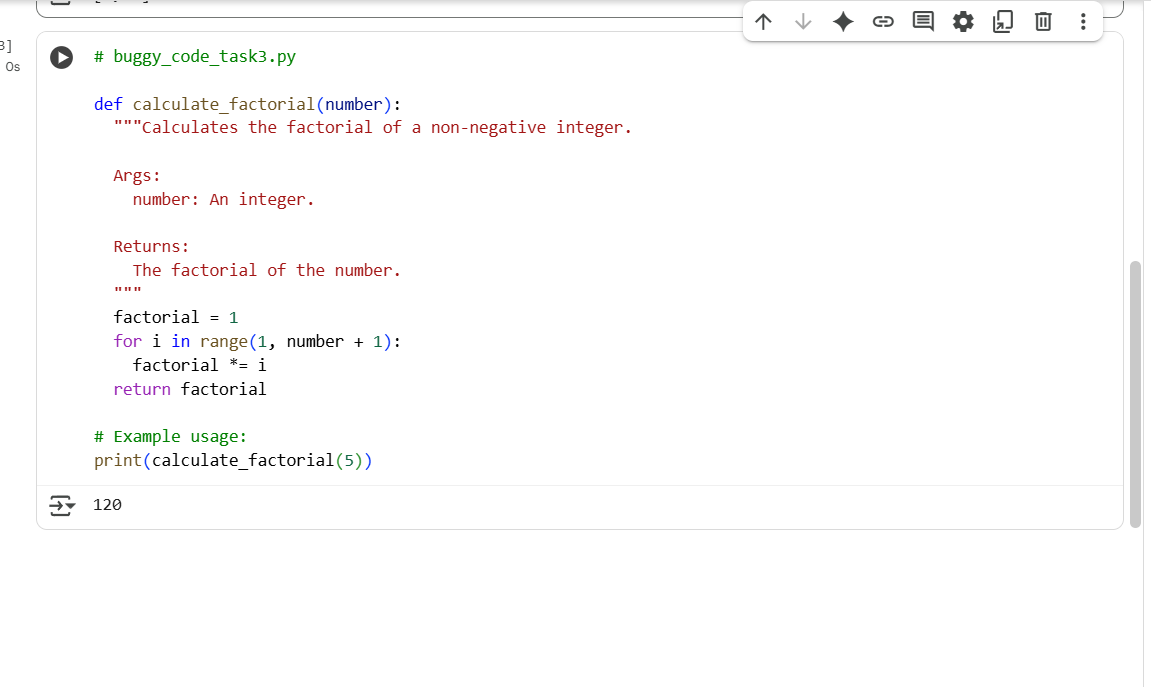
print(c(5))

Expected Output:

Function renamed to calculate\_factorial.

Proper indentation, variable naming, docstrings, and formatting.

AI should provide a more readable version

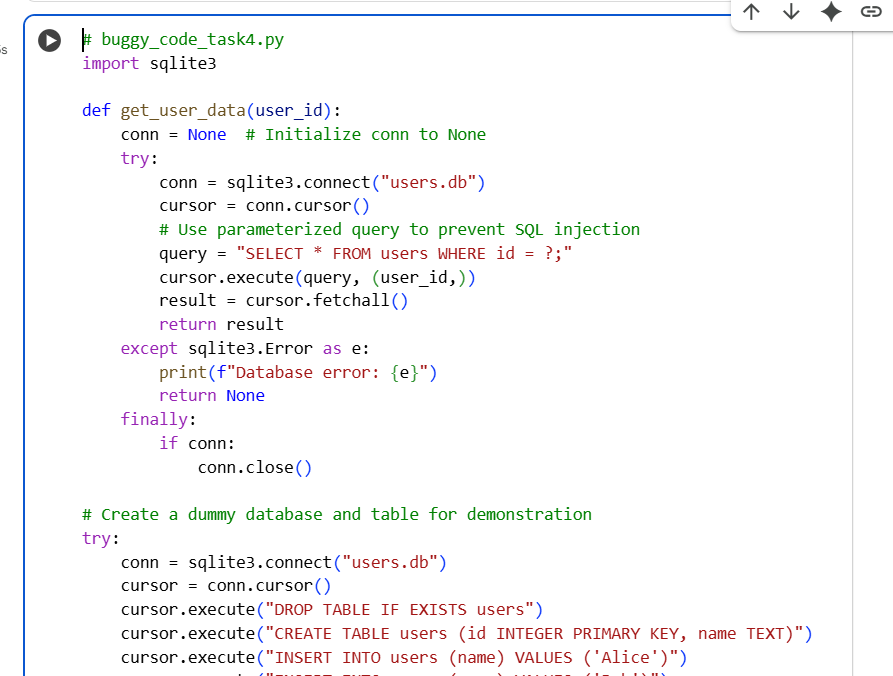


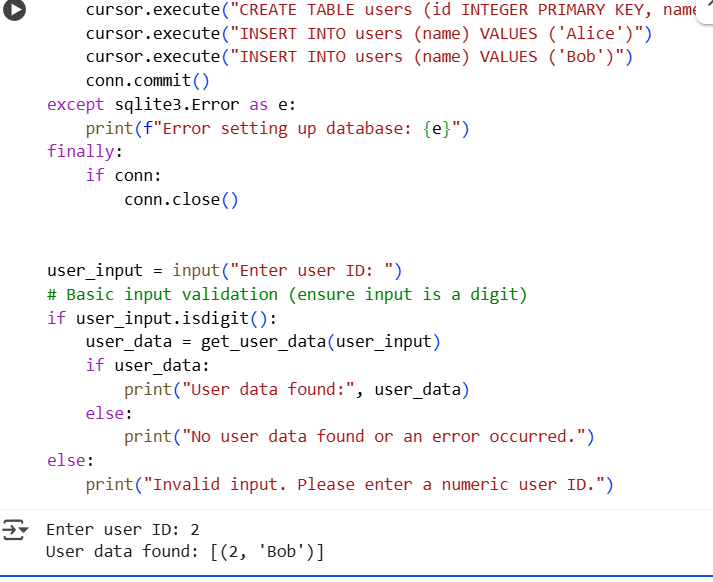
* def calculate\_factorial(number):: This line defines a function named calculate\_factorial that takes one argument, an integer called number. The descriptive name makes the function's purpose clear.
* """Calculates the factorial of a non-negative integer. ... """: This is a docstring. It's a multi-line string used to explain what the function does, its arguments (Args), and what it returns (Returns). Docstrings are important for code documentation and readability.
* factorial = 1: Inside the function, this line initializes a variable named factorial to 1. This variable will store the calculated factorial. We start with 1 because the factorial of 0 is 1, and it serves as the base case for the multiplication in the loop.
* for i in range(1, number + 1):: This line starts a for loop. The range(1, number + 1) function generates a sequence of numbers starting from 1 and going up to and including the value of number. In each iteration, the current number in this sequence is assigned to the variable i.
* factorial \*= i: Inside the loop, this line multiplies the current value of factorial by the current value of i and updates the factorial variable with the result. This is the core of the factorial calculation.
* return factorial: After the loop finishes, this line returns the final calculated value of factorial from the function.
* # Example usage:: This is a comment indicating that the following line demonstrates how to use the function.
* print(calculate\_factorial(5)): This line calls the calculate\_factorial function with the argument 5. The function will calculate the factorial of 5 (which is 5 \* 4 \* 3 \* 2 \* 1 = 120), and the print() function will then display this result to the console.

**Task 4: Security and Error Handling Enhancement**

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





Explaination:

* user\_input = input("Enter user ID: "): Prompts the user to enter a user ID.
* if user\_input.isdigit():: This line adds basic input validation. It checks if the user's input consists only of digits. This prevents non-numeric input from being passed to the database query.
* user\_data = get\_user\_data(user\_input): If the input is valid, this line calls the get\_user\_data function with the user's input.
* if user\_data:: Checks if the get\_user\_data function returned data (not None).
* print("User data found:", user\_data): If data was found, it's printed.
* else: print("No user data found or an error occurred."): If get\_user\_data returned None (due to an error or no user found), this message is printed.
* else: print("Invalid input. Please enter a numeric user ID."): If the initial input validation fails, this message is printed.

In summary, the key enhancements are the use of parameterized queries in the get\_user\_data function to prevent SQL injection and the implementation of a try-except-finally block to handle database errors and ensure the connection is always closed. Basic input validation is also added for better robustness.

**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")

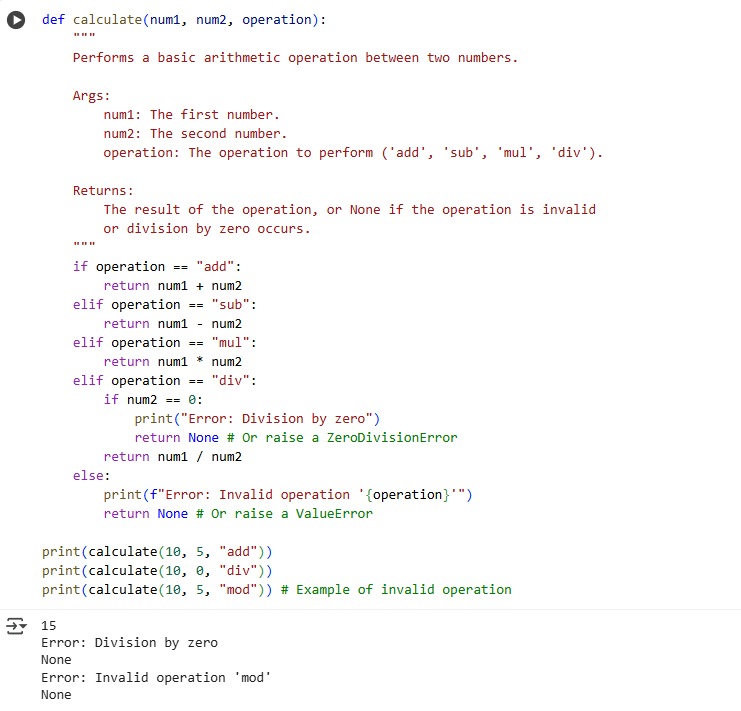
print(calc(10,5,"add"))

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



Explaination:

The code defines a function called calc that takes three arguments: x, y, and z. It performs a calculation based on the value of z:

If z is "add", it returns the sum of x and y.

If z is "sub", it returns the difference between x and y.

If z is "mul", it returns the product of x and y.

If z is "div", it returns the result of dividing x by y.

If z is anything else, it prints "wrong".

The code then calls this function twice:

print(calc(10, 5, "add")) will call calc with x=10, y=5, and z="add", and print the result (15).

print(calc(10, 0, "div")) will call calc with x=10, y=0, and z="div". This will attempt to perform division by zero, which will cause an error.