

# AI ASSISTED CODING

## LAB-10.4

NAME:MOHAMMED ANAS SIDDIQUI

ENROLL.NO:2403A52088

BATCH:04

### TASK-01:

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:

Identify the errors in the given code and give the corrected code.

### CODE & OUTPUT:

The screenshot shows a code editor interface with a dark theme. At the top, there is a navigation bar with icons for files like 11.2.3.py, 11.2.4.py, etc., and a tab for 10.4.1.py which is currently active. Below the navigation bar is the code editor area containing the following Python script:

```
10.4.1.py > ...
1 def add_numbers(a, b):
2     """This function adds two numbers and returns the result."""
3     result = a + b
4     return result
5
6 # You can call the function and print the output like this:
7 print(add_numbers(10, 20))
8
```

Below the code editor is a terminal window showing the command line interface. It displays the following text:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/10.4.1.py
30
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

### OBSERVATION:

The ai corrected all the errors in the given code and make the code much more efficient to understand and it also gave the indentation to the code.

### TASK-02:

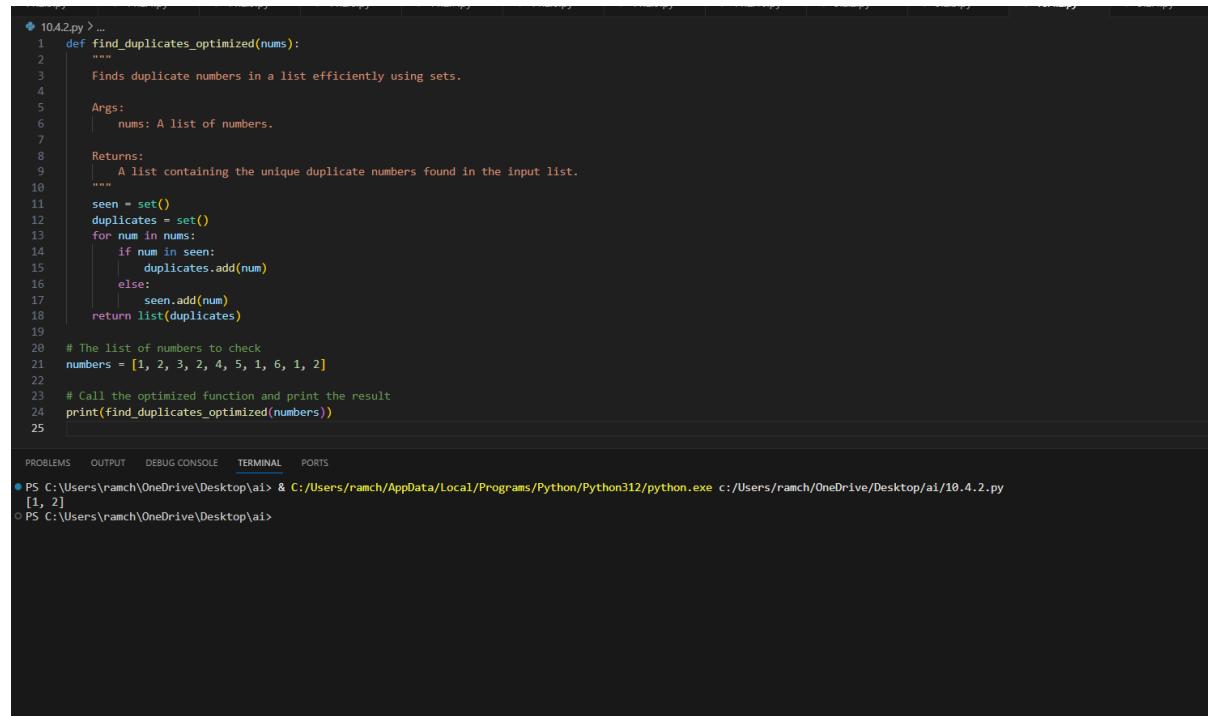
Optimize inefficient logic while keeping the result correct.

```
# 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:

Correct the given code with the proper indentation and give the corrected code.

### CODE & OUTPUT:



The screenshot shows a terminal window with the following content:

```
10.4.2.py > ...
1 def find_duplicates_optimized(nums):
2     """
3         Finds duplicate numbers in a list efficiently using sets.
4
5     Args:
6         nums: A list of numbers.
7
8     Returns:
9         A list containing the unique duplicate numbers found in the input list.
10    """
11    seen = set()
12    duplicates = set()
13    for num in nums:
14        if num in seen:
15            duplicates.add(num)
16        else:
17            seen.add(num)
18    return list(duplicates)
19
20 # The list of numbers to check
21 numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
22
23 # Call the optimized function and print the result
24 print(find_duplicates_optimized(numbers))
25
```

Below the code, the terminal shows the command being run and the output:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/10.4.2.py
[1, 2]
```

### OBSERVATION:

The ai edited the code with proper indentation and also corrected all the errors and the code is finding the duplicate numbers efficiently.

### TASK-03:

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:

rename the function name in the given code and correct code with correcting all the errors and also give the proper indentation.

### CODE & OUTPUT:

The screenshot shows a code editor with multiple tabs at the top, one of which is '10.4.3.py'. The main area contains the following Python code:

```
10.4.3.py > ...
1 def calculate_factorial(n):
2     """
3         Calculates the factorial of a non-negative integer.
4
5     Args:
6         n: A non-negative integer.
7
8     Returns:
9         The factorial of n.
10    """
11    # Initialize the result to 1, as the factorial of 0 is 1.
12    factorial_result = 1
13
14    # Iterate from 1 up to and including n
15    for i in range(1, n + 1):
16        factorial_result = factorial_result * i
17
18    return factorial_result
19
20 # Example usage: Calculate the factorial of 5
21 print(calculate_factorial(5))
22
```

Below the code editor is a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/10.4.3.py
120
○ PS C:\Users\ramch\OneDrive\Desktop\ai>
```

### OBSERVATION:

The code is corrected by the ai according to the prompt given by me in which I have mentioned that update the function name and correct the errors and give the proper indentation.

**TASK-04:**

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:**

Correct the code by applying proper indentations and identify all the errors and correct them give the corrected code.

**CODE & OUTPUT:**

## OBSERVATION:

The code corrects the code by removing all the errors and applies the exception handling to the code to make it more efficient and accurate.

## **TASK-05:**

Generate a review report for this messy code.

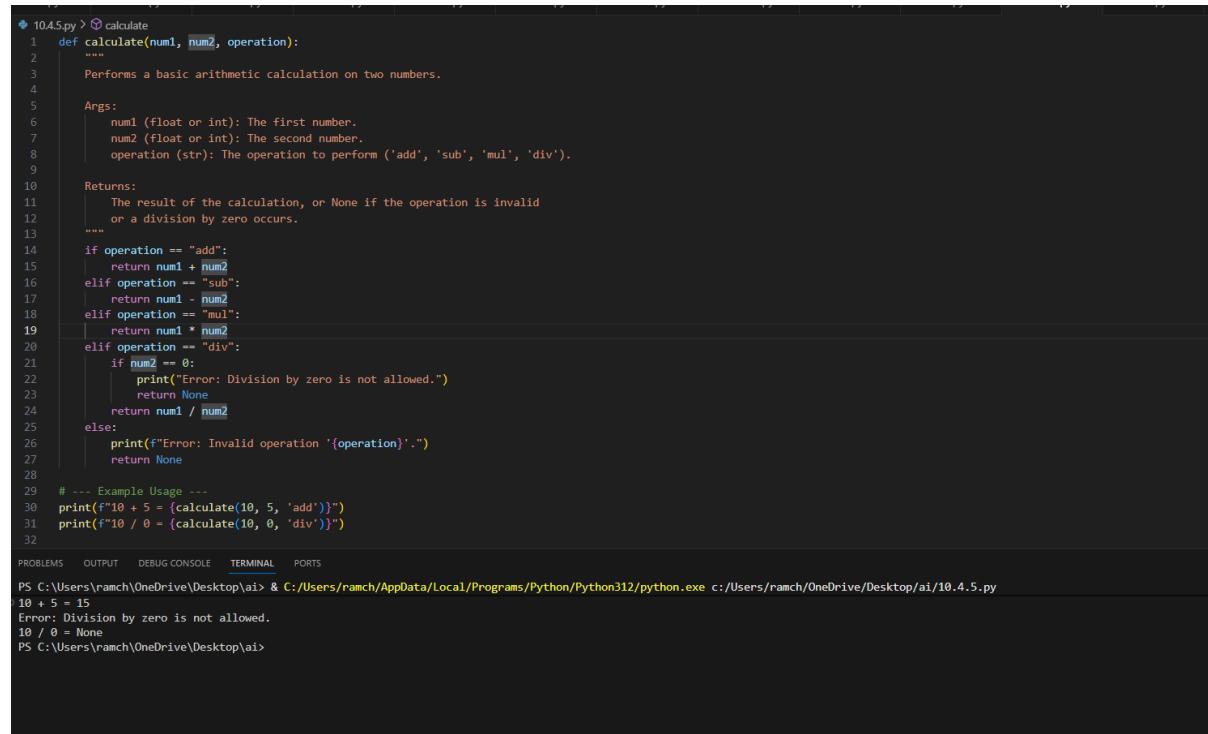
```
# buggy_code_task5.py
```

```
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:**

Correct the code and give the report explaining the code.

**CODE & OUTPUT:**



The screenshot shows a code editor with a dark theme. The code file is named 10.4.5.py. The code has been refactored into a more readable and maintainable form. It includes a docstring and type annotations for the parameters and return value. The code handles division by zero and invalid operations. The terminal output at the bottom shows the execution of the script and its results.

```
10.4.5.py > calculate
1 def calculate(num1, num2, operation):
2     """
3         Performs a basic arithmetic calculation on two numbers.
4
5         Args:
6             num1 (float or int): The first number.
7             num2 (float or int): The second number.
8             operation (str): The operation to perform ('add', 'sub', 'mul', 'div').
9
10        Returns:
11            The result of the calculation, or None if the operation is invalid
12            or a division by zero occurs.
13        """
14        if operation == "add":
15            return num1 + num2
16        elif operation == "sub":
17            return num1 - num2
18        elif operation == "mul":
19            return num1 * num2
20        elif operation == "div":
21            if num2 == 0:
22                print("Error: Division by zero is not allowed.")
23                return None
24            return num1 / num2
25        else:
26            print(f"Error: Invalid operation '{operation}'")
27            return None
28
29 # --- Example Usage ---
30 print(f"10 + 5 = {calculate(10, 5, 'add')}")
31 print(f"10 / 0 = {calculate(10, 0, 'div')}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/10.4.5.py
10 + 5 = 15
Error: Division by zero is not allowed.
10 / 0 = None
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

**EXPLANATION:**

AI-Generated Code Review Report

The provided Python code snippet for a simple calculator has several areas that can be improved to meet standard coding practices. Below is a detailed analysis based on your criteria.

### 1. Missing Docstrings

The function `calc` lacks a docstring. Docstrings are crucial for explaining what a function does, its parameters, and what it returns. This makes the code easier for others (and your future self) to understand and use.

### 2. Inconsistent Formatting and PEP 8 Compliance

The code does not follow standard Python formatting guidelines (PEP 8), which harms readability.

Indentation: The bodies of the `if`, `elif`, and `else` statements are not correctly indented. Python relies on indentation to define code blocks.

Inline return: The line `elif z=="sub": return x-y` has the `return` statement on the same line as the condition. While syntactically allowed, it's inconsistent with the other branches and generally discouraged for readability.

### 3. Missing Error Handling

A critical bug exists in the code. The test case `print(calc(10, 0, "div"))` will cause the program to crash with a `ZeroDivisionError` because it attempts to divide by zero. The function should handle this case gracefully.

### 4. Non-Descriptive Function and Variable Names

The names used (`calc`, `x`, `y`, `z`) are very generic. Using more descriptive names like `calculate`, `num1`, `num2`, and `operation` would make the function's purpose immediately clear without needing to read the implementation.

Summary of Improvements:

Clarity: Function and variable names are now descriptive.

Documentation: A comprehensive docstring has been added.

Robustness: It now includes a check to prevent `ZeroDivisionError` and handles invalid operations gracefully.

Readability: The code is formatted consistently according to PEP 8 guidelines.

**OBSERVATION:**

AI generated the accurate report of the code and it also corrected the code according to the prompt in an efficient way.