AI Assistant Coding

Name: CH. Arush Nandhan

HTNO: 2303A51880

Batch-14

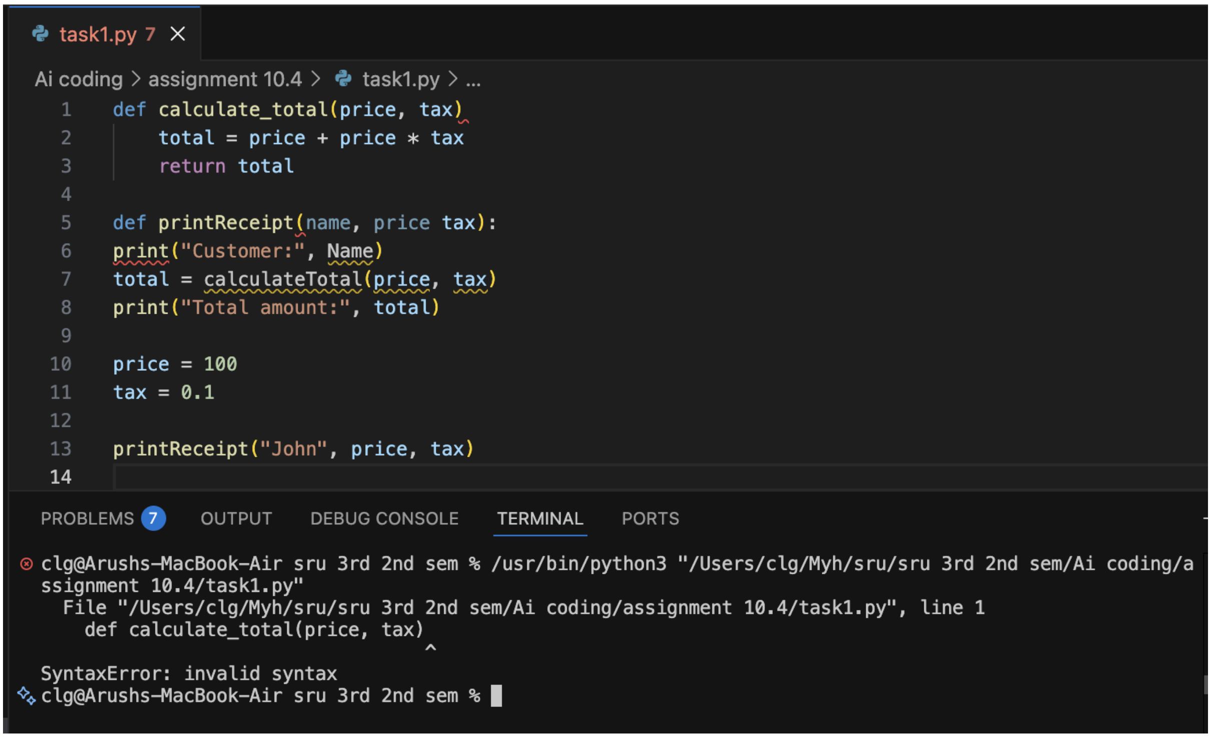
Date: 05-02-2026

Assignment - 10.4

solution:

Task #1:

Original Code:



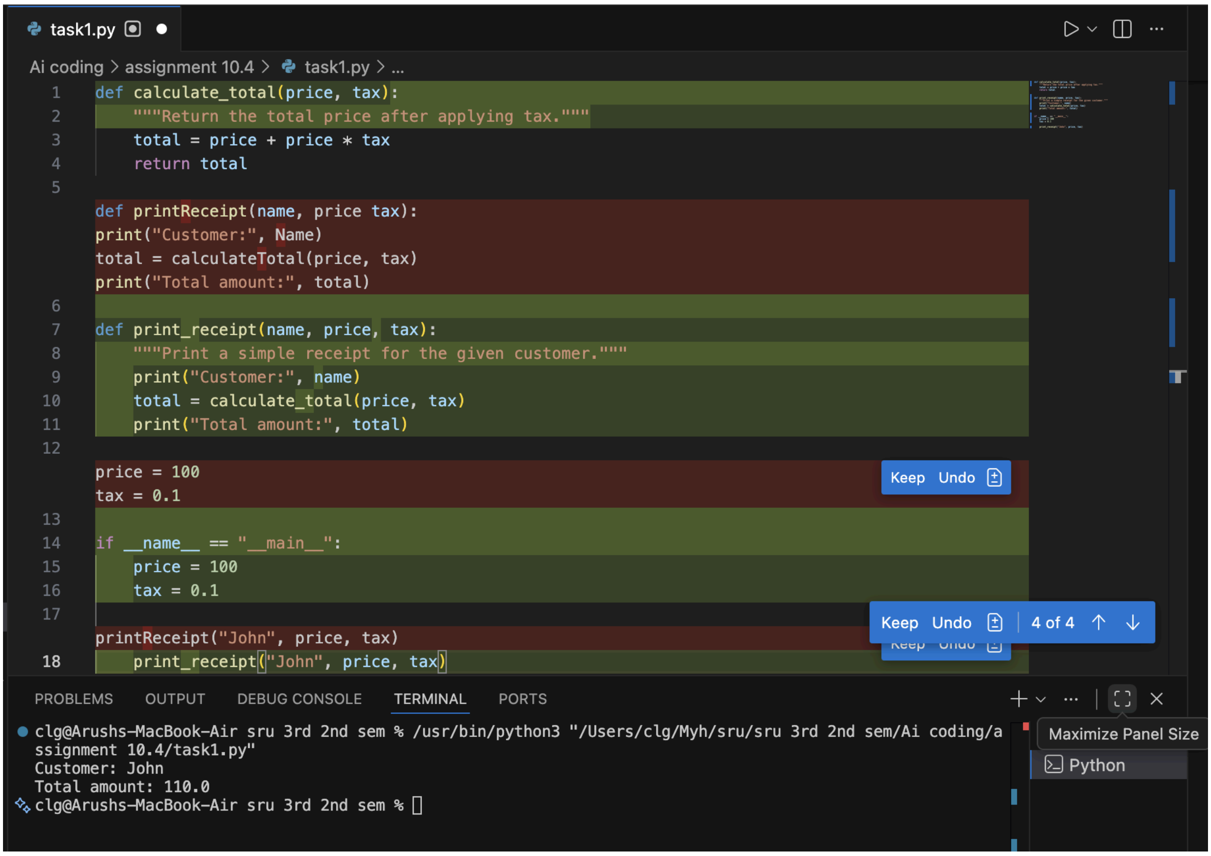
```python
def calculate_total(price, tax)
    total = price + price * tax
    return total

def printReceipt(name, price tax):
print("Customer:", Name)
total = calculateTotal(price, tax)
print("Total amount:", total)

price = 100
tax = 0.1

printReceipt("John", price, tax)
```

```
clg@Arushs-MacBook-Air sru 3rd 2nd sem % /usr/bin/python3 "/Users/clg/Myh/sru/sru 3rd 2nd sem/Ai coding/a
ssignment 10.4/task1.py"
  File "/Users/clg/Myh/sru/sru 3rd 2nd sem/Ai coding/assignment 10.4/task1.py", line 1
    def calculate_total(price, tax)
                                   ^
SyntaxError: invalid syntax
clg@Arushs-MacBook-Air sru 3rd 2nd sem % 
```

Prompt:

Review the following Python script and identify all syntactic and structural errors. Correct the syntax errors, indentation issues, incorrect variable names, and faulty function calls. Generate a clean, readable, and fully executable version of the script. Also provide an explanation describing the syntax fixes, naming corrections, and structural improvements made.

AI Generated Code:



```python
def calculate_total(price, tax):
    """Return the total price after applying tax."""
    total = price + price * tax
    return total

def printReceipt(name, price tax):
print("Customer:", Name)
total = calculateTotal(price, tax)
print("Total amount:", total)

def print_receipt(name, price, tax):
    """Print a simple receipt for the given customer."""
    print("Customer:", name)
    total = calculate_total(price, tax)
    print("Total amount:", total)

price = 100
tax = 0.1

if __name__ == "__main__":
    price = 100
    tax = 0.1

printReceipt("John", price, tax)
    print_receipt("John", price, tax)
```

```
clg@Arushs-MacBook-Air sru 3rd 2nd sem % /usr/bin/python3 "/Users/clg/Myh/sru/sru 3rd 2nd sem/Ai coding/a
ssignment 10.4/task1.py"
Customer: John
Total amount: 110.0
clg@Arushs-MacBook-Air sru 3rd 2nd sem % 
```

Syntax fixes: Added missing colon to def calculate_total(...) and to def print_receipt(...), and added missing comma between parameters.

Indentation: Corrected indentation of print_receipt body so function blocks are properly indented.

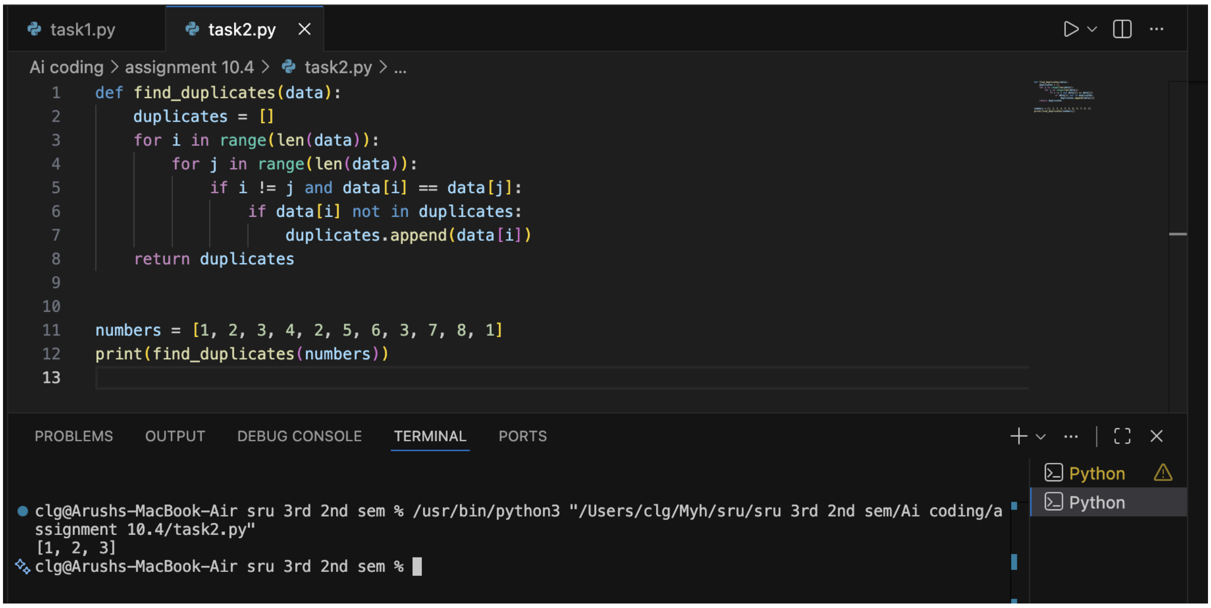
Naming: Normalized function names to snake_case (calculate_total, print_receipt) and corrected Name ⟶ name.

Function call: Replaced wrong call calculateTotal with calculate_total.

Structure: Added if __name__ == "__main__": guard and moved example usage under it.

Task #2:

Original Code:



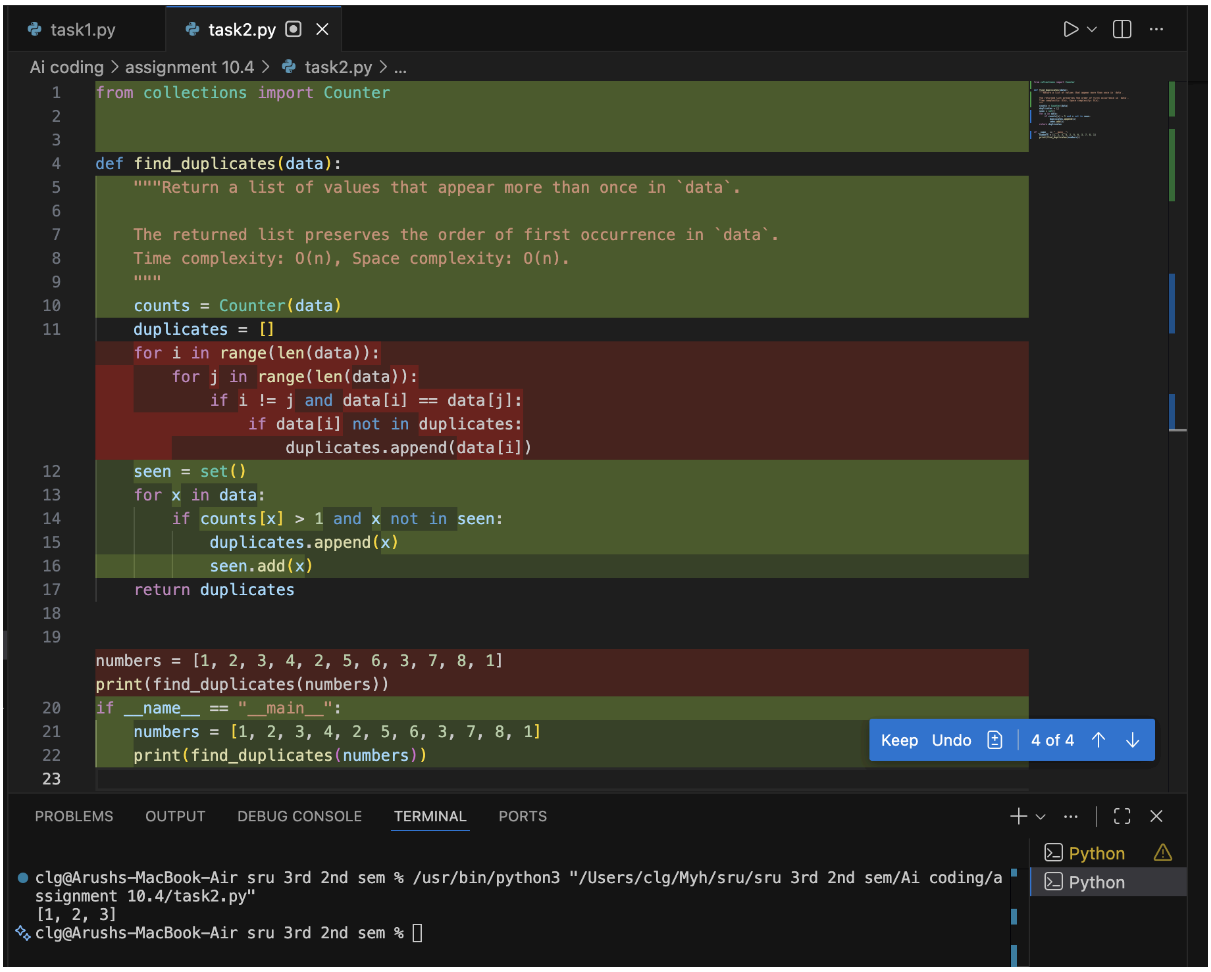
```python
def find_duplicates(data):
    duplicates = []
    for i in range(len(data)):
        for j in range(len(data)):
            if i != j and data[i] == data[j]:
                if data[i] not in duplicates:
                    duplicates.append(data[i])
    return duplicates


numbers = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print(find_duplicates(numbers))
```

```
clg@Arushs-MacBook-Air sru 3rd 2nd sem % /usr/bin/python3 "/Users/clg/Myh/sru/sru 3rd 2nd sem/Ai coding/a
ssignment 10.4/task2.py"
[1, 2, 3]
clg@Arushs-MacBook-Air sru 3rd 2nd sem %
```

Prompt:

Analyze the following Python function that identifies duplicate values using nested loops. Review the logic to identify performance bottlenecks and refactor the code to improve time complexity while preserving the correctness of the output. Provide the optimized implementation and explain why the original approach was inefficient and how the improved version enhances performance.

AI Generated Code:



```python
from collections import Counter


def find_duplicates(data):
    """Return a list of values that appear more than once in `data`.

    The returned list preserves the order of first occurrence in `data`.
    Time complexity: O(n), Space complexity: O(n).
    """
    counts = Counter(data)
    duplicates = []
    for i in range(len(data)):
        for j in range(len(data)):
            if i != j and data[i] == data[j]:
                if data[i] not in duplicates:
                    duplicates.append(data[i])
    seen = set()
    for x in data:
        if counts[x] > 1 and x not in seen:
            duplicates.append(x)
            seen.add(x)
    return duplicates


numbers = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print(find_duplicates(numbers))
if __name__ == "__main__":
    numbers = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
    print(find_duplicates(numbers))
```

```
clg@Arushs-MacBook-Air sru 3rd 2nd sem % /usr/bin/python3 "/Users/clg/Myh/sru/sru 3rd 2nd sem/Ai coding/a
ssignment 10.4/task2.py"
[1, 2, 3]
clg@Arushs-MacBook-Air sru 3rd 2nd sem % 
```

Why the original approach was inefficient

The original used two nested loops comparing each pair (indices i and j). That is O(n^2) work just to detect matches.

Additionally, it performed if data[i] not in duplicates inside the inner loop (list membership test is O(k)), which can further increase work — worst-case approaching O(n^3) behavior for some inputs.

This repeated comparison and repeated membership checks cause significant redundant work for large inputs.

Why the refactor is better

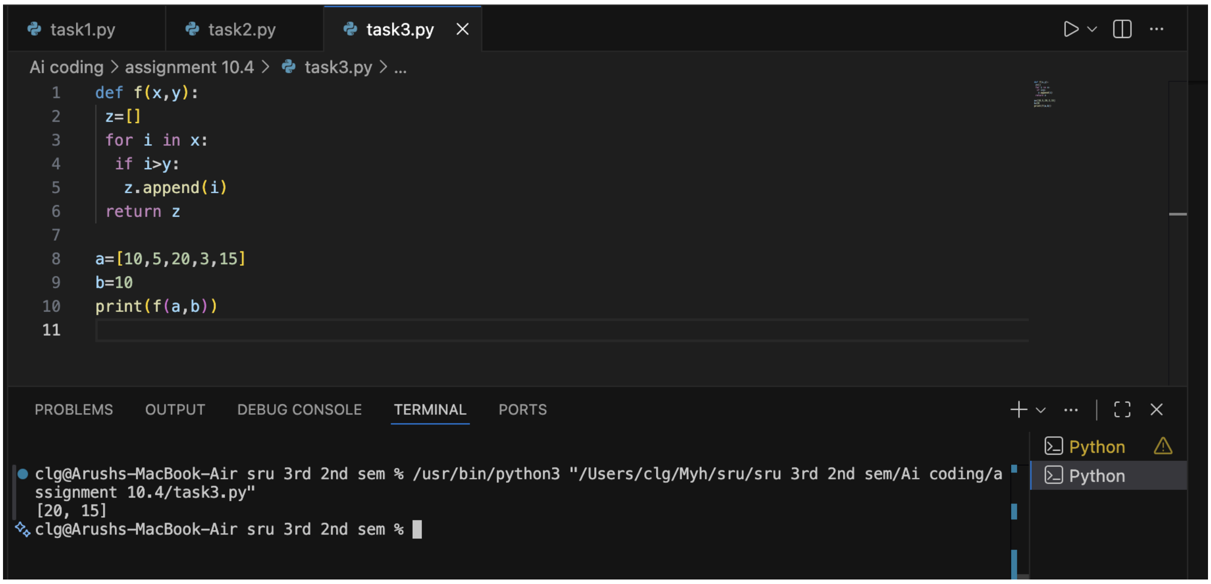
Counting occurrences with a hash-based Counter uses a single pass (O(n)). Iterating the list once to collect items with count > 1 is another O(n) pass — overall O(n).

Using set for seen makes membership checks O(1) average-case, avoiding expensive list scans.

The refactor eliminates redundant pairwise comparisons and repeated list membership checks, producing the same logical result (unique duplicate values) while preserving the order of first occurrence.

Task #3:

Original Code:



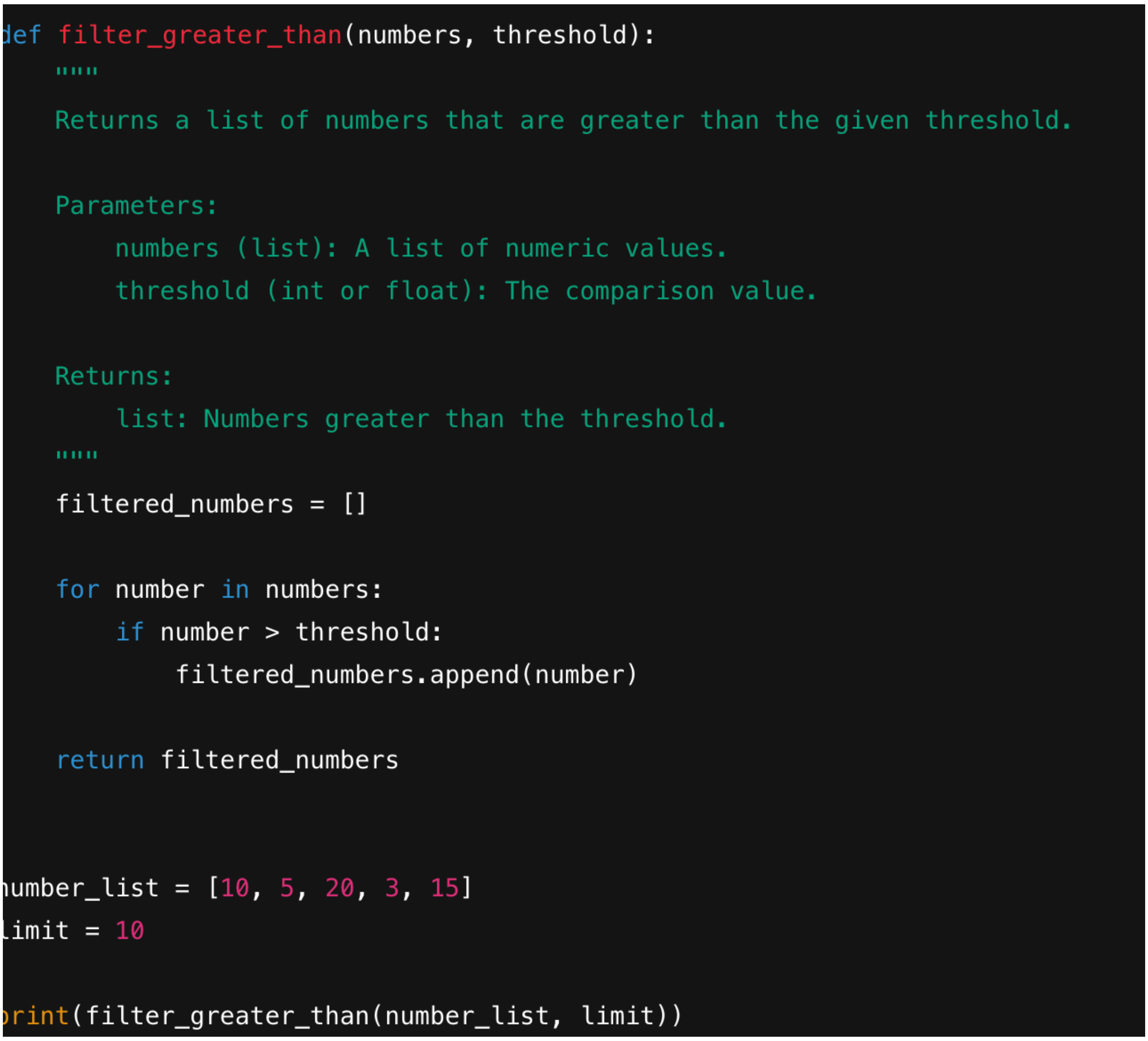
```
def f(x,y):
 z=[]
 for i in x:
  if i>y:
   z.append(i)
 return z

a=[10,5,20,3,15]
b=10
print(f(a,b))
```

```
clg@Arushs-MacBook-Air sru 3rd 2nd sem % /usr/bin/python3 "/Users/clg/Myh/sru/sru 3rd 2nd sem/Ai coding/a
ssignment 10.4/task3.py"
[20, 15]
clg@Arushs-MacBook-Air sru 3rd 2nd sem % 
```

Prompt:

Refactor the following Python function to improve clarity and long-term maintainability. Improve cryptic function and variable names, correct indentation, and apply proper formatting standards. Add meaningful documentation explaining the purpose of the function. Provide a clean, well-structured version of the code and explain the readability improvements made.

AI Generated Code:



```python
def filter_greater_than(numbers, threshold):
    """
    Returns a list of numbers that are greater than the given threshold.

    Parameters:
        numbers (list): A list of numeric values.
        threshold (int or float): The comparison value.

    Returns:
        list: Numbers greater than the threshold.
    """
    filtered_numbers = []

    for number in numbers:
        if number > threshold:
            filtered_numbers.append(number)

    return filtered_numbers


number_list = [10, 5, 20, 3, 15]
limit = 10

print(filter_greater_than(number_list, limit))
```

Readability Improvements

Renamed function f ⟶ filter_greater_than.

Renamed variables x, y, z, a, b ⟶ meaningful names.

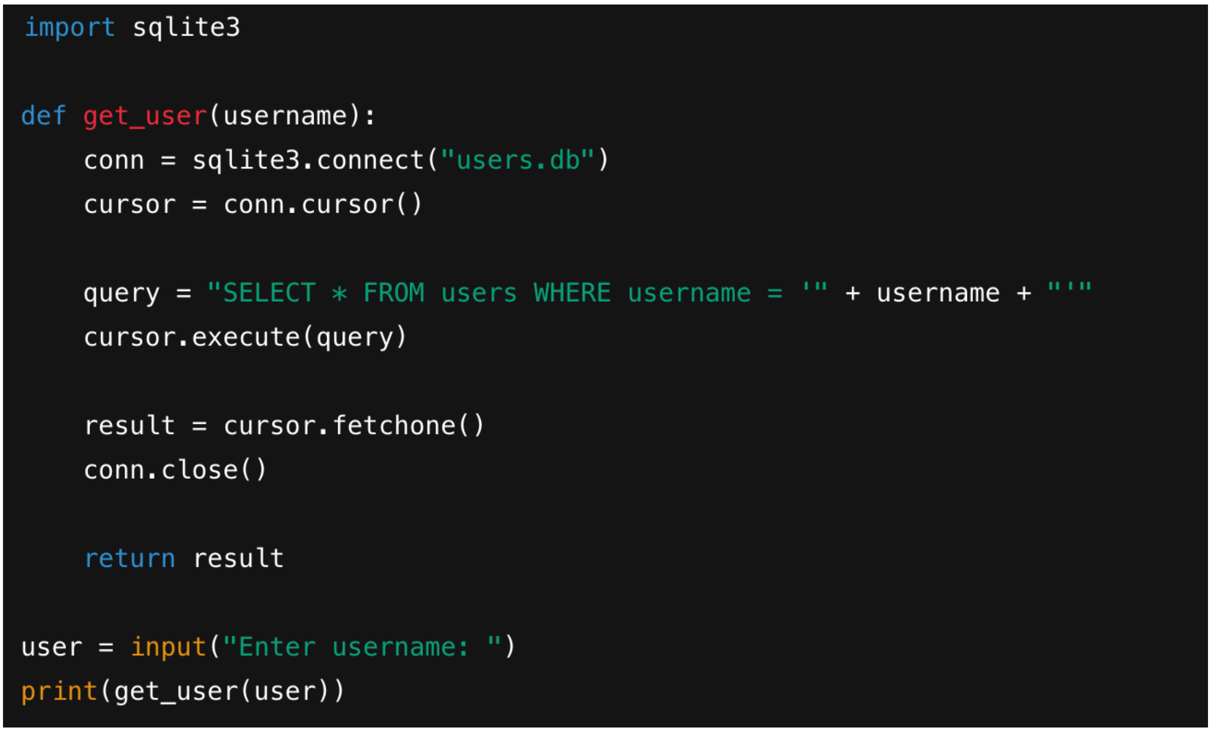
Applied proper indentation and spacing.

Added clear docstring explaining purpose, parameters, and return value.

Improved structure for long-term maintainability.

Task #4:

Original Code:

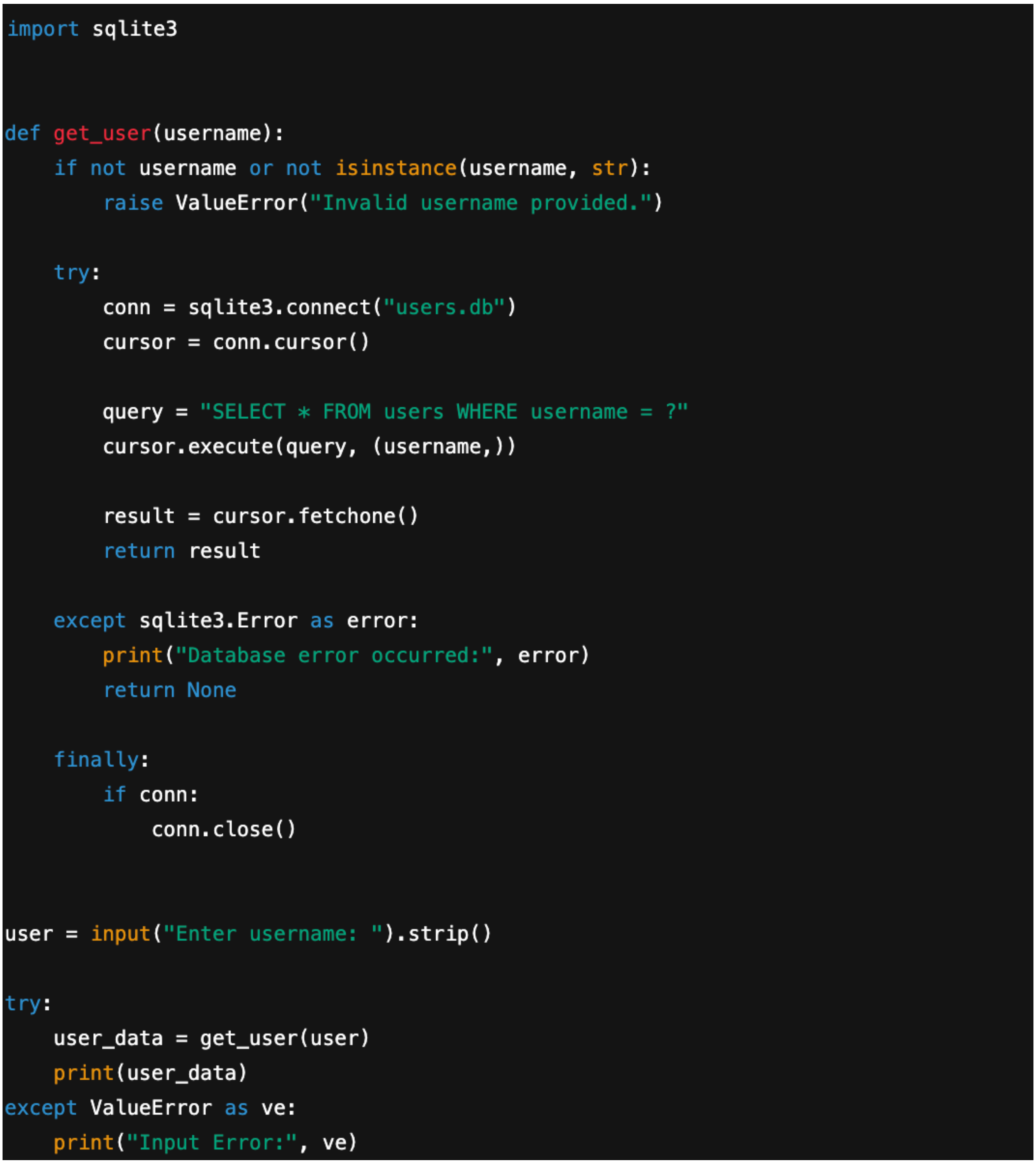


```python
import sqlite3

def get_user(username):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()

    query = "SELECT * FROM users WHERE username = '" + username + "'"
    cursor.execute(query)

    result = cursor.fetchone()
    conn.close()

    return result

user = input("Enter username: ")
print(get_user(user))
```

Prompt:

Review the following Python script and identify security vulnerabilities and reliability issues. Refactor the code using safe coding practices, replace unsafe SQL query construction with secure parameterized statements, add input validation, and implement proper exception handling. Provide a production-ready version of the script and explain the security and reliability improvements made.

AI Generated Code:



```python
import sqlite3


def get_user(username):
    if not username or not isinstance(username, str):
        raise ValueError("Invalid username provided.")

    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()

        query = "SELECT * FROM users WHERE username = ?"
        cursor.execute(query, (username,))

        result = cursor.fetchone()
        return result

    except sqlite3.Error as error:
        print("Database error occurred:", error)
        return None

    finally:
        if conn:
            conn.close()


user = input("Enter username: ").strip()

try:
    user_data = get_user(user)
    print(user_data)
except ValueError as ve:
    print("Input Error:", ve)
```

✔ Security and Reliability Improvements

1. SQL Injection Prevention

Replaced string concatenation with parameterized query (?).

Prevents malicious SQL input.

2. Input Validation

Checks if username is valid and non-empty.

Raises ValueError for invalid input.

3. Exception Handling

Added try-except block for database errors.

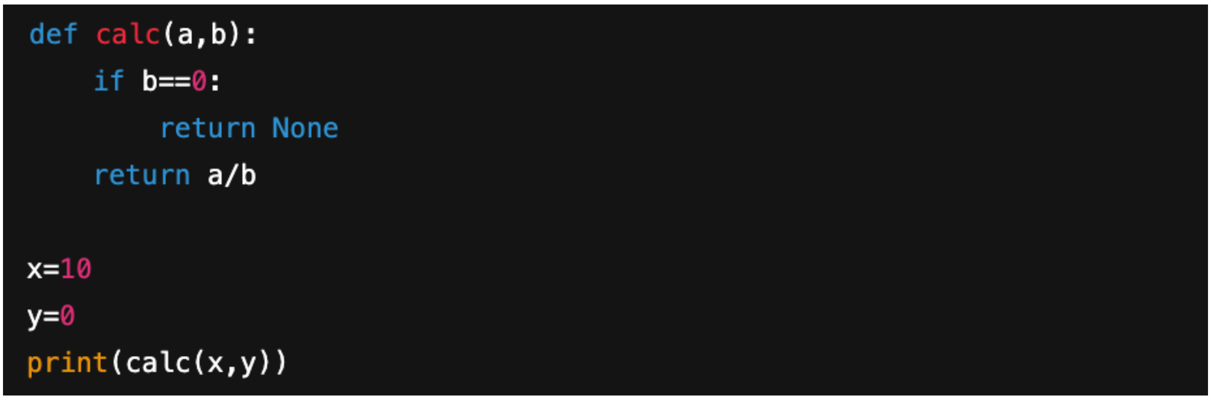
Prevents application crash.

4. Resource Safety

Used finally block to ensure connection closes properly.

Task #5:

Original Code:

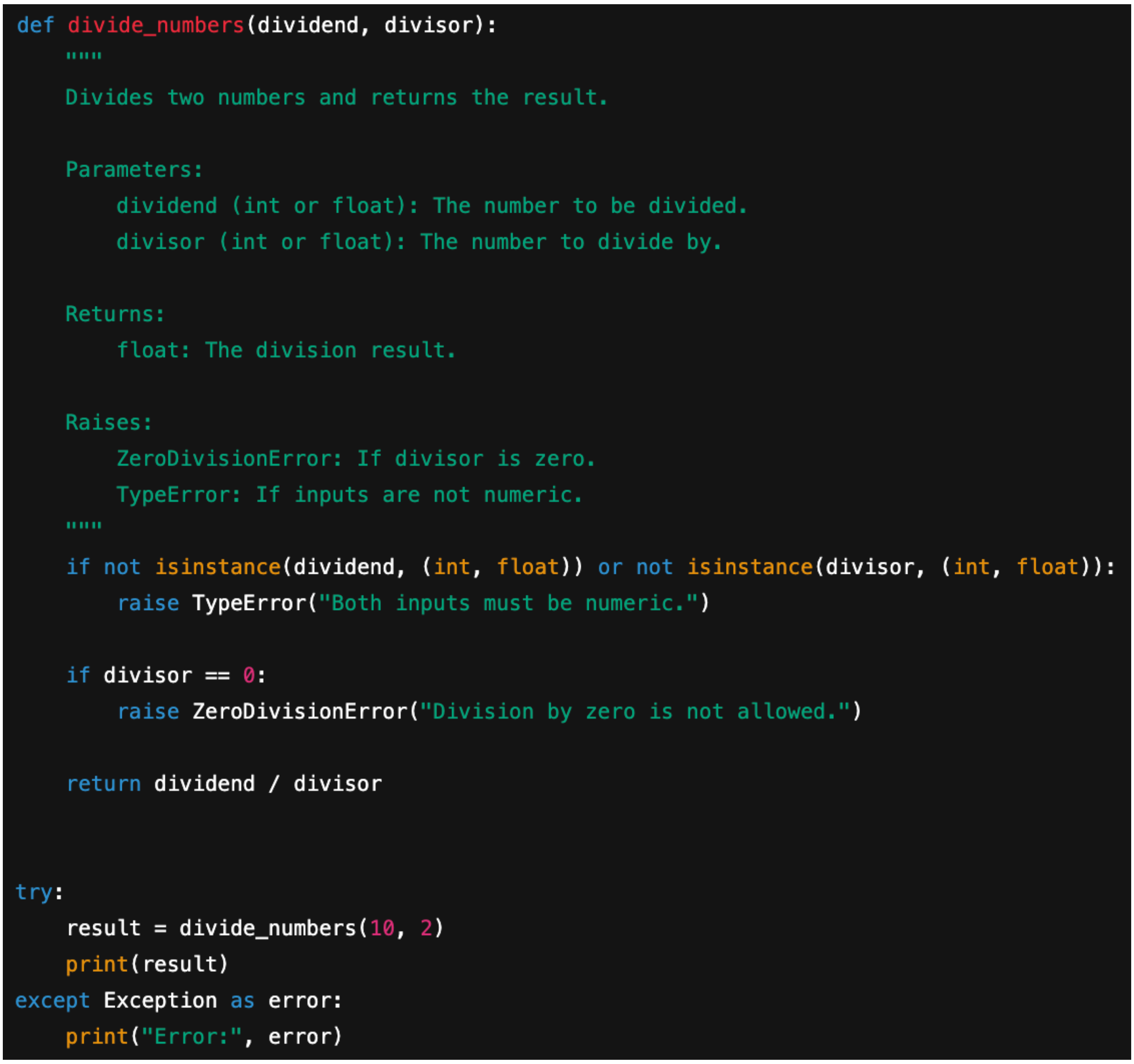


```python
def calc(a,b):
    if b==0:
        return None
    return a/b

x=10
y=0
print(calc(x,y))
```

Prompt:

Analyze the following poorly written Python script and generate a structured code review report evaluating code readability, naming conventions, formatting and style consistency, error handling, documentation quality, and maintainability. Identify quality issues, potential risk areas, and code smells, and provide improvement suggestions. Optionally include an improved version of the code.

AI Generated Code:



```python
def divide_numbers(dividend, divisor):
    """
    Divides two numbers and returns the result.

    Parameters:
        dividend (int or float): The number to be divided.
        divisor (int or float): The number to divide by.

    Returns:
        float: The division result.

    Raises:
        ZeroDivisionError: If divisor is zero.
        TypeError: If inputs are not numeric.
    """
    if not isinstance(dividend, (int, float)) or not isinstance(divisor, (int, float)):
        raise TypeError("Both inputs must be numeric.")

    if divisor == 0:
        raise ZeroDivisionError("Division by zero is not allowed.")

    return dividend / divisor


try:
    result = divide_numbers(10, 2)
    print(result)
except Exception as error:
    print("Error:", error)
```