Assignment-10.3

Name: D. Sravika Reddy Hall-Ticket No: 2403a510d0

Batch No: 05 Course: AI Assisted Coding

Task 1: Syntax and Error Detection

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

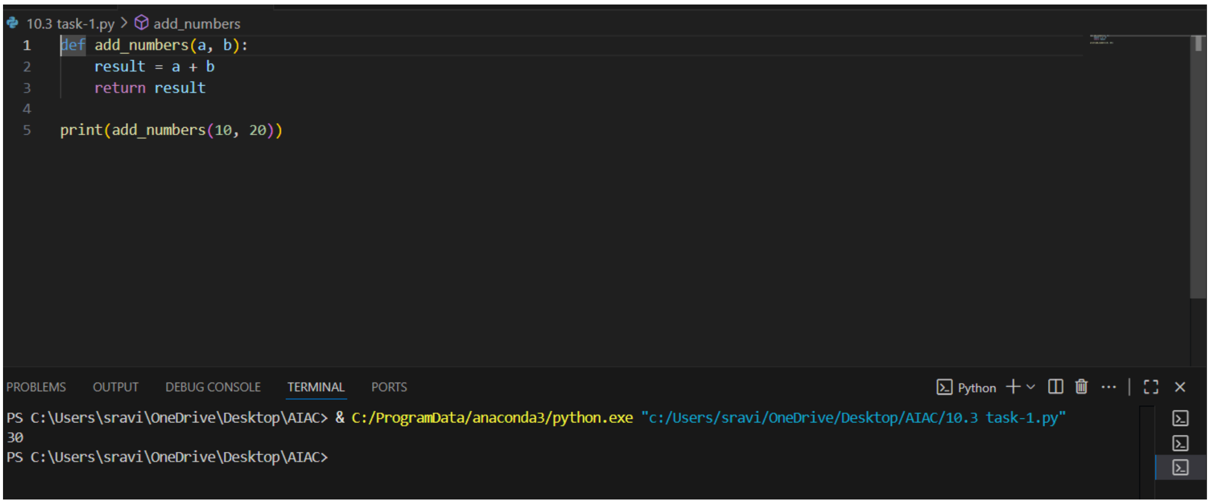
```
# buggy_code_task1.py
def add_numbers(a, b)
result = a + b
return reslt
print(add_numbers(10 20))
```

Expected Output:

- Corrected code with proper syntax (: after function, fixed variable name, corrected function call).
- AI should explain what was fixed.

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

Corrected Code & Output:



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

print(add_numbers(10, 20))
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe "c:/Users/sravi/OneDrive/Desktop/AIAC/10.3 task-1.py"
30
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

Explanation:

- Added a colon : after the function definition (def add_numbers(a, b):).
- Fixed the variable name typo (reslt → result) in the return statement.
- Added a comma between arguments in the function call (add_numbers(10, 20)).

- Properly indented the function body.

Task 2: Logical and Performance Issue Review

Task: 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))
```

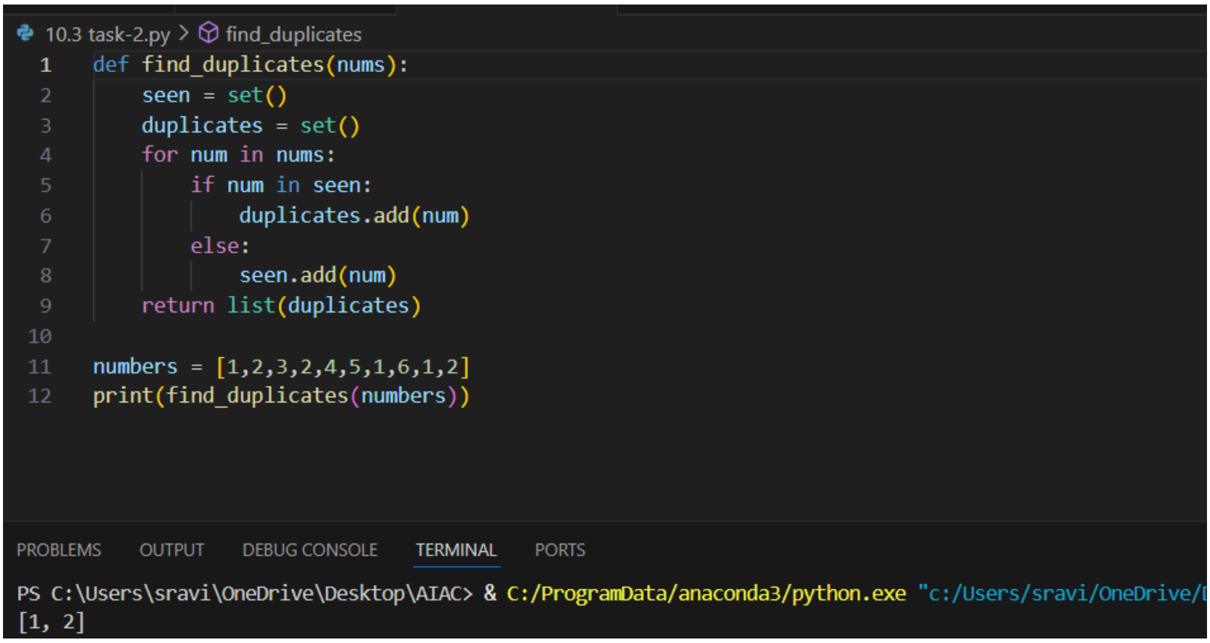
Expected Output:

• More efficient duplicate detection (e.g., using sets).

• AI should explain the optimization

#Prompt: Optimize inefficient logic while keeping the result correct.

Code & Output:



```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))
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe "c:/Users/sravi/OneDrive/
[1, 2]
```

Explanation:

- The original code used nested loops, resulting in O(n^2) time complexity.

- The optimized code uses two sets: seen to track numbers already encountered, and duplicates to store numbers that appear more than once.
- This reduces the time complexity to O(n), making it much faster for large lists.
- The output remains correct: [1, 2] (order may vary due to set usage).

Task 3: Code Refactoring for Readability

Task: 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))
```

Expected Output:

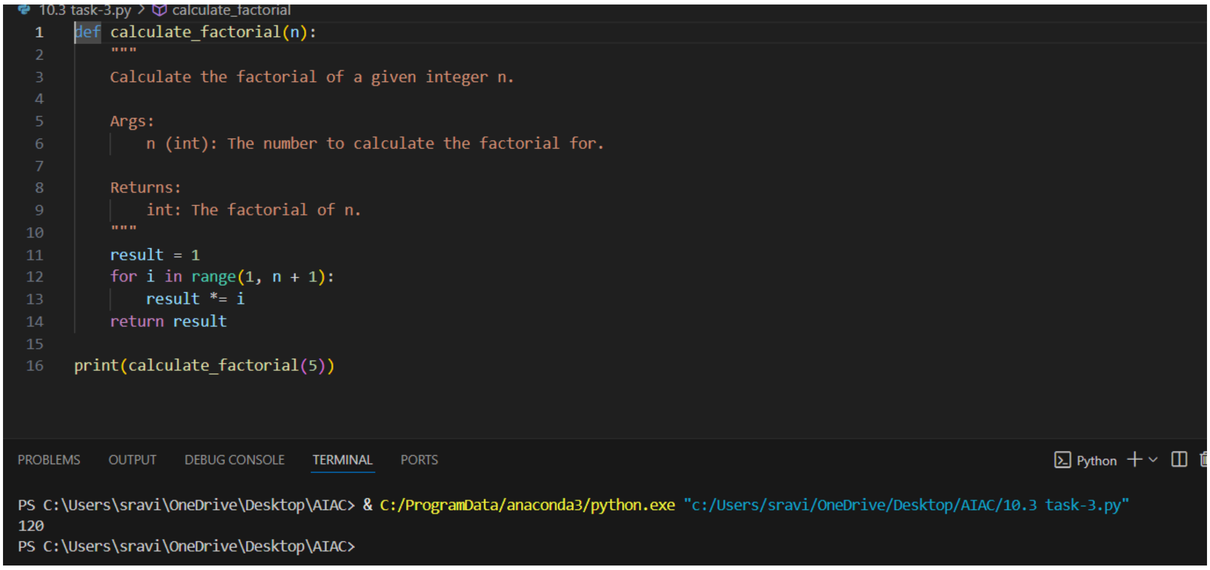
Function renamed to calculate_factorial.

Proper indentation, variable naming, docstrings, and formatting.

AI should provide a more readable version.

#Prompt: Refactor messy code into clean, PEP 8–compliant, well-structured code.

Code & Output:



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

    Args:
        n (int): The number to calculate the factorial for.

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

print(calculate_factorial(5))
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe "c:/Users/sravi/OneDrive/Desktop/AIAC/10.3 task-3.py"
120
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

Explanation:

- Renamed the function to calculate_factorial for clarity.
- Improved variable names (result instead of x, i remains standard for loops).
- Added a docstring describing the function, arguments, and return value.
- Fixed indentation and followed PEP 8 formatting for readability.
- The code is now clean, well-structured, and easy to understand.

Task 4: Security and Error Handling Enhancement

Task: 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))
```

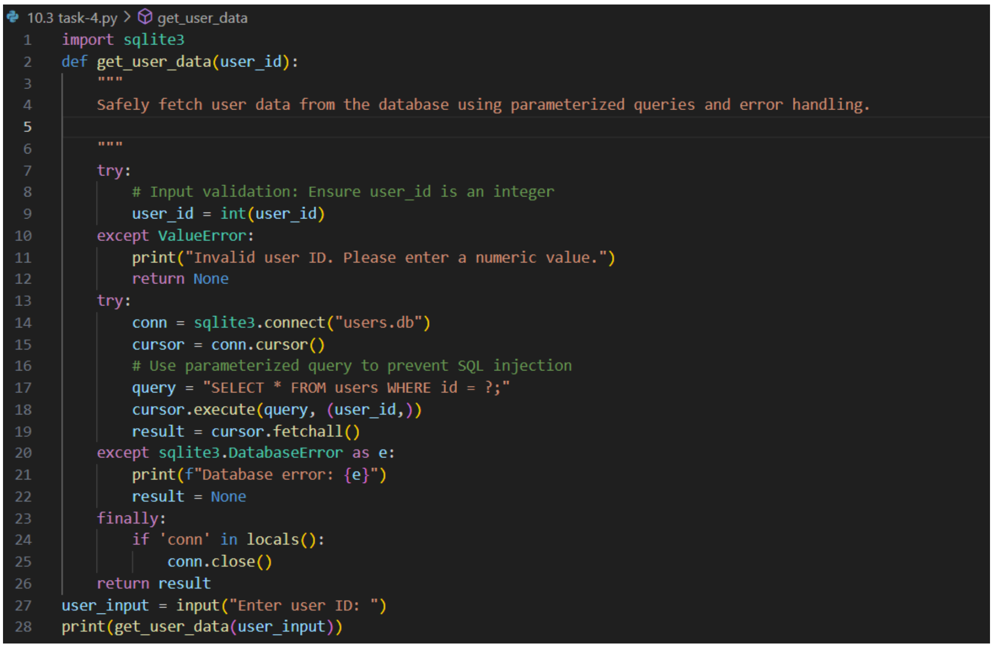
Expected Output:

Safe query using parameterized SQL (? placeholders).

Try-except block for database errors.

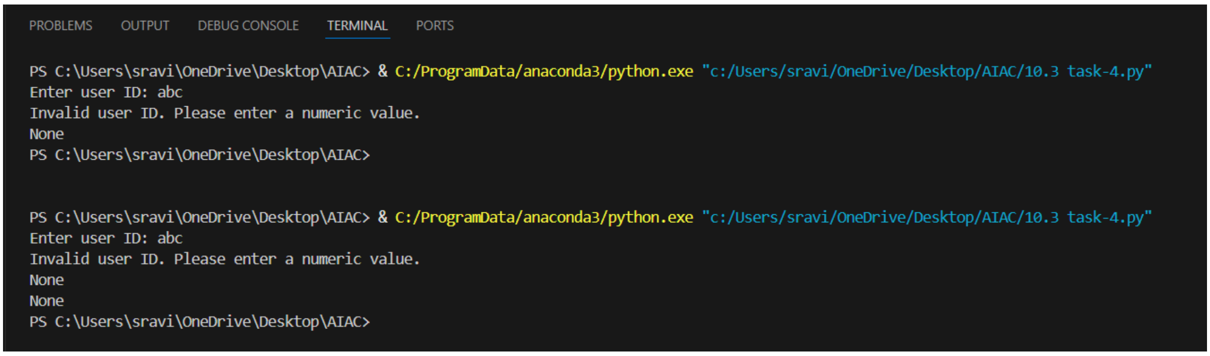
Input validation before query execution

Code & Output:



```python
import sqlite3
def get_user_data(user_id):
    """
    Safely fetch user data from the database using parameterized queries and error handling.

    """
    try:
        # Input validation: Ensure user_id is an integer
        user_id = int(user_id)
    except ValueError:
        print("Invalid user ID. Please enter a numeric value.")
        return None
    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()
        # Use parameterized query to prevent SQL injection
        query = "SELECT * FROM users WHERE id = ?;"
        cursor.execute(query, (user_id,))
        result = cursor.fetchall()
    except sqlite3.DatabaseError as e:
        print(f"Database error: {e}")
        result = None
    finally:
        if 'conn' in locals():
            conn.close()
    return result
user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```



```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe "c:/Users/sravi/OneDrive/Desktop/AIAC/10.3 task-4.py"
Enter user ID: abc
Invalid user ID. Please enter a numeric value.
None
PS C:\Users\sravi\OneDrive\Desktop\AIAC>


PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe "c:/Users/sravi/OneDrive/Desktop/AIAC/10.3 task-4.py"
Enter user ID: abc
Invalid user ID. Please enter a numeric value.
None
None
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

Explanation:

- Uses parameterized SQL (?) to prevent SQL injection.
- Validates input to ensure user_id is an integer.
- Wraps database operations in a try-except block to handle errors gracefully.
- Closes the database connection in a finally block for safety.

Task 5: Automated Code Review Report Generation

Task: Generate a review report for this messy code.

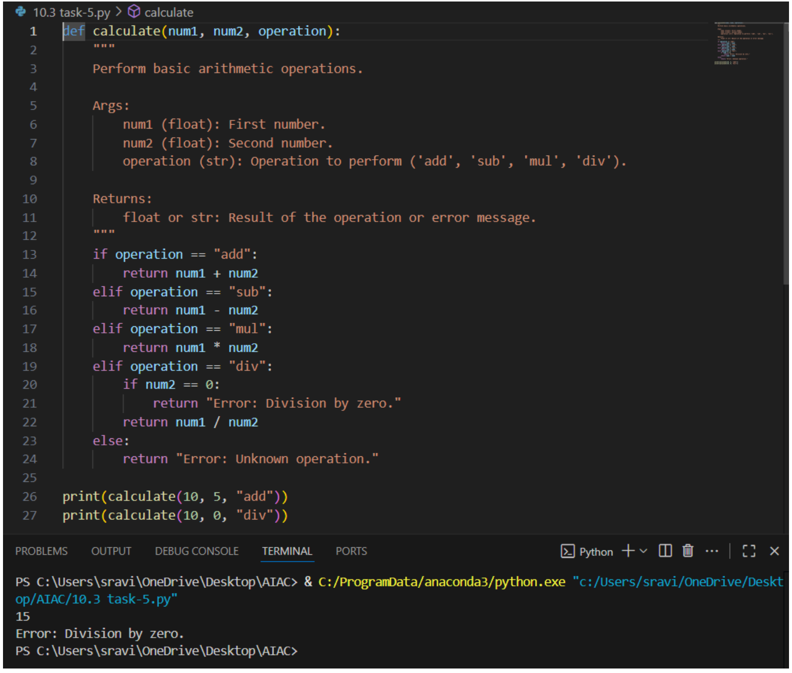
```
# buggy_code_task5.py
def calc(x,y,z):
if z=="add":
return x+y
elif z=="sub": return x-y
elif z=="mul":
return x*y
elif z=="div":
return x/y
else: print("wrong")
print(calc(10,5,"add"))
print(calc(10,0,"div"))
```

Expected Output:

AI-generated review report should mention:

- o Missing docstrings
- o Inconsistent formatting (indentation, inline return)
- o Missing error handling for division by zero
- o Non-descriptive function/variable names
- o Suggestions for readability and PEP 8 compliance

Code & Output:



```python
def calculate(num1, num2, operation):
    """
    Perform basic arithmetic operations.

    Args:
        num1 (float): First number.
        num2 (float): Second number.
        operation (str): Operation to perform ('add', 'sub', 'mul', 'div').

    Returns:
        float or str: Result of the operation or error message.
    """
    if operation == "add":
        return num1 + num2
    elif operation == "sub":
        return num1 - num2
    elif operation == "mul":
        return num1 * num2
    elif operation == "div":
        if num2 == 0:
            return "Error: Division by zero."
        return num1 / num2
    else:
        return "Error: Unknown operation."

print(calculate(10, 5, "add"))
print(calculate(10, 0, "div"))
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe "c:/Users/sravi/OneDrive/Deskt
op/AIAC/10.3 task-5.py"
15
Error: Division by zero.
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

1. **Missing Docstrings:**
    - The function calc does not have a docstring explaining its purpose, parameters, and return value.
2. **Inconsistent Formatting:**
    - Indentation is inconsistent throughout the function.
    - Some return statements are inline with elif, while others are on a new line.
    - The else block uses print instead of return, which is inconsistent with other branches.
3. **Missing Error Handling:**
    - No error handling for division by zero. Calling calc(10, 0, "div") will raise a ZeroDivisionError.
4. **Non-descriptive Names:**
    - The function name calc and variable names x, y, z are not descriptive. More meaningful names should be used (e.g., operation, num1, num2).
5. **Suggestions for Readability and PEP 8 Compliance:**
    - Add a docstring to the function.
    - Use consistent indentation (4 spaces per level).
    - Use descriptive names for the function and parameters.
    - Handle division by zero with a try-except block or conditional check.
    - Return a value in the else block for consistency.
    - Format the code according to PEP 8 guidelines