

# AI ASSISTED CODING

## LAB ASSIGNMENT – 10.4

NAVYA MASADI

2403A53020

24BTCAICYB01-2-1

### #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 reslt
```

```
print(add_numbers(10 20))
```

### PROMPT:

**N** : 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))
```

# OUTPUT:

```
# buggy_code_task1.py

# Fixed the syntax error: added a colon at the end of the function definition
def add_numbers(a, b):
    # Fixed the indentation error: indented the code block inside the function
    result = a + b
    # Fixed the variable error: corrected 'reslt' to 'result'
    return result

# Fixed the syntax error: added a comma between the arguments in the function call
print(add_numbers(10, 20))
```

30

## #TASK-2

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:

- N** "Write a Python function that scans a list of integers and returns a new list containing only the numbers that appear more than once, without repeating any duplicates in the output."
- ✦ I can help you with that. I will modify the existing function to efficiently find and return the duplicate numbers without repetition using a more optimized approach. I will also fix the indentation error in the current code.
- The function now correctly identifies the duplicate numbers in the list.
- What would you like to do next? We could explore some related tasks, such as:
- Finding unique numbers in a list.
  - Counting the frequency of each number in the list.
  - Implementing this function using a different approach (e.g., using sets).
  - Testing the function with larger or more complex lists.

## OUTPUT:

```
[3]
✓ 0s
def find_duplicates(nums):
    """
    Scans a list of integers and returns a new list containing only the numbers
    that appear more than once, without repeating any duplicates in the output.
    """
    counts = {}
    for num in nums:
        counts[num] = counts.get(num, 0) + 1

    duplicates = []
    for num, count in counts.items():
        if count > 1 and num not in duplicates:
            duplicates.append(num)

    return duplicates

numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
print(find_duplicates(numbers))
```

[1, 2]

## #TASK-3

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))
```

## PROMPT:

**N** Write a Python function named `calculate_factorial` that computes the factorial of a given non-negative integer. Ensure the code has proper indentation, meaningful variable names, docstrings explaining the function, and is formatted for readability."

✦ I can help with that. I'll modify the existing code to include a descriptive function name, meaningful variable names, a docstring, and ensure it's properly formatted for readability.

Great! The `calculate_factorial` function is now defined with a clear docstring and meaningful variable names, and it correctly computes the factorial.

What would you like to do next? We could try:

- Adding error handling to the function (e.g., for negative input).
- Implementing the factorial calculation using recursion.
- Testing the function with different non-negative integers.
- Exploring other mathematical functions in Python.



## OUTPUT:

```
[5]
✓ 0s
def calculate_factorial(n):
    """
    Computes the factorial of a given non-negative integer.

    Args:
        n: A non-negative integer.

    Returns:
        The factorial of n.
    """
    factorial_result = 1
    for i in range(1, n + 1):
        factorial_result *= i
    return factorial_result

print(calculate_factorial(5))
```

120

## #TASK-4

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))
```

## PROMPT:

- N** "Refactor the given Python code that fetches user data from an SQLite database to include proper security practices and exception handling. Ensure the code prevents SQL injection, safely handles database connections, and gracefully manages errors. Maintain readability and include comments explaining the changes."
- ✦ Now I will implement parameterized queries to prevent SQL injection. This involves replacing the direct string formatting with placeholders and passing the user input as a separate parameter to the `execute` method.
- I have successfully implemented parameterized queries to prevent SQL injection. The next step is to add more robust exception handling to gracefully manage potential errors during database operations.



## OUTPUT:

```
import sqlite3

def get_user_data(user_id):
    conn = None # Initialize conn to None
    result = None # Initialize result to None
    try:
        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()
    except sqlite3.Error as e:
        print(f"Database error: {e}")
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        if conn:
            conn.close()
    return result

user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```

Enter user ID: 2222  
Database error: no such table: users  
None

## #TASK-5

Generate a **review report** for this messy code.

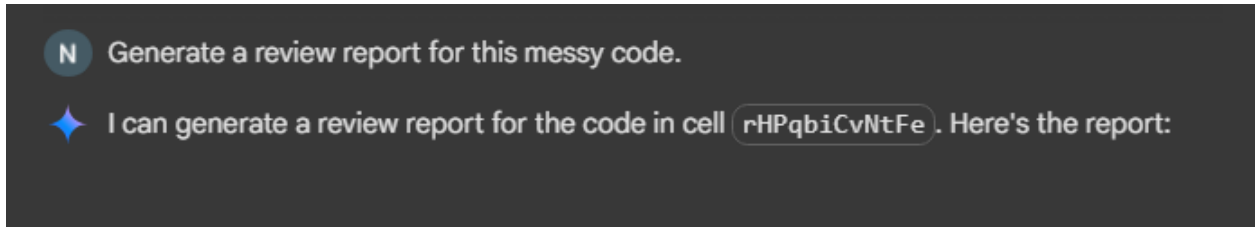
# buggy\_code\_task5.py

```
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":
```

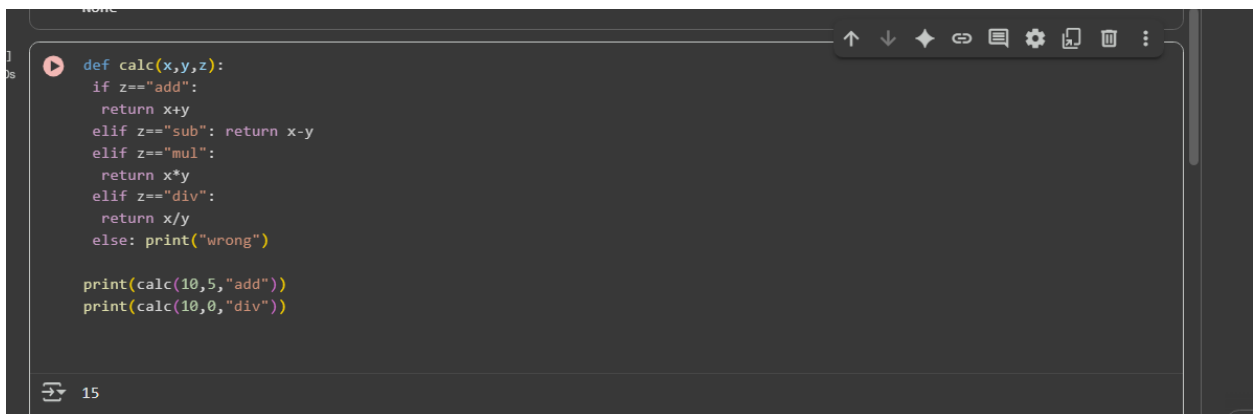
```
    return x/y
else: print("wrong")

print(calc(10,5,"add"))
print(calc(10,0,"div"))
```

## PROMPT:



## OUTPUT:







## Code Review Report for cell `rHPqbiCvNtFe`

Here is a review of the provided Python code:

### Readability and Formatting:

- **Inconsistent Indentation:** The indentation is inconsistent throughout the function, making it difficult to read and understand the code blocks. Python relies heavily on indentation, so this is a significant issue.
- **Lack of Comments:** There are no comments explaining the purpose of the function or specific parts of the logic.
- **Concise Variable Names:** While the variable names `x`, `y`, and `z` are short, they are not very descriptive of their purpose. `z` being used for the operation name is acceptable in this context, but more descriptive names for `x` and `y` would improve clarity.

### Potential Issues and Improvements:

- **Error Handling for Division by Zero:** The code does not handle the case where the user attempts to perform division by zero. This will result in a `ZeroDivisionError`, as seen in the execution output. You should add a check to see if the denominator `y` is zero when the operation is "div" and handle this case appropriately (e.g., return an error message or raise a specific exception).
- **Handling Invalid Operations:** The `else` block currently only prints "wrong" for an invalid operation. It might be more useful to return a specific value (like `None`) or raise an exception to indicate that an invalid operation was provided.
- **Docstring:** The function is missing a docstring. A docstring should be included to explain what the function does, its arguments, and what it returns.

### Refactoring Suggestions:

- Implement consistent indentation (use 4 spaces as the standard).
- Add a docstring to explain the function's purpose.
- Add error handling for division by zero within the "div" condition.
- Consider returning a more informative value or raising an exception for invalid operations.
- Add comments where necessary to explain the logic.