ASSIGNMENT-10.4

Name: B. shivasai

HT NO.:2403A51257

Batch No.:11

TEST – 1

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

ASSIG-10.4 > TASK1 > ...

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

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

OUTPUT:

```
PS C:\AI> python -u "c:\AI\ASSIG-10.4\TASK1"
30
PS C:\AI>
```

Explanation of fixes:

- Added missing colon (:) after the function definition line.
- Indented the function body correctly (2 spaces or 4 spaces).
- Corrected the variable name from reslt to result.
- Added missing comma between arguments in the function call add_numbers(10, 20).

TEST – 2

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.

```
ASSIG-10.4 > TASK2 > ...
def find_duplicates(nums):
    """
    Returns a list of duplicate elements in nums.
    Optimized using sets for efficient lookup.
    """
    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:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\AI> python -u "c:\AI\ASSIG-10.4\TASK2"
[1, 2]
PS C:\AI>
```

Explanation :

- Original approach: Uses nested loops (O(n^2)) comparing every element with every other element, which is inefficient for large lists.
- New approach: Uses two sets:
  - seen to keep track of elements already encountered.
  - duplicates to store elements that appear more than once.
- This reduces the complexity to **O(n)** because each element is processed only once.

- Sets provide **O(1)** average-time complexity for membership tests, making the duplicate detection much faster.
- Finally, the duplicates set is converted back to a list for output.

TEST – 3

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.

TASK-3 > ...

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

    Args:
        n (int): The number to compute 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))
```

OUTPUT:

```
PS C:\AI> python -u "c:\AI\TASK-3"
120
PS C:\AI>
```

Task – 4

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.

TASK-4 > ...

```python
import sqlite3

def initialize_db():
    """
    Creates the users table and inserts sample data if the table is empty.
    """
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        email TEXT NOT NULL
    )
    ''')

    # Insert sample data only if table is empty
    cursor.execute("SELECT COUNT(*) FROM users")
    if cursor.fetchone()[0] == 0:
        cursor.execute("INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com')")
        cursor.execute("INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com')")
        conn.commit()

    conn.close()
```

TASK-4 > ...

```python
def initialize_db():
    print("Database initialized with sample data.")


def get_user_data(user_id: int):
    """
    Fetches user data for the given user_id from the database.

    Args:
        user_id (int): The ID of the user to fetch.

    Returns:
        list of tuples: List of user records matching the user_id.
    """
    try:
        conn = sqlite3.connect('users.db')
        cursor = conn.cursor()

        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: {e}")
```

```python
TASK-4 > main
def get_user_data(user_id: int):
        print(f"Database error: {e}")
        return []

    finally:
        if conn:
            conn.close()


def main():
    initialize_db()

    user_input = input("Enter user ID: ")

    if user_input.isdigit():
        user_id = int(user_input)
        data = get_user_data(user_id)
        if data:
            print("User data:", data)
        else:
            print("No user found with that ID.")
    else:
        print("Invalid input. Please enter a numeric user ID.")
if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS C:\AI> python -u "c:\AI\TASK-4"
Enter user ID: 1
Database error: no such table: users
[]
PS C:\AI> 3
3
PS C:\AI> python -u "c:\AI\TASK-4"
Enter user ID: 123
Database error: no such table: users
[]
PS C:\AI> python -u "c:\AI\TASK-4"
Enter user ID: adc
Invalid user ID. Please enter a valid integer.
PS C:\AI> python -u "c:\AI\TASK-4"
Enter user ID: 3
Database error: no such table: users
[]
PS C:\AI>
```

TEST – 5

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

task4 task1 task5 TASK1 TASK5 TASK2 tasl

ASSIG-10.4 > TASK5 > calculate

```python
def calculate(a, b, operation):
    """
    Perform a basic arithmetic operation on two numbers.

    Args:
        a (float): First operand.
        b (float): Second operand.
        operation (str): Operation to perform ('add', 'sub', 'mul', 'div').

    Returns:
        float: Result of the operation, or None if invalid.
    """
    if operation == "add":
        result = a + b
        print(f"Adding {a} + {b} = {result}")
        return result
    elif operation == "sub":
        result = a - b
        print(f"Subtracting {a} - {b} = {result}")
        return result
    elif operation == "mul":
        result = a * b
        print(f"Multiplying {a} * {b} = {result}")
        return result
    elif operation == "div":
```

ASSIG-10.4 > TASK5 > calculate

```python
def calculate(a, b, operation):
    elif operation == "sub":
        result = a - b
        print(f"Subtracting {a} - {b} = {result}")
        return result
    elif operation == "mul":
        result = a * b
        print(f"Multiplying {a} * {b} = {result}")
        return result
    elif operation == "div":
        if b == 0:
            print("Error: Division by zero.")
            return None
        result = a / b
        print(f"Dividing {a} / {b} = {result}")
        return result
    else:
        print("Error: Unknown operation.")
        return None

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

OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Code

```
PS C:\AI> python -u "c:\AI\ASSIG-10.4\TASK5"
Adding 10 + 5 = 15
15
Error: Division by zero.
None
PS C:\AI>
```