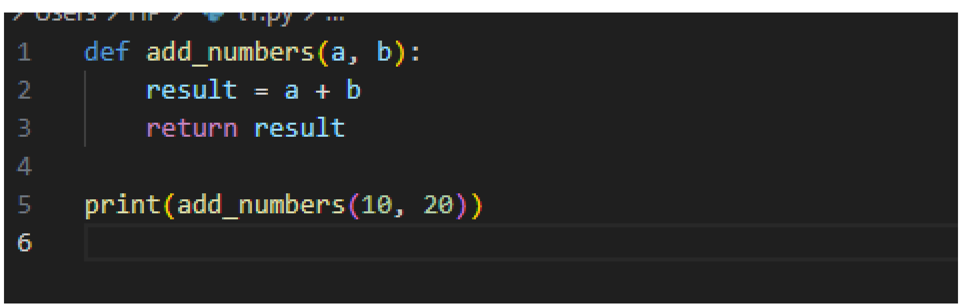
Ai assisted coding assignment 10.3

Task 1: Syntax and Error Detection

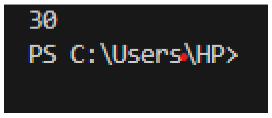
Prompt: "Write a Python function that takes a list of integers and returns a new list with only the even numbers, preserving the original order. Then test it with a sample list.

Code:



```python
def add_numbers(a, b):
    result = a + b
    return result

print(add_numbers(10, 20))

```

Output:



```
30
PS C:\Users\HP>
```

Observation:

1) *The function* `add_numbers` *correctly takes two parameters and returns their sum.*

2) *Proper indentation is essential in Python to define the function body and avoid syntax errors.*

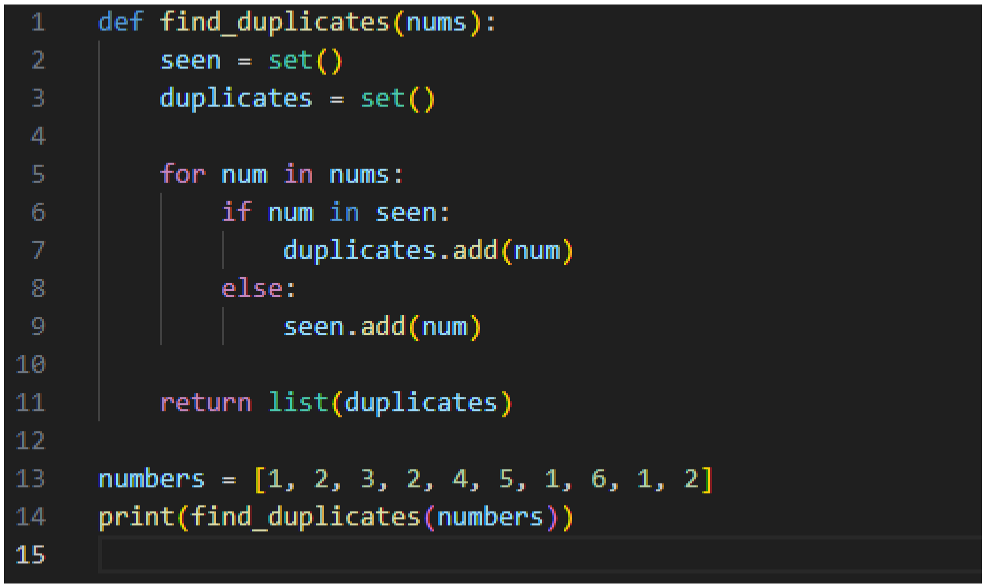
3) *The colon after the function declaration is mandatory to start the function block.*

4) *Using descriptive and consistent variable names like* `result` *helps avoid confusion and errors.*

Task 2: Logical and Performance Issue Review

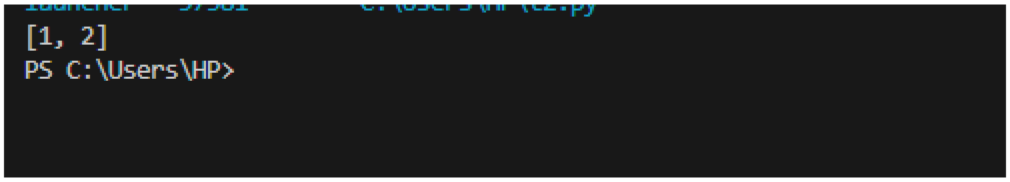
Prompt2 **: identify duplicate values in a list, but its current implementation is inefficient due to unnecessary nested loops and redundant checks. Your task is to** refactor the code **to improve its performance** without changing its output**.**

Code :



```python
def find_duplicates(nums):
    seen = set()
    duplicates = set()

    for num in nums:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)

    return list(duplicates)

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

Output:



```
[1, 2]
PS C:\Users\HP>
```

Observation:

1)The function uses nested loops that result in repeated and unnecessary comparisons between elements.

2)It has a time complexity of O(n²), which makes it inefficient for large input lists.

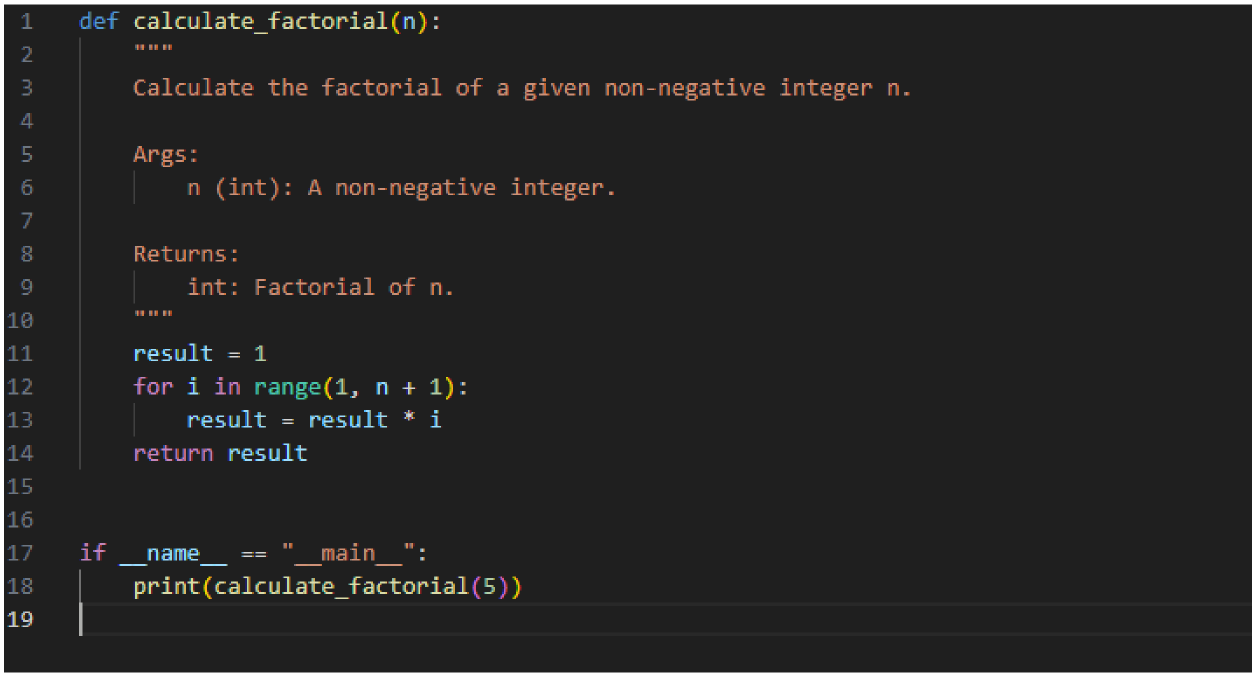
3)The check `nums[i] not in duplicates` is performed multiple times, increasing the overall runtime.

4)Using a list to check for existing duplicates leads to slower membership tests; a set would be more efficient

Task 3: Code Refactoring for Readability

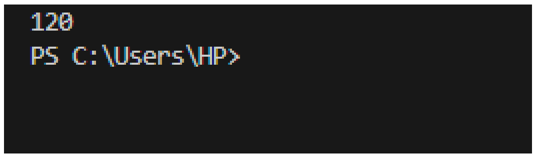
Prompt: compute the factorial of a given number. However, the code is poorly written, with unclear naming, inconsistent formatting, and no regard for Python's style conventions.

Code:



```python
def calculate_factorial(n):
    """
    Calculate the factorial of a given non-negative integer n.

    Args:
        n (int): A non-negative integer.

    Returns:
        int: Factorial of n.
    """
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result


if __name__ == "__main__":
    print(calculate_factorial(5))
```

Output:



```
120
PS C:\Users\HP>
```

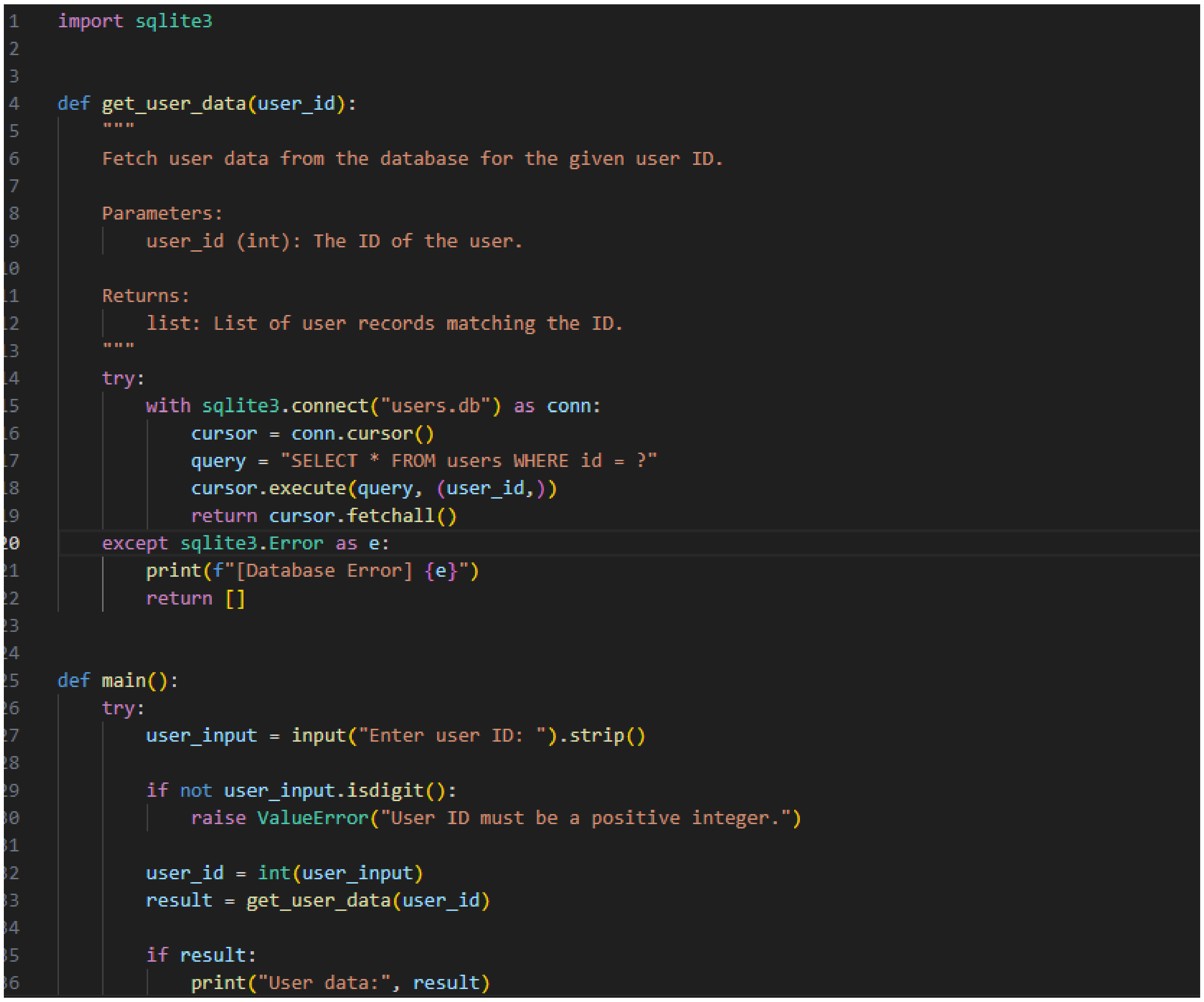
Observation:

1)The function name c is not descriptive and does not convey its purpose.
2)Variable names like x are vague and should be renamed for better readability.
3)he code lacks proper indentation, which makes it harder to read and violates Python syntax standards.
4)There are no comments or documentation to explain what the function does.
5)The function does not follow PEP 8 naming conventions or formatting guidelines.

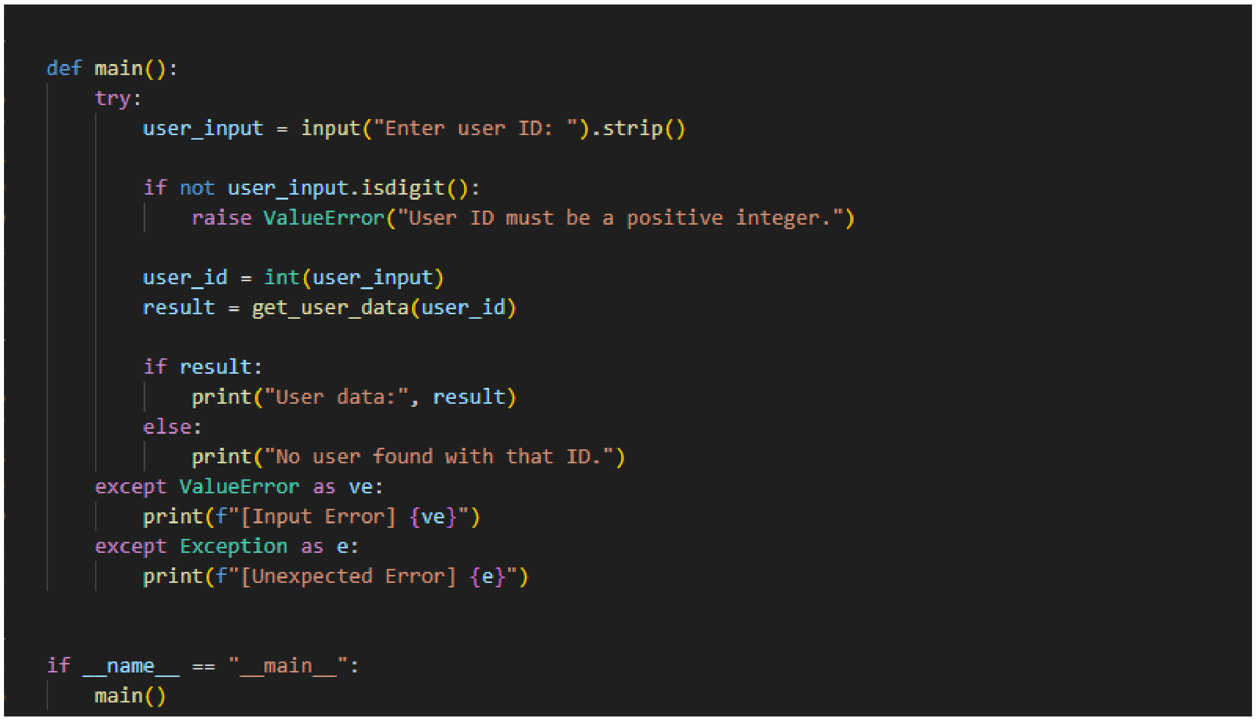
Task 4: Security and Error Handling Enhancement

Prompt: **An SQLite database using a user-provided ID. However, it lacks proper** security measures **and** exception handling**, making it vulnerable to issues like** SQL injection **and unexpected runtime errors.**

Code:

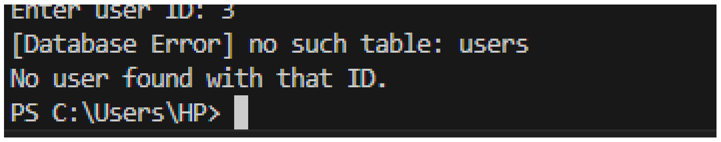


```python
import sqlite3


def get_user_data(user_id):
    """
    Fetch user data from the database for the given user ID.

    Parameters:
        user_id (int): The ID of the user.

    Returns:
        list: List of user records matching the ID.
    """
    try:
        with sqlite3.connect("users.db") as conn:
            cursor = conn.cursor()
            query = "SELECT * FROM users WHERE id = ?"
            cursor.execute(query, (user_id,))
            return cursor.fetchall()
    except sqlite3.Error as e:
        print(f"[Database Error] {e}")
        return []


def main():
    try:
        user_input = input("Enter user ID: ").strip()

        if not user_input.isdigit():
            raise ValueError("User ID must be a positive integer.")

        user_id = int(user_input)
        result = get_user_data(user_id)

        if result:
            print("User data:", result)
```



```python
def main():
    try:
        user_input = input("Enter user ID: ").strip()

        if not user_input.isdigit():
            raise ValueError("User ID must be a positive integer.")

        user_id = int(user_input)
        result = get_user_data(user_id)

        if result:
            print("User data:", result)
        else:
            print("No user found with that ID.")
    except ValueError as ve:
        print(f"[Input Error] {ve}")
    except Exception as e:
        print(f"[Unexpected Error] {e}")


if __name__ == "__main__":
    main()
```

Output:



```
Enter user ID: 3
[Database Error] no such table: users
No user found with that ID.
PS C:\Users\HP>
```

Observation:

1) The code is vulnerable to **SQL injection** because it uses string formatting to build the SQL query.

2) There is **no input validation**, so any input from the user is passed directly into the query, which is unsafe.

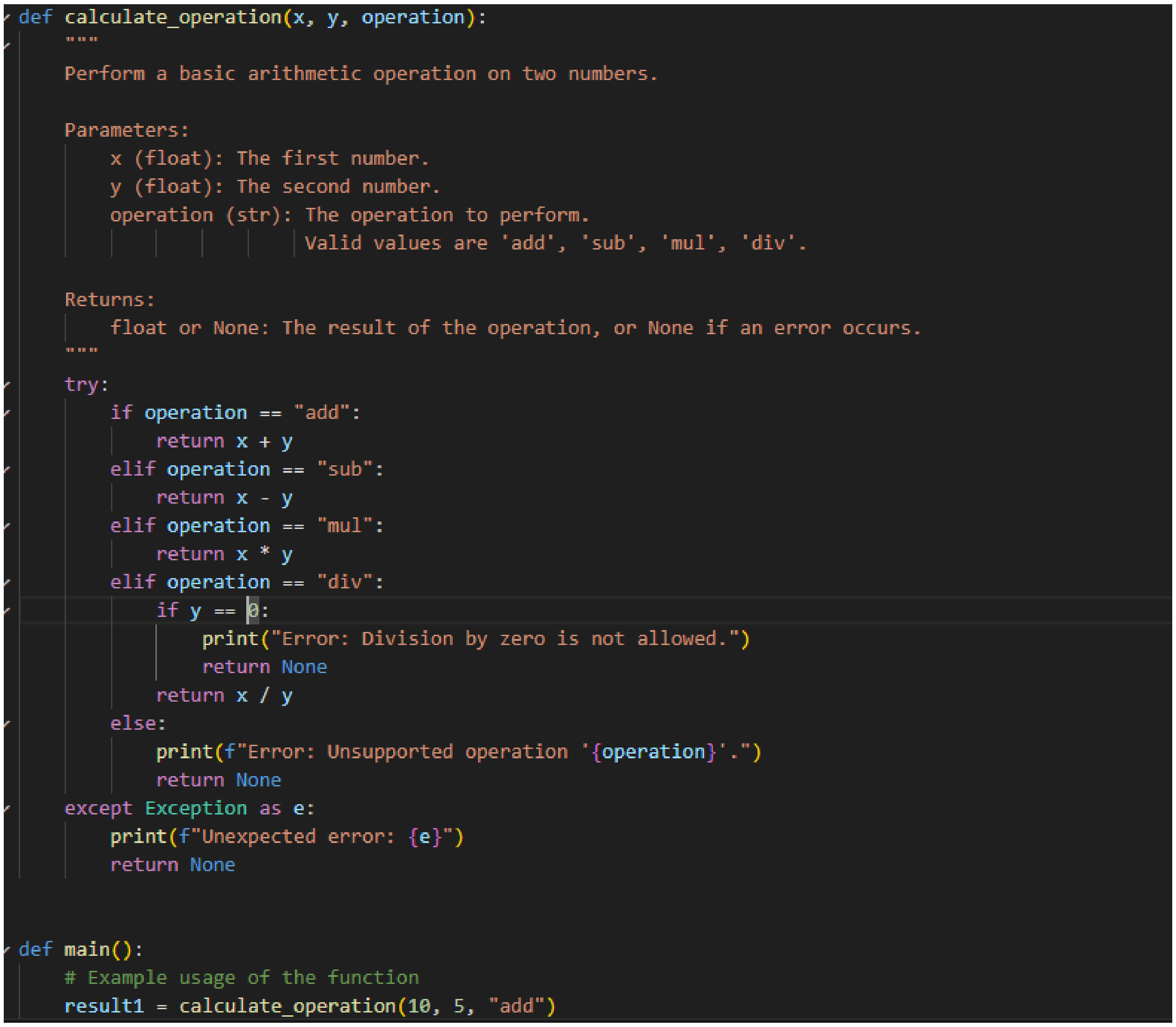
3) **Exception handling is missing**, meaning any database connection or execution error will crash the program.

4) The **database connection is not safely managed**—if an error occurs before `conn.close()`, the connection might remain open.

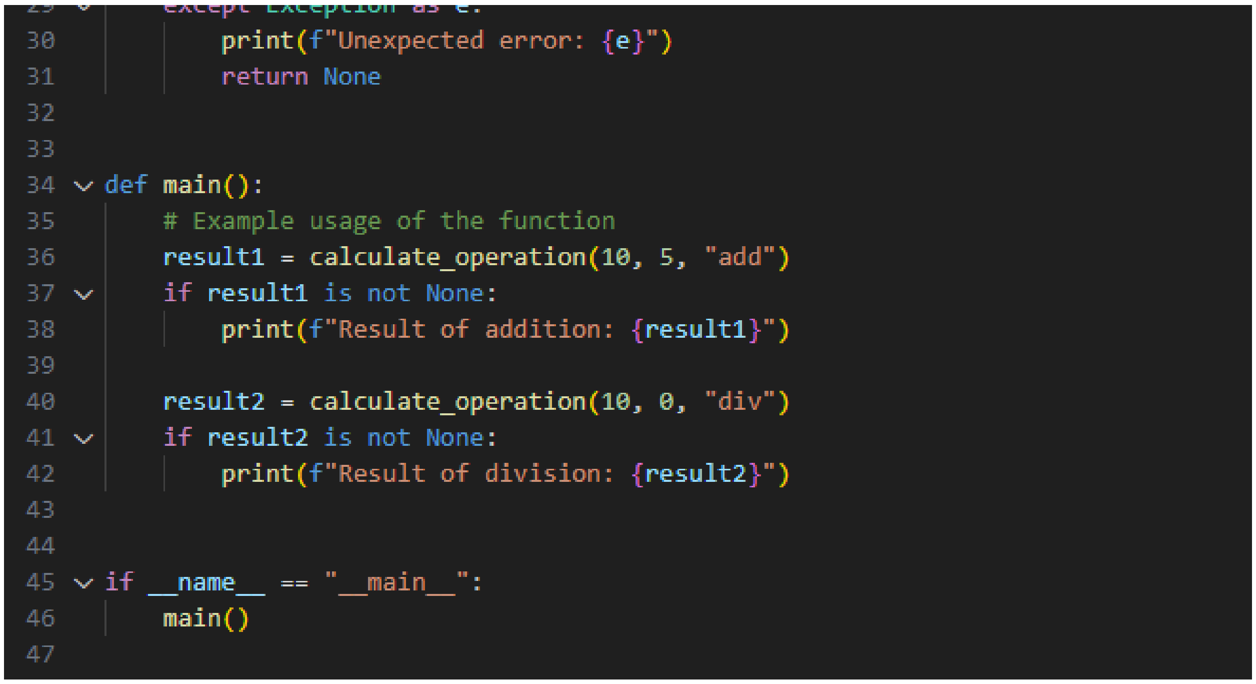
Task 5: Automated Code Review Report Generation

Prompt: Generate a Python function performs basic arithmetic operations based on the given operator string. However, the code is poorly written, lacks proper structure, and does not handle errors gracefully.

Code:

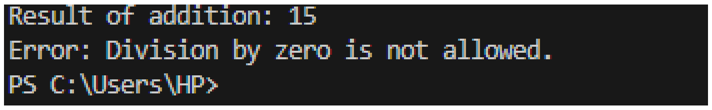


```python
def calculate_operation(x, y, operation):
    """
    Perform a basic arithmetic operation on two numbers.

    Parameters:
        x (float): The first number.
        y (float): The second number.
        operation (str): The operation to perform.
                         Valid values are 'add', 'sub', 'mul', 'div'.

    Returns:
        float or None: The result of the operation, or None if an error occurs.
    """
    try:
        if operation == "add":
            return x + y
        elif operation == "sub":
            return x - y
        elif operation == "mul":
            return x * y
        elif operation == "div":
            if y == 0:
                print("Error: Division by zero is not allowed.")
                return None
            return x / y
        else:
            print(f"Error: Unsupported operation '{operation}'.")
            return None
    except Exception as e:
        print(f"Unexpected error: {e}")
        return None


def main():
    # Example usage of the function
    result1 = calculate_operation(10, 5, "add")
```



```python
    except Exception as e:
        print(f"Unexpected error: {e}")
        return None


def main():
    # Example usage of the function
    result1 = calculate_operation(10, 5, "add")
    if result1 is not None:
        print(f"Result of addition: {result1}")

    result2 = calculate_operation(10, 0, "div")
    if result2 is not None:
        print(f"Result of division: {result2}")


if __name__ == "__main__":
    main()
```

Output:



```
Result of addition: 15
Error: Division by zero is not allowed.
PS C:\Users\HP>
```

Observation:

1) The function name `calc` is generic and does not clearly describe its purpose—`calculate` or `perform_operation` would be more descriptive.

2)The code lacks proper **indentation and formatting**, violating **PEP 8** style guidelines.

3) The `elif z=="sub": return x-y` line is written on a single line, making it harder to read and inconsistent with other branches.