AI Assisted Coding

Lab Assignment – 10.3

NAME : Neha Shazneen

ROLL NO : 2403A510A1

BATCH : 05

DEPARTMENT : CSE

TASK 1 : Syntax and Error Detection

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 and fix syntax, indentation, and variable errors in the given Python script. Provide the corrected code and explain the changes made.

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

Code Generated :

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

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

Explanation of fixes:

Added a colon (:) After the function definition.

Fixed indentation for the function body.

Corrected the variable name from reslt to result in the return statement.

Added a comma between 10 and 20 in the function call.

Output :

```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehvee
n\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Pytho
n\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-w
in32-x64\bundled\libs\debugpy\launcher' '59684' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI
 associated coding\10.3t1.py'
30
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
```

Observation :

The file contains a simple Python function add_numbers() that performs basic addition and returns the result, followed by a function call that prints the output.The code appears to be a basic example for syntax and error detection, with proper Python syntax and no apparent errors in the current implementation.

TASK 2 : Logical and Performance Issue Review

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 :

Optimize the following Python function to efficiently find all duplicate elements in a list, ensuring each duplicate appears only once in the result. The current implementation uses nested loops and is slow for large lists. Keep the output correct.

Instructions:

- Remove unnecessary nested loops.
- Ensure the function returns a list of duplicate values, each only once.
- Optimize for performance.

Code Generated :

10.3t2.py > ...

```python
# 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))
```

Output :

```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '55372' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding\10.3t2.py'
[1, 2]
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
```

Observation :

- The code uses nested loops to compare every pair of elements in the list, resulting in $O(n^2)$ time complexity, which is inefficient for large lists.
- It checks if two elements are equal and their indices are different, then adds the element to the duplicates list if it's not already present.
- The logic is correct for finding duplicates, but inefficient.
- The output for the given input is [2, 1], which is correct, but the approach can be optimized using a dictionary or set to reduce time complexity.

TASK 3 : Code Refactoring for Readability

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 :

Write a Python program that calculates the factorial of a given number.

- Define a function that takes an integer n as input.
- The function should return the factorial of n (i.e., the product of all positive integers from 1 to n).
- Ensure the code follows PEP 8 style guidelines (proper naming, indentation, and readability).
- Add a docstring to describe the function.
- Test your function by calculating the factorial of 5.

Code Generated :

10.3t3.py > ...

```python
def factorial(n: int) -> int:
    """
    Calculates the factorial of a given positive integer n.

    Args:
        n (int): The integer for which the factorial is to be calculated. Must be >= 0.

    Returns:
        int: The factorial of n.

    Example:
        result = factorial(5)
    """
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Test the function by calculating the factorial of 5
print(factorial(5))
```

Output :

```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI a
ssociated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\PythoPS C:\Users\Summya Mehveen\OneDrive\Desktop\AI
 associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\
AppData\Local\Programs\Python\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\
in32-x64\bundled\libs\debugpy\launcher' '64689' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding\10.3
t3.py'
120
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
```

Observation :

1. Functionality:

o The function factorial(n: int) -> int correctly calculates the factorial of a given non-negative integer n.

o For n = 5, it returns 120, which is the expected result.

2. Code Quality & Readability:

o The code follows PEP 8 guidelines (proper indentation, spacing, meaningful variable and function names).

o The function name factorial is descriptive and self-explanatory.

o The loop implementation is clear and easy to understand.

3. Documentation:

o A docstring is provided with a clear explanation of:

1. What the function does.

2. Argument type and expected constraints (n >= 0).

3. Return value.

4. Example usage.

o This makes the code easy to understand and maintain.

4. Correctness:

o The code produces the correct output for the given test case (factorial(5) → 120).

5. Possible Improvements:

o No explicit error handling for negative numbers. If n < 0, the code will return 1, which is not mathematically correct. Adding input validation would improve robustness.

o Could also demonstrate recursion as an alternative approach for factorial.

TASK 4 : Security and Error Handling Enhancement

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 :

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))rewrite code with the following improvements:

Use parameterized SQL queries with ? placeholders.

Add try-except-finally blocks to handle database errors safely.

Validate user input so only numeric IDs are allowed.

Ensure the connection is always closed properly.

Code Generated :

10.3t4.py > ...

```python
import sqlite3

def get_user_data(user_id):
    """
    Retrieve user data from the database using a safe parameterized query.

    Args:
        user_id (str): The user ID to search for (must be numeric)

    Returns:
        list: List of tuples containing user data, or empty list if no data found

    Raises:
        ValueError: If user_id is not numeric
        sqlite3.Error: If database operation fails
    """
    # Validate input - ensure only numeric IDs are allowed
    if not user_id.isdigit():
        raise ValueError("User ID must be numeric")

    conn = None
    try:
        # Establish database connection
        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()
        return result

```

```
    except sqlite3.Error as e:
        print(f"Database error occurred: {e}")
        return []
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return []
    finally:
        # Ensure connection is always closed properly
        if conn:
            conn.close()

def main():
    """
    Main function to handle user input and display results.
    """
    try:
        user_input = input("Enter user ID: ")
        result = get_user_data(user_input)

        if result:
            print("User data found:")
            for row in result:
                print(row)
        else:
            print("No user data found or an error occurred.")

    except ValueError as e:
        print(f"Input error: {e}")
    except KeyboardInterrupt:
        print("\nOperation cancelled by user.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

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

Output :

```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Meh
veen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Progran\
n\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.
n\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.
0-win32-x64\bundled\libs\debugpy\launcher' '58848' '--' 'c:\Users\Summya'
Enter user ID: 2403a510a1@sru.edu.in
Input error: User ID must be numeric
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
```

Observation :

Security Enhancement: The code now uses parameterized queries (? placeholders) instead of string formatting, completely eliminating the SQL injection vulnerability that existed in the original code where user input was directly interpolated into the SQL string.

Robust Error Handling: The implementation includes comprehensive try-except-finally blocks with proper connection cleanup, input validation for numeric-only user IDs, and graceful error handling that prevents the application from crashing while providing meaningful error messages to users.

TASK 5 : Automated Code Review Report Generation

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 :

Review the following Python function for code quality, correctness, and robustness. Identify issues such as formatting, error handling, and edge cases. Suggest improvements to make the code cleaner and safer.

Instructions:

- Point out formatting and indentation problems.
- Check for missing error handling (e.g., division by zero).
- Suggest improvements for code readability and robustness.

- Comment on the use of print statements versus return values.

Code Generated :

103t5.py > ...

```python
# 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":
        if y == 0:
            return "Error: Division by zero"
        return x / y
    else:
        return "Error: Invalid operation"

print(calc(10, 5, "add"))
```

Output :

```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '57545' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding\103t5.py'
15
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
```

Observation :

- The code is now properly formatted and readable.
- The function calc handles four operations: add, sub, mul, and div.
- Division by zero is handled gracefully, returning an error message.
- Invalid operations return an error message.
- Only one test case (calc(10, 5, "add")) is printed; other operations are not tested.
- The function is robust and safe for basic arithmetic operations.