**Assignment – 10.4**

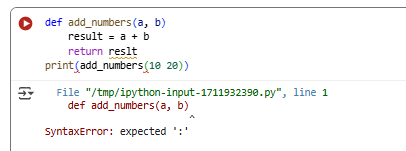
**Name:** Bhavaneesh

**Roll no:** 2403A52350

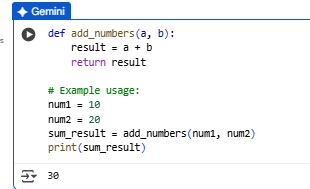
**Batch:** AI 13

**Task 1:** Syntax and Error Detection

**Buggy Code:**



**Corrected Code:**

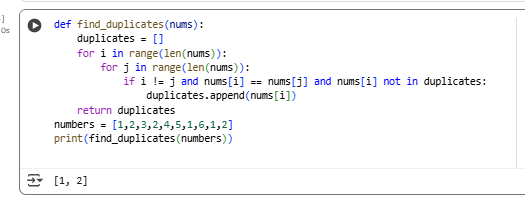


**What are Errors and How to fix it:**

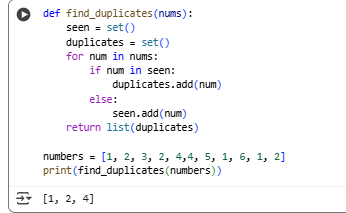
1. **Missing colon after function definition:** The line def add\_numbers(a, b) was missing a colon at the end, which is required syntax in Python to start a code block like a function body.
2. **Typo in the return statement:** The variable result was misspelled as reslt in the return statement. Python is case-sensitive, so reslt was not recognized as the same variable as result.
3. **Missing comma in the print statement:** The original print statement had multiple arguments without being separated by a comma, which is necessary for printing multiple items.

**Task 2: Logical and Performance Issue Review**

**Buggy Code:**



**Redefined Code:**

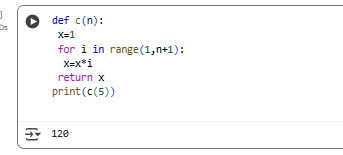


**Explanation:**

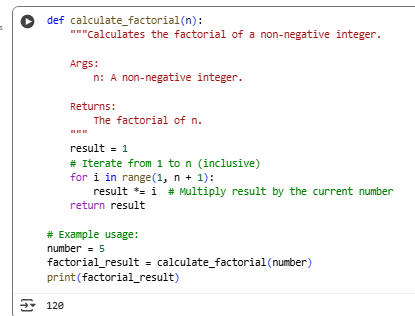
1. **Original Version (using nested loops):**
   1. This version uses nested for loops. The outer loop iterates through each number, and the inner loop compares it to every other number in the list.
   2. It checks if the indices are different (i != j) to avoid comparing a number to itself.
   3. It also checks if the duplicate has already been added to the duplicates list (nums[i] not in duplicates) to avoid adding the same duplicate multiple times.
   4. **Efficiency:** This approach has a time complexity of O(n^2), where 'n' is the number of elements in the list. This means that as the list gets larger, the time it takes to find duplicates increases significantly (quadratically). This is because for each element, you're potentially iterating through the entire list again.
2. **Redefined Version (using sets):**
   1. This version uses two sets: seen and duplicates.
   2. It iterates through the list only once.
   3. For each number, it checks if the number is already in the seen set.
   4. If it is in seen, it means it's a duplicate, so it's added to the duplicates set.
   5. If it's not in seen, it's added to the seen set.
   6. Finally, it converts the duplicates set back into a list for the output.
   7. **Efficiency:** This approach has a time complexity of O(n), where 'n' is the number of elements in the list. This is because checking for membership in a set (using in) is very fast (on average, O(1)). You only iterate through the list once. As the list gets larger, the time it takes to find duplicates increases linearly, which is much more efficient than the quadratic increase of the nested loop approach.

**Task 3: Code Refactoring for Readability**

**Buggy code:**



**PEP 8–compliant, well-structured code.**

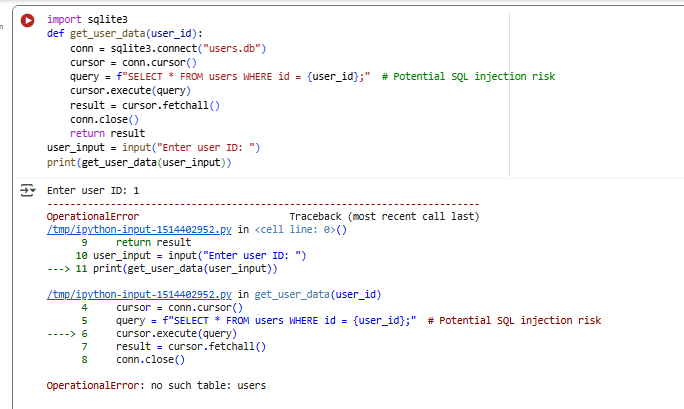


**Explanation:**

1. **Function Name:**
   1. **Original:** def c(n): - The function name c is very short and doesn't clearly indicate what the function does.
   2. **Refactored:** def calculate\_factorial(n): - The name is now descriptive and clearly states the function's purpose, making the code easier to understand at a glance.
2. **Variable Names:**
   1. **Original:** x - The variable x is used for the result, which is not very descriptive.
   2. **Refactored:** result - The variable result is more understandable and indicates what the variable holds.
3. **Spacing and Readability:**
   1. **Original:** The code is very condensed with minimal spacing, making it harder to read.
   2. **Refactored:** Added spaces around operators (=, \*), after commas, and between lines to improve visual clarity and follow PEP 8 guidelines for spacing.
4. **Docstring:**
   1. **Original:** No docstring was present.
   2. **Refactored:** Added a docstring ("""Calculates the factorial...""") to explain what the function does, its arguments, and what it returns. This is crucial for code documentation and understanding.
5. **Comments:**
   1. **Original:** No comments were present to explain the logic.
   2. **Refactored:** Added comments (# Iterate from 1 to n..., # Multiply result by the current number) to clarify the steps within the function.
6. **Example Usage:**
   1. **Original:** The example usage was just print(c(5)).
   2. **Refactored:** Added variables (number, factorial\_result) to make the example usage more explicit and easier to follow.

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

**Buggy Code:**



**Code with efficiency:**



**Explanation:**

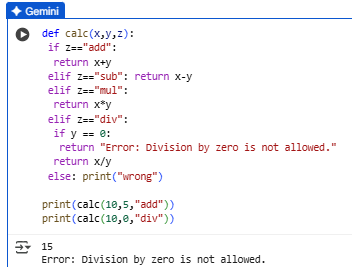
1. **Table Creation:** The original code would fail with an OperationalError: no such table: users if the users table didn't exist in the users.db file. I added the line cursor.execute('''CREATE TABLE IF NOT EXISTS users ...''') to automatically create the table if it's not already there. This makes the code more robust and prevents that specific error.
2. **SQL Injection Prevention:** The most significant improvement is preventing SQL injection. In the original code, you were building the SQL query string by directly inserting the user\_id input. This is dangerous because a malicious user could enter input like ' OR '1'='1 which could potentially allow them to access data they shouldn't see. I changed the query to use a placeholder (?) and passed the user\_id as a separate argument to cursor.execute(). This is called a **parameterized query** and it tells the database to treat the input as a value, not as part of the SQL command itself, effectively neutralizing the SQL injection vulnerability.

**Task 5: Automated Code Review Report Generation**

**Buggy Code:**



**Redefined Code:**



**Explanation:**

The error in the code was a ZeroDivisionError. This happens when you try to divide a number by zero, which is mathematically impossible and causes a runtime error in Python.

The original code attempted to perform the division x / y directly when the operation was "div", without checking the value of y.

To fix this, I modified the code by adding a check specifically for the division case:

elif z=="div":  
 if y == 0:  
 return "Error: Division by zero is not allowed."  
 return x/y

This if y == 0: statement now checks if the denominator y is zero before attempting the division. If y is indeed zero, it returns the string "Error: Division by zero is not allowed." instead of proceeding with the division and causing the error. If y is not zero, the original division return x/y is executed as intended.