Assignment – 10.4

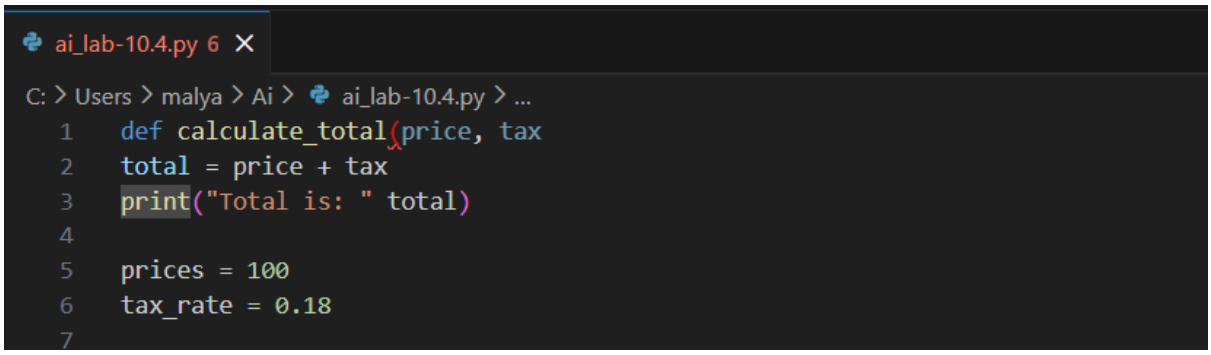
M. Sai Teja

Batch_14

2303A510A3

Task 1: AI-Assisted Syntax and Code Quality Review

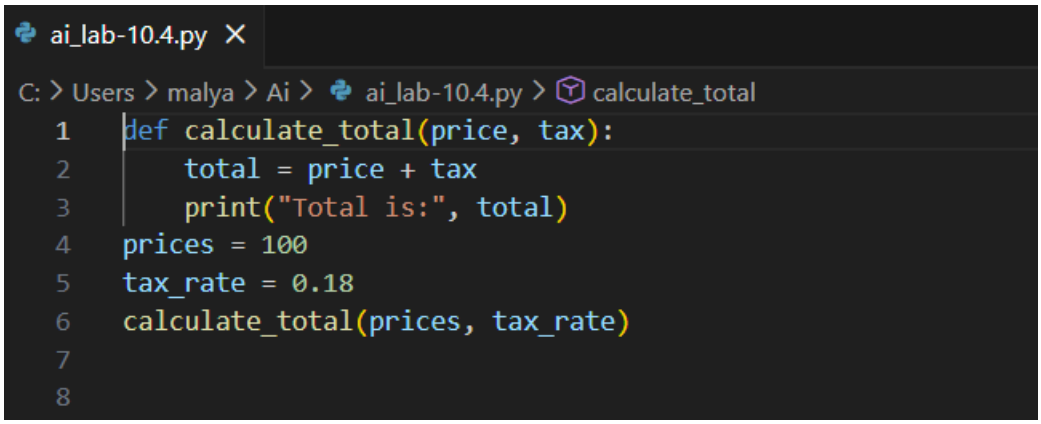
Original Script (With Errors):



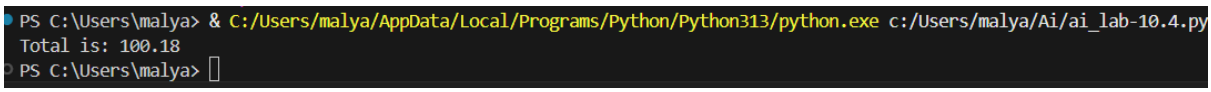
Prompt:

Analyse the above Python script. Identify syntax errors, indentation issues, incorrect variable names, and faulty function calls. Correct the script and explain each fix clearly.

Corrected code:



Output:



Explanation:

Explanation of Fixes:1. Function Definition: The original code had a syntax error in the function definition. The correct syntax for defining a function in Python is `def function_name(parameters): . I added a colon at the end of the function definition line to fix this issue.

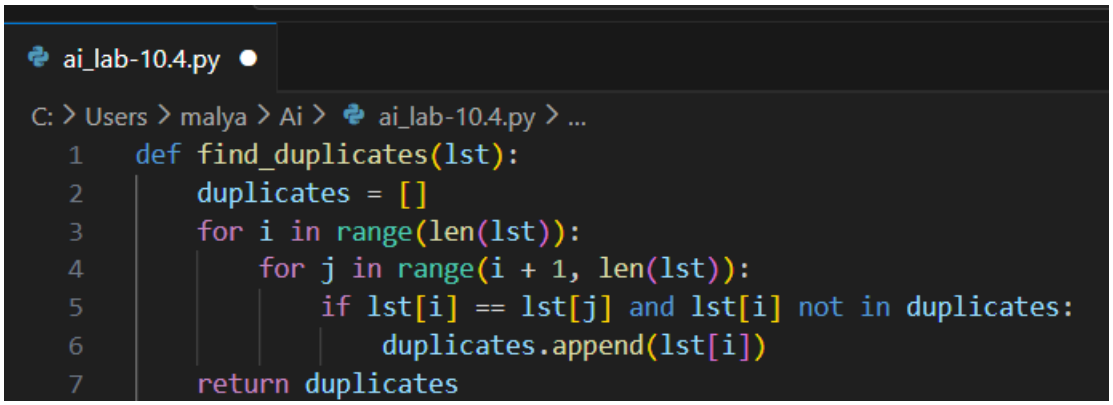
2. Print Statement:In the original code, the print statement was incorrect. It should use a comma to separate the string and the variable, or it can use string formatting. I corrected it to `print("Total is:", total)` to properly display the total amount.

3. Function Call: The original code had a function call `calculateTotal(prices, tax_rate)` which is incorrect because the function name is `calculate_total`. I corrected the function call to `calculate_total(prices, tax_rate)` to match the defined function name.

4. Indentation: The original code had inconsistent indentation. In Python, proper indentation is crucial for defining blocks of code. I ensured that the code inside the function is indented correctly to avoid any indentation errors.

→Task 2: Performance-Oriented Code Review

Original code:



Prompt:

Analyse the above duplicate detection function. Identify performance bottlenecks and refactor it for better time complexity.

Explanation:

The original function has a time complexity of O(n^2) due to the nested loops, which can be inefficient for large lists. To improve the performance, we can use a set to track seen elements and another set to track duplicates. This way, we can achieve a time complexity of O(n).

Corrected code:

Output:

→Task 3: Readability & Maintainability Refactoring

Original code:

Prompt:

Improve this Python code to make it clean and easy to understand.
Fix indentation, use better variable and function names, follow PEP 8 format, and add a short docstring.
Do not change the logic of the program.

Explanation:

This function takes a list of numbers and a threshold value, and returns a new list containing only the numbers that are greater than the threshold.

Parameters:

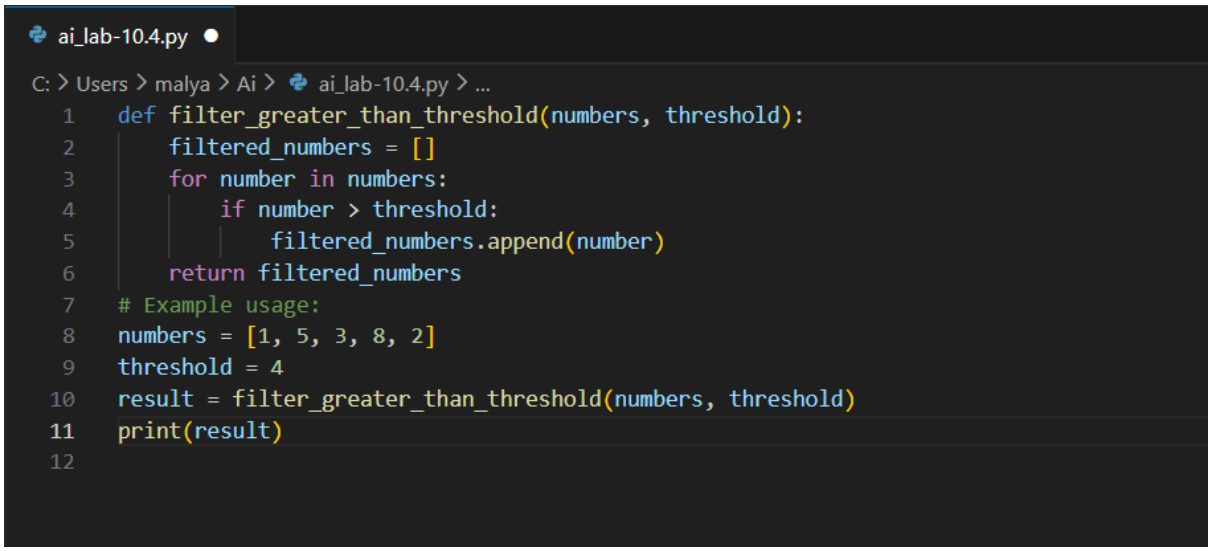
numbers (list): A list of numbers to be filtered.

threshold (int or float): The threshold value to compare against.

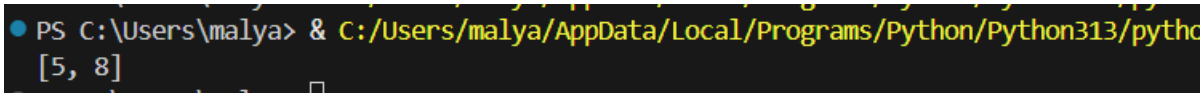
Returns:

list: A list of numbers that are greater than the threshold.

Corrected code:



Output:



→Task 4: Secure Coding and Reliability Review

Original code:

Prompt:

Review the following Python code for security issues and errors.
Fix unsafe SQL queries, add input validation, include proper exception handling, and make the code safe for production use.Keep the original functionality unchanged

Explanation:

The original Python script had security and reliability issues. The main problem was SQL injection due to unsafe string concatenation in the SQL query. This was fixed by using parameterized queries, which treat user input as data and prevent malicious code execution.

Input validation was also added to ensure that only valid and non-empty usernames are processed. Additionally, exception handling was implemented using a try-except block to prevent the program from crashing due to database errors.

Corrected code:

ai_lab-10.4.py

C: > Users > malya > Ai > ai_lab-10.4.py > ...

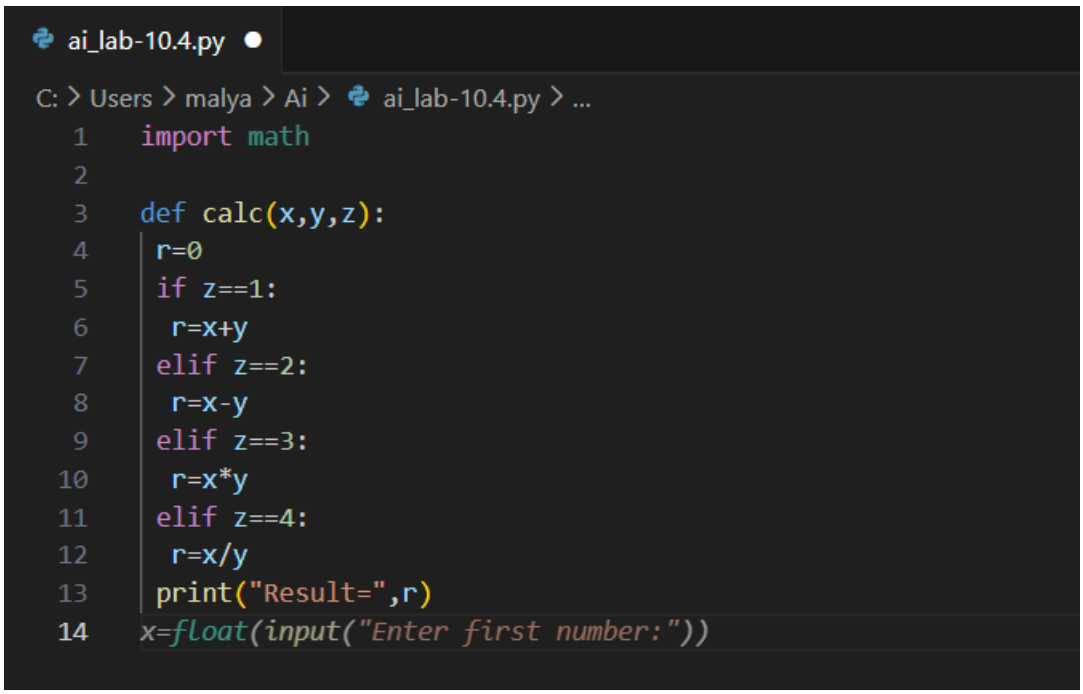
```python
import sqlite3
def get_user(username):
    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()
        query = "SELECT * FROM users WHERE username = ?"
        cursor.execute(query, (username,))
        result = cursor.fetchall()
        return result
    except sqlite3.Error as e:
        print(f"Database error: {e}")
        return None
    except Exception as e:
        print(f"Error: {e}")
        return None
    finally:
        if conn:
            conn.close()
# Example usage
username = "john_doe"
user_data = get_user(username)
if user_data:
    print(f"User data for {username}: {user_data}")
else:    print(f"No data found for user: {username}")

```

Output:

```
PS C:\Users\malya> & C:/Users/malya/AppData/Local/Programs/Python/Python
Database error: no such table: users
No data found for user: john_doe
```

→ Task 5: AI-Based Automated Code Review Report

Original Code:



Prompt:

Perform a structured code review on the above Python script. Evaluate readability, naming conventions, formatting, documentation quality, error handling, and maintainability. Identify code smells and risk areas. Provide improvement suggestions without only fixing the code

Explanation:

1. Readability: The code is relatively simple and straightforward, but it could be improved by adding comments to explain the purpose of the function and the parameters. Additionally, using more descriptive variable names would enhance readability.

2. Naming Conventions: The function name "calc" is not very descriptive. A more meaningful name, such as "perform_operation" or "calculate_result," would better convey the purpose of the function. The variable names "x," "y," and "z" are also not descriptive. Using names like "num1," "num2," and "operation_code" would improve clarity.

3. Formatting: The code is generally well-formatted, but it could benefit from consistent spacing around operators and after commas for better readability.

For example, "r=0" could be written as "r = 0" and "print("Result=",r)" could be written as "print("Result =", r)".

4. Documentation Quality: The code lacks documentation. Adding a docstring at the beginning of the function to explain its purpose, parameters, and return value would enhance the quality of the documentation.

5. Error Handling: The code does not include any error handling. For example, if the user inputs an invalid operation code (not between 1 and 4), the function will not handle it gracefully. Adding error handling to manage invalid inputs would improve the robustness of the code.

6. Maintainability: The code is simple, but it could be made more maintainable by using a dictionary to map operation codes to their corresponding operations. This would make it easier to add new operations in the future without modifying the existing code structure.

Improvement Suggestions:

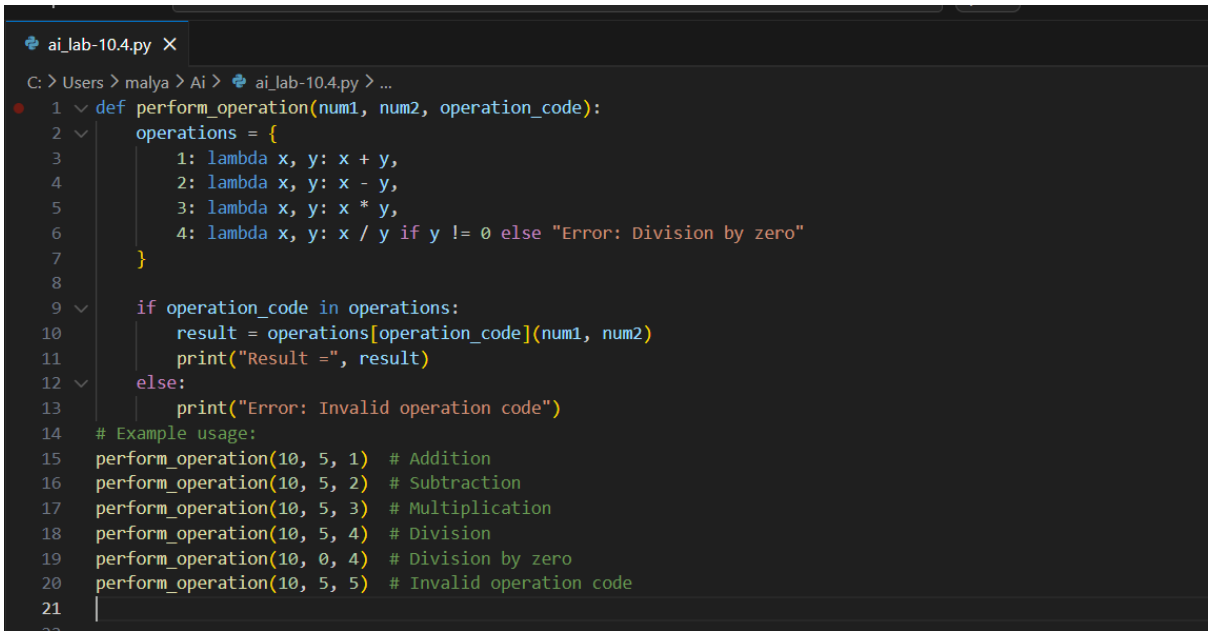
1. Add a docstring to the function to explain its purpose and parameters.

2. Use more descriptive variable names to enhance readability.

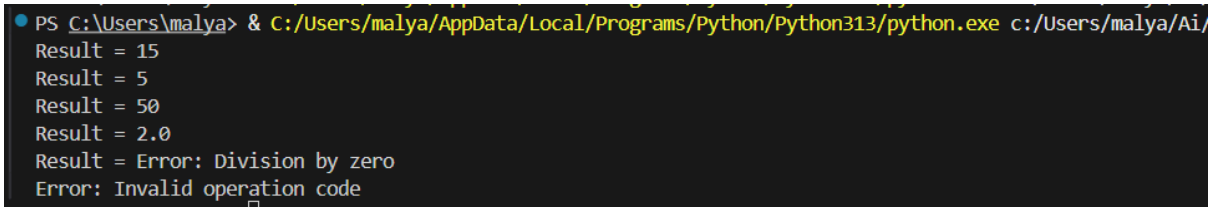
3. Implement error handling to manage invalid operation codes.

4. Use a dictionary to map operation codes to their corresponding operations for better maintainability.

Corrected code:



Output:


PS C:\Users\malya> & C:/Users/malya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/malya/Ai/
Result = 15
Result = 5
Result = 50
Result = 2.0
Result = Error: Division by zero
Error: Invalid operation code