AI ASSISTED CODING

NAME: B.NANDINI

ENROLL NUMBER: 2403A52390

BATCH NUMBER :14

Lab assignment-10.4

**Task 1: Syntax and Error Detection**

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

# buggy\_code\_task1.py

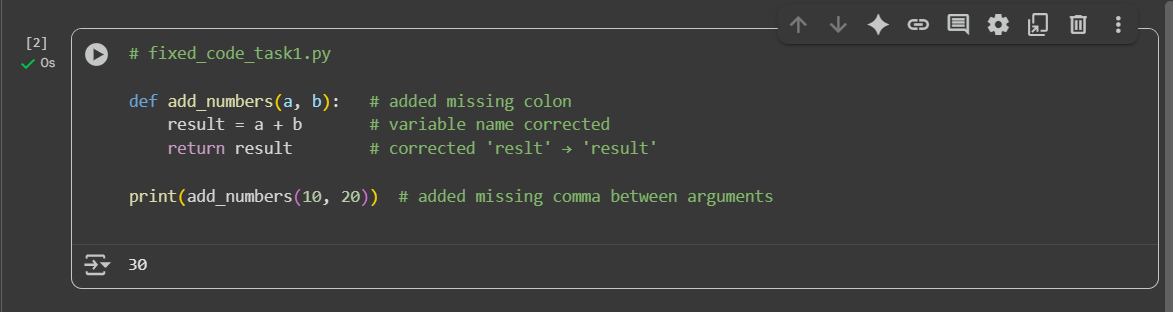
def add\_numbers(a, b)

result = a + b

return reslt

print(add\_numbers(10 20))

code:



Fixes made:

* Added : after def add\_numbers(a, b)
* Corrected variable typo reslt → result
* Added missing comma in print(add\_numbers(10, 20))

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:

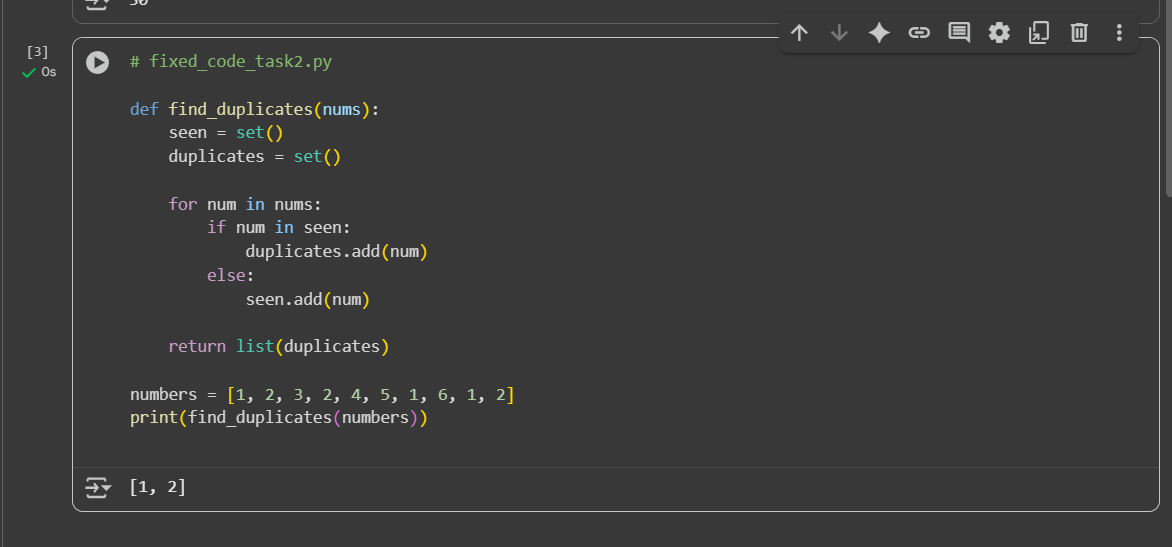
duplicates.append(nums[i])

return duplicates

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

print(find\_duplicates(numbers))

code:



**Optimizations:**

* Replaced nested loops with **single loop**.
* Used **set membership** (O(1)) instead of repeated list lookups (O(n)).
* Collect duplicates in a **set** (avoids duplicates automatically).
* Converted result back to a list at the end.

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

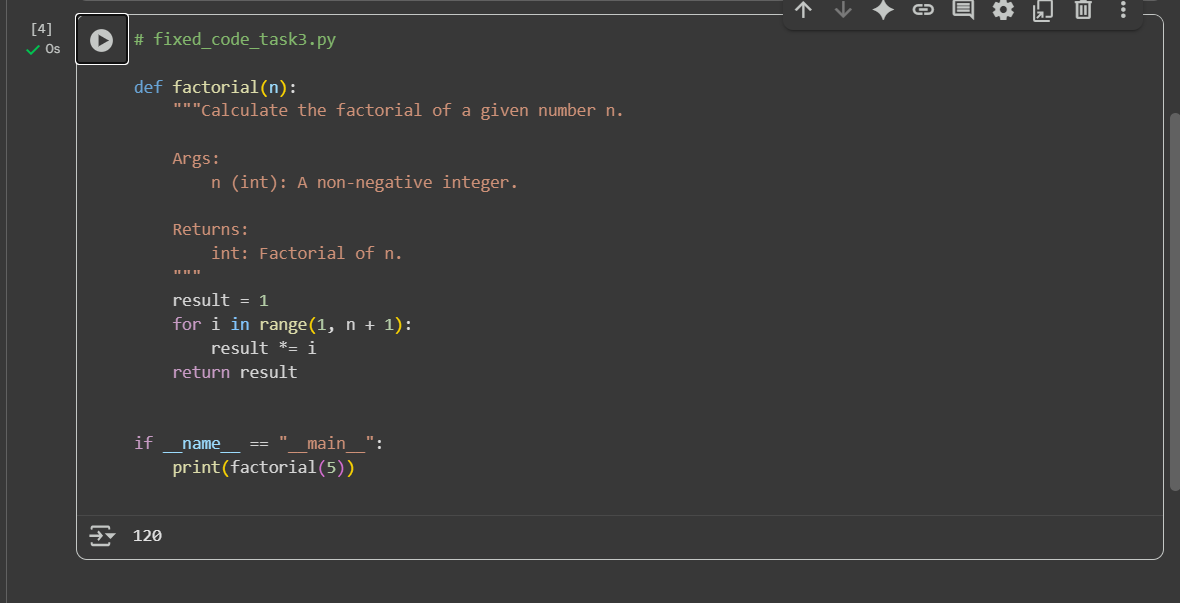
for i in range(1,n+1):

x=x\*i

return x

print(c(5))

code:



**Refactoring improvements:**

* Renamed function c → factorial (clearer).
* Renamed variable x → result.
* Added **docstring** (PEP 257).
* Fixed indentation (4 spaces).
* Wrapped print in if \_\_name\_\_ == "\_\_main\_\_": → better structure for reusability.

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()

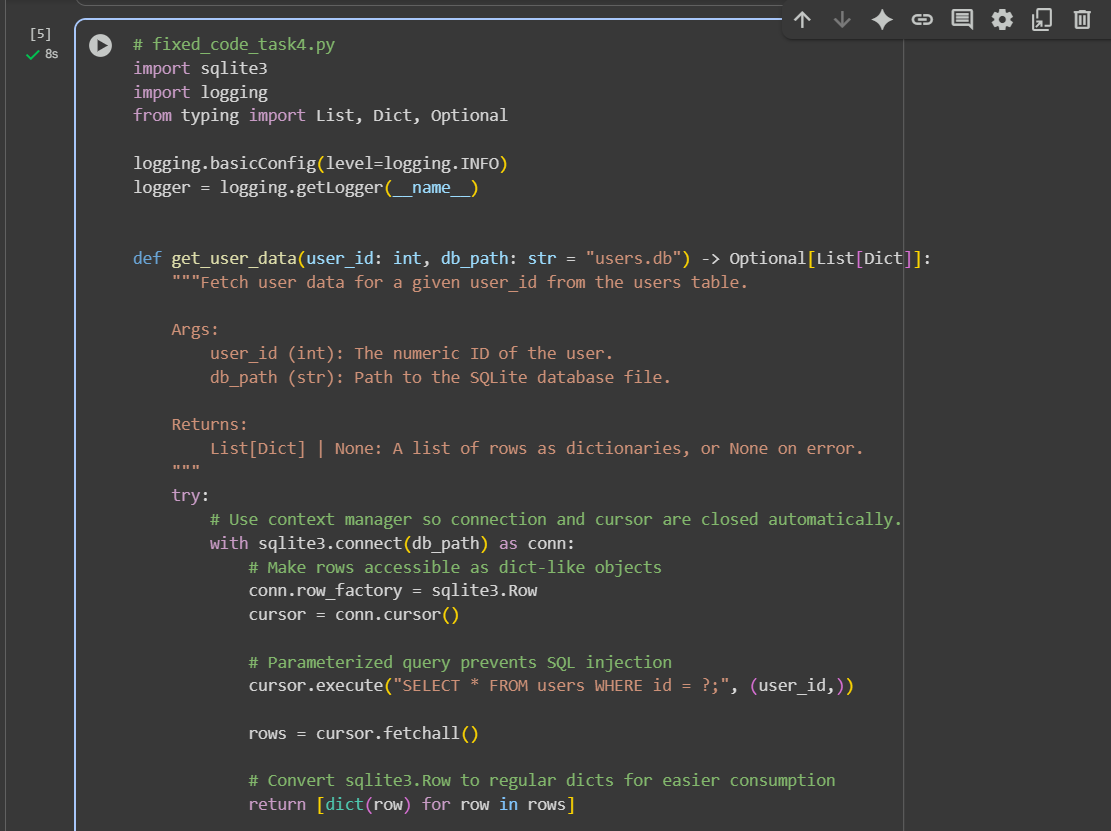
conn.close()

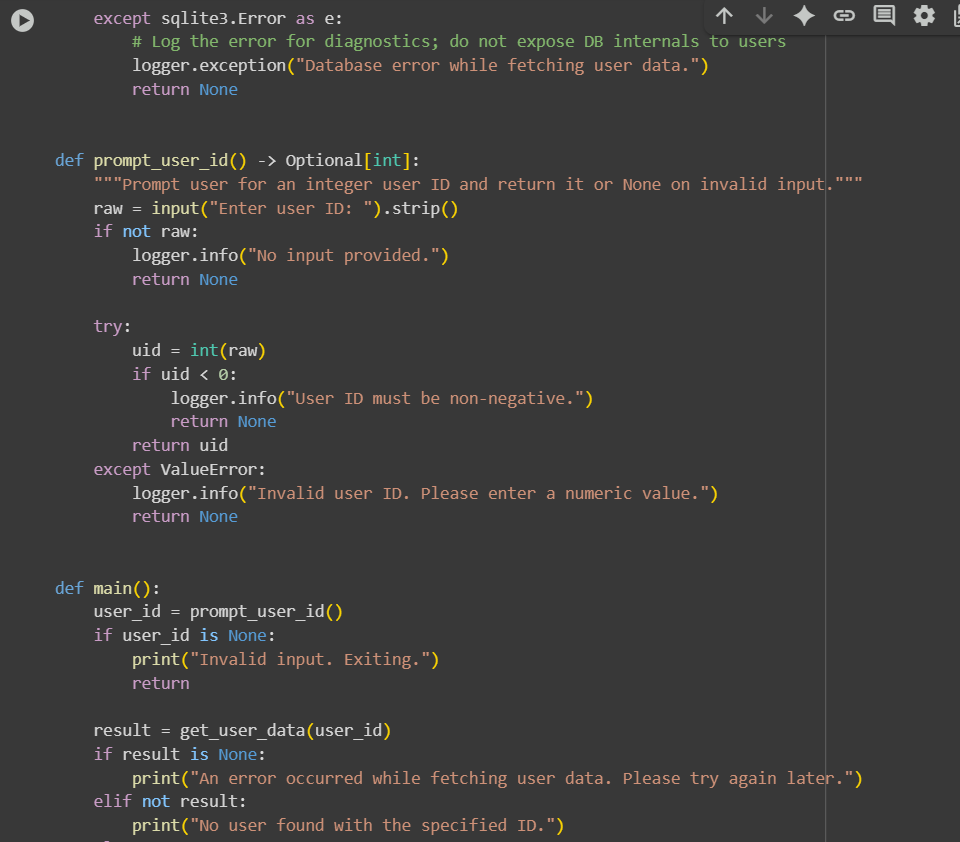
return result

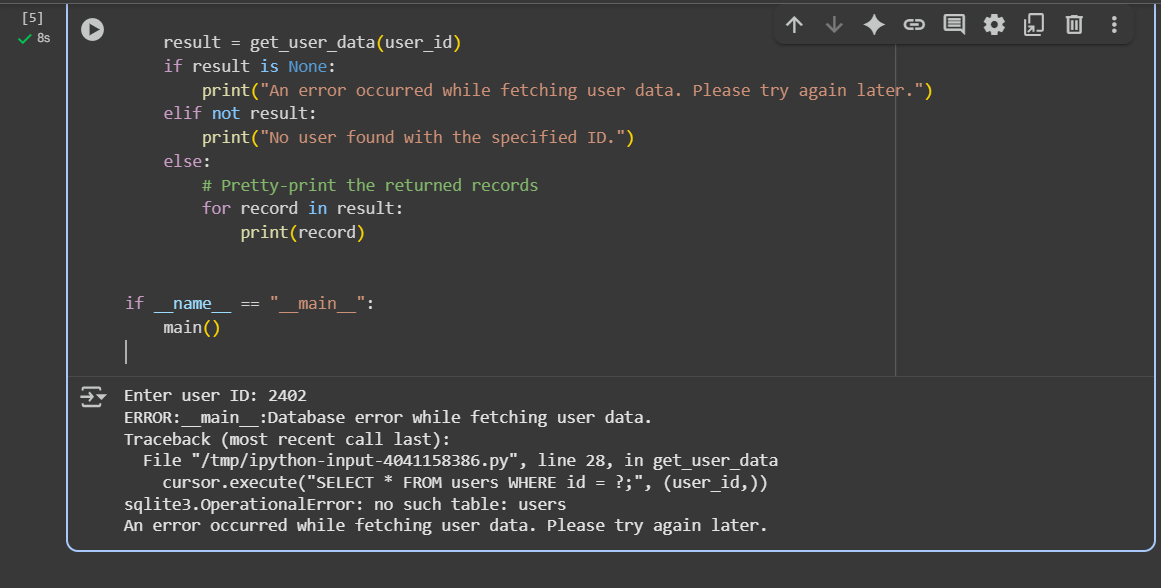
user\_input = input("Enter user ID: ")

print(get\_user\_data(user\_input))

code:







Code explanation:

 Uses **parameterized queries** (prevents SQL injection).

 Validates and converts the user\_id input to an int.

 Uses **context managers** (with) to ensure DB resources are closed.

 Sets row\_factory to sqlite3.Row and returns rows as dictionaries.

 Adds **exception handling** for ValueError and sqlite3.Error.

 Uses logging instead of printing raw exceptions (safer for production).

 Avoids exposing internal error details to the end user

**Notes & further improvements**

* If you expect multiple ways to identify a user (username, email), add separate, strongly validated paths rather than interpolating them into SQL.
* For production systems consider:
  + Storing DB credentials/config separately (not hardcoded).
  + Using an application-level ORM (e.g., SQLAlchemy) for more complex DB logic and safety.
  + More granular logging levels and possibly masking sensitive fields before logging.
* If users.db or the users table doesn't exist yet, this code will handle the error gracefully and log the exception.

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":

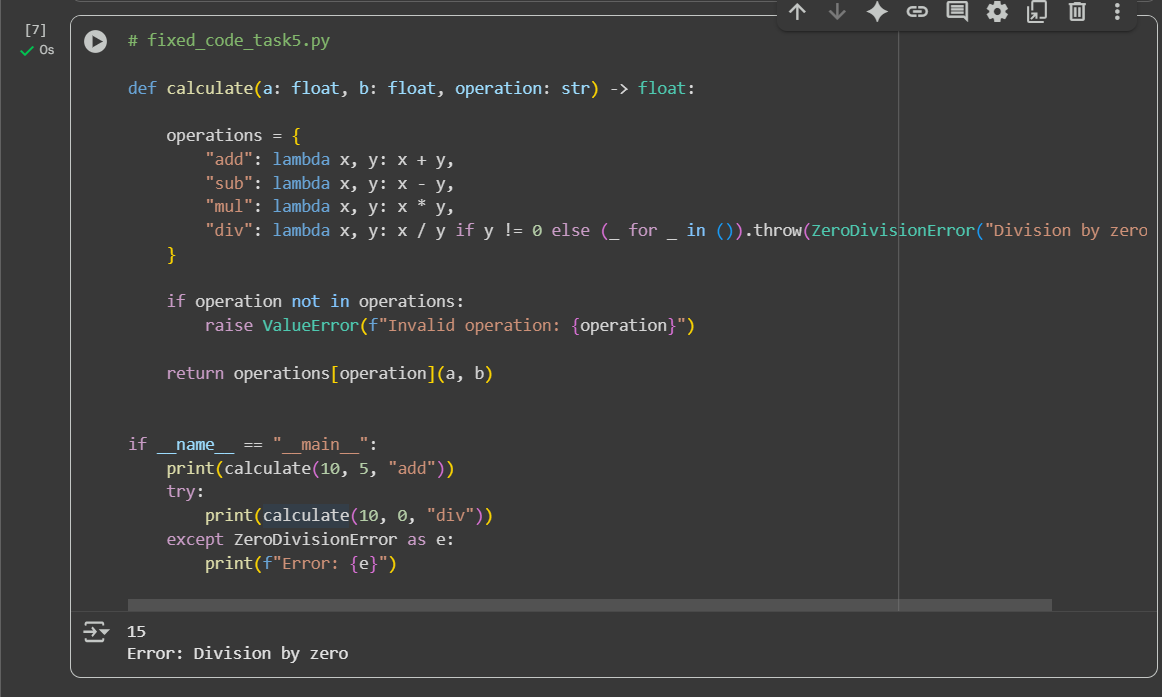
return x/y

else: print("wrong")

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

print(calc(10,0,"div"))

code:



**Code Review Report**

**File:** buggy\_code\_task5.py

**✅ Strengths**

1. **Functionality covered:** Supports four basic arithmetic operations (add, subtract, multiply, divide).
2. **Simple and direct structure:** Easy to follow for small use cases.

**⚠️ Issues Identified**

**1. PEP 8 Non-Compliance**

* Indentation inconsistent (if and elif blocks should be indented 4 spaces).
* Inline statement elif z=="sub": return x-y breaks readability.
* Function and variable names (calc, x, y, z) are not descriptive.

**2. Error Handling**

* Division by zero is not handled → causes ZeroDivisionError.
* Invalid operation (else) prints "wrong" but does not return a proper value (returns None).

**3. Readability & Maintainability**

* No **docstring** explaining the function’s purpose, arguments, or return type.
* Operation names ("add", "sub", etc.) are magic strings → prone to typos.
* Mixing of print and return inside the function → inconsistent interface.

**4. Scalability**

* Every new operation requires another elif.
* Better design would use a **dictionary dispatch pattern** for cleaner extensibility