

AI ASSISTED CODING

LAB-10.4

NAME: B, Nishant

ENROLL.NO:2403A52089

BATCH: AI - 04

TASK-01:

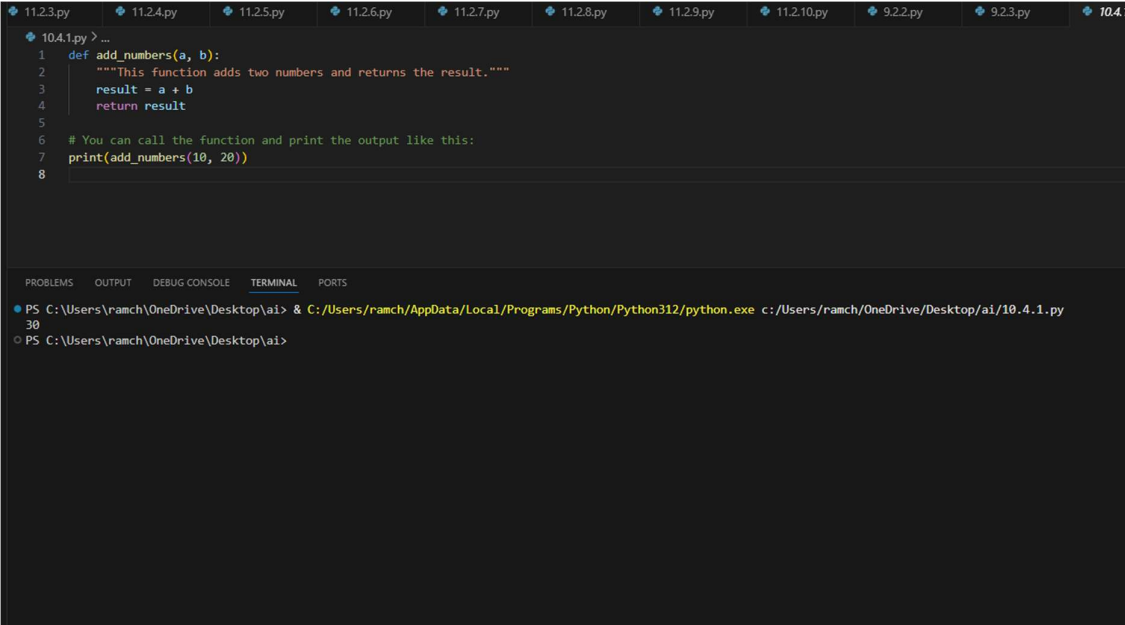
Identify and fix syntax, indentation, and variable errors in the given script.

```
# 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 with a dark theme. The top bar displays several open files: 11.2.3.py, 11.2.4.py, 11.2.5.py, 11.2.6.py, 11.2.7.py, 11.2.8.py, 11.2.9.py, 11.2.10.py, 9.2.2.py, 9.2.3.py, and 10.4.1.py. The active file, 10.4.1.py, contains the following Python code:

```
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
```

The bottom panel of the editor is divided into three tabs: PROBLEMS, OUTPUT, and TERMINAL. The TERMINAL tab is active, showing the command prompt output:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/10.4.1.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:

```
1042.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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

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.

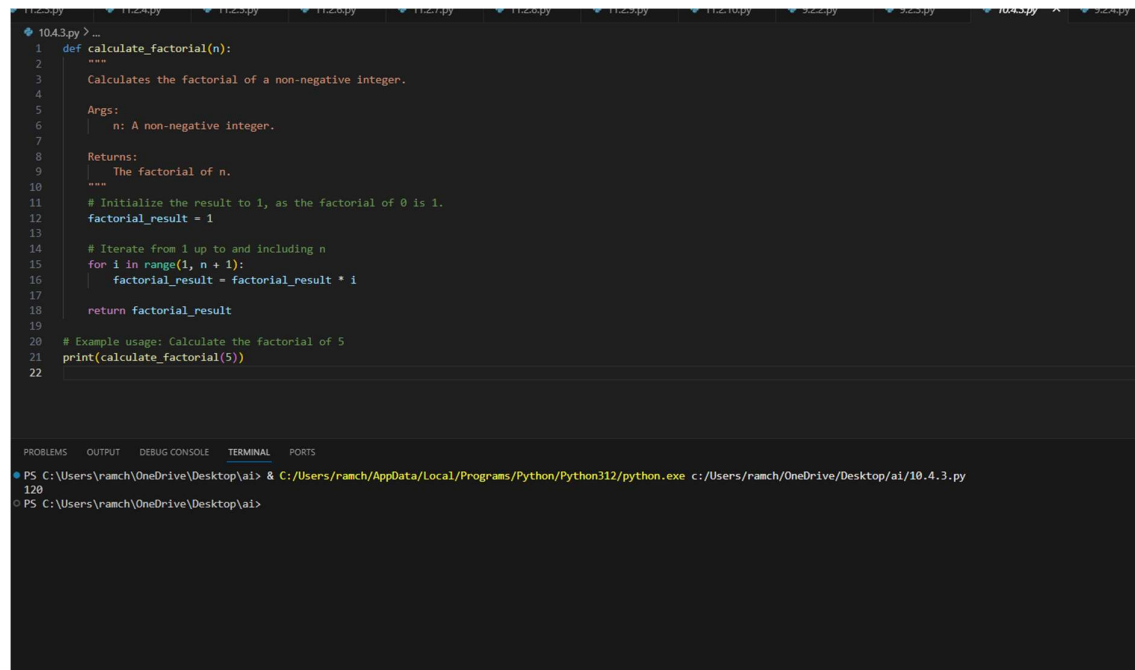
buggy_code_task3.py

```
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 a dark theme. The main window displays a Python script named `1043.py`. The script defines a function `calculate_factorial(n)` with a docstring, initializes `factorial_result = 1`, iterates from 1 to `n` to calculate the factorial, and returns the result. An example usage is shown at the bottom: `print(calculate_factorial(5))`. The bottom panel shows the terminal output: `PS C:\Users\ranch\OneDrive\Desktop\ai> & C:\Users\ranch\AppData\Local\Programs\Python\Python312\python.exe c:/Users/ranch/OneDrive/Desktop/ai/10.4.3.py` followed by the output `120`.

```
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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\ranch\OneDrive\Desktop\ai> & C:\Users\ranch\AppData\Local\Programs\Python\Python312\python.exe c:/Users/ranch/OneDrive/Desktop/ai/10.4.3.py  
120  
PS C:\Users\ranch\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:

```
10.4.4.py > setup_dummy_database
1 import sqlite3
2
3 def setup_dummy_database():
4     """Creates and populates a dummy users.db for demonstration."""
5     conn = sqlite3.connect("users.db")
6     cursor = conn.cursor()
7     # Drop table if it exists to start fresh
8     cursor.execute("DROP TABLE IF EXISTS users")
9     # Create table
10    cursor.execute("""
11        CREATE TABLE users (
12            id INTEGER PRIMARY KEY,
13            name TEXT NOT NULL,
14            email TEXT NOT NULL
15        );
16    """)
17    # Insert some sample data
18    cursor.execute("INSERT INTO users (id, name, email) VALUES (?, ?, ?)", (1, 'Alice', 'alice@example.com'))
19    cursor.execute("INSERT INTO users (id, name, email) VALUES (?, ?, ?)", (2, 'Bob', 'bob@example.com'))
20    conn.commit()
21    conn.close()
22    print("Dummy database 'users.db' created and populated.")
23
24 def get_user_data(user_id):
25     """
26     Securely fetches user data from the database for a given user ID.
27
28     Args:
29         user_id: The ID of the user to retrieve.
30
31     Returns:
32         A list of tuples containing the user data, or None if an error occurs.
33     """
34     conn = None # Initialize conn to None to handle connection errors
35     try:
36         # Connect to the database
37         conn = sqlite3.connect("users.db")
38         cursor = conn.cursor()
39
40         # Use a parameterized query to prevent SQL injection
41         # The '?' is a placeholder for the data.
42         query = "SELECT * FROM users WHERE id = ?;"
43
44         # Execute the query, passing the user_id as a tuple
45         cursor.execute(query, (user_id,))
46
47         result = cursor.fetchall()
48         return result
```

```
10.4.4.py > setup_dummy_database
24 def get_user_data(user_id):
47     result = cursor.fetchall()
48     return result
49
50 except sqlite3.Error as e:
51     # Handle potential database-related errors (e.g., table not found, DB locked)
52     print(f"Database error: {e}")
53     return None
54
55 finally:
56     # Ensure the database connection is closed, even if an error occurred
57     if conn:
58         conn.close()
59
60 # --- Example Usage ---
61 if __name__ == "__main__":
62     # To run this example, you first need to create a dummy database.
63     setup_dummy_database() # Uncomment to create the DB and table
64
65     user_input = input("Enter user ID: ")
66     user_data = get_user_data(user_input)
67
68     if user_data:
69         print("\n--- User Data ---")
70         for row in user_data:
71             print(row)
72         print("-----")
73     else:
74         print("Could not retrieve user data or user not found.")
75
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/10.4.4.py
Dummy database 'users.db' created and populated.
Enter user ID: 1

--- User Data ---
(1, 'Alice', 'alice@example.com')
-----
PS C:\Users\ramch\OneDrive\Desktop\ai> █
```

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.

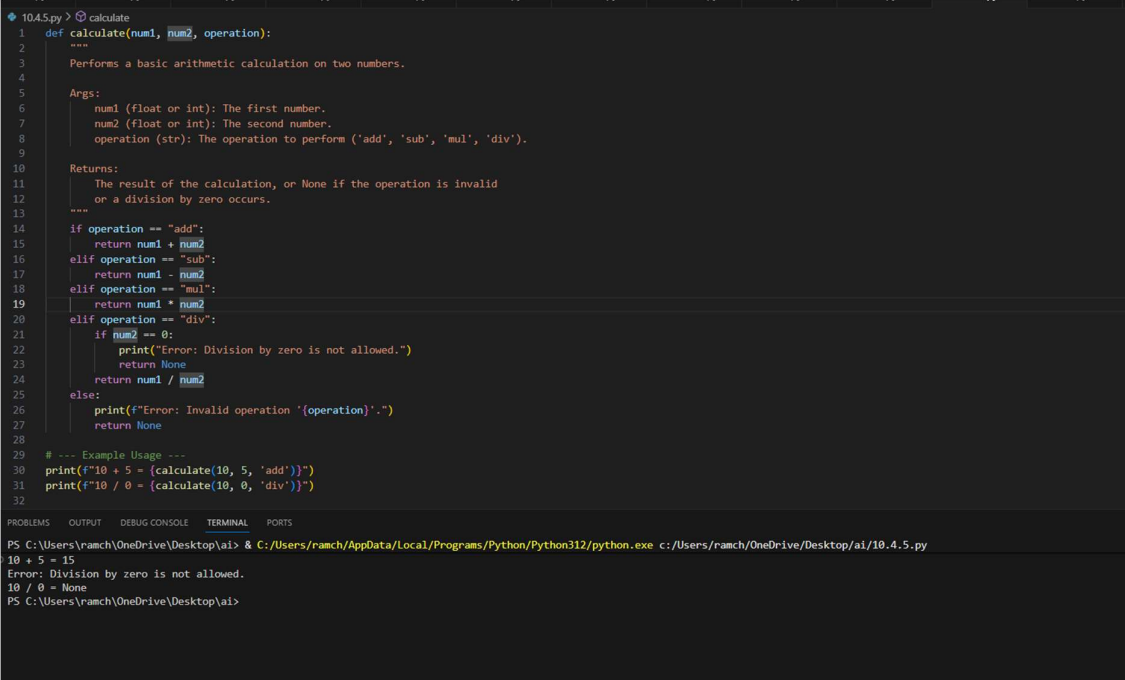
```
# buggy_code_task5.py
```

```
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 displays a code editor with a Python script named `10.4.5.py`. The script defines a function `calculate` that takes three arguments: `num1`, `num2`, and `operation`. It performs basic arithmetic operations: addition, subtraction, multiplication, and division. The function includes docstrings for arguments and returns, and handles errors like division by zero and invalid operations. Example usage is provided at the bottom of the script.

```
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')}")
32
```

The terminal output shows the execution of the script. It prints the result of `10 + 5` as `15`. For `10 / 0`, it prints an error message: `Error: Division by zero is not allowed.` and `None`.

```
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.