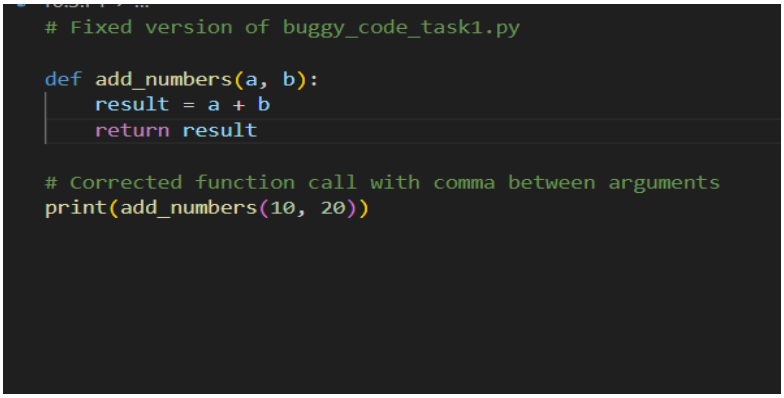
Q.NO:	**NAME:ANUSHA PEDDAPELLI** **ROLL.NO:2403A51103** **BATCH NO:06** **ASSIGNMENT 10.3** **Task 1: Syntax and Error Detection** **Task:** Identify and fix syntax, indentation, and variable errors in the given script. # buggy_code_task1.py def add_numbers(a, b) result = a + b return reslt print(add_numbers(10 20)) **Expected Output**: • Corrected code with proper syntax (: after function, fixed variable name, corrected function call). • AI should explain what was fixed. **PROMPT:** You are given a buggy Python script. Identify and fix syntax, indentation, and variable errors. Ensure the function works properly and produces the correct output. **CODE:**  **OBSERVATION:** □ Function definition error → Missing colon (:) after def add_numbers(a, b). □ Indentation issue → result = a + b and return result must be indented inside the function.	Week5 - Thursday

 Variable name mismatch → reslt was undefined; corrected to result.
 Function call error → Arguments were written as 10 20 without **a** comma; fixed to 10, 20.

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))
```

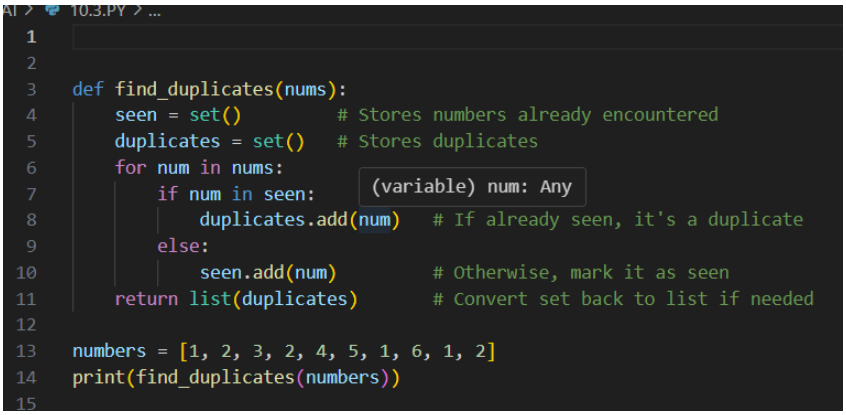
Expected Output:

- More efficient duplicate detection (e.g., using sets).
- AI should explain the optimization.

PROMPT:

You are given a Python script with inefficient duplicate detection logic. Optimize the code to improve performance while keeping the output correct. Use better data structures (like sets) instead of nested loops.

CODE:



OBSERVATION:

 Removed nested loops → Original code checked every element against every other, making it **O(n²)**.

 Used sets for efficiency → Set lookup is **O(1)** on average, reducing

	overall complexity to **O(n)**.
	  Avoided repeated checks → duplicates.add(num) ensures no duplicates are added multiple times.
	  Result correctness → Output remains the same ([1, 2] or order may vary since sets are unordered	

	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:
	You are given a messy Python script that calculates a factorial. Refactor the code to follow **PEP 8 standards** with proper naming, indentation, docstrings, and readability.
	CODE:	

```python
# Refactored version of buggy_code_task3.py

def calculate_factorial(n):
    """
    Calculate the factorial of a given number.

    Args:
        n (int): A non-negative integer.

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


# Example usage
print(calculate_factorial(5))
print(calculate_factorial(0))  # Edge case: factorial of 0
```

OBSERVATION:

 Function name improved → Renamed c to calculate_factorial (clearer, self-explanatory).

 Variable naming → Changed x → result for readability.

 Indentation fixed → Proper 4-space indentation per **PEP 8**.

 Docstring added → Clear explanation of function purpose, parameters, and return type.

 Code formatting → Blank lines and spacing make it structured and readable.

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.

PROMPT:

You are given a Python script that retrieves user data from a database.

The code is vulnerable to **SQL injection** and lacks **error handling**.

Refactor the script to:

- Use **parameterized queries** (? placeholders).
- Add **try-except blocks** for database error handling.
- Validate

CODE:

```
10.3.PY > ...
# Refactored version of buggy_code_task4.py

import sqlite3

def get_user_data(user_id):
    """
    Fetch user data securely from the database using parameterized queries.

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

    Returns:
        list: List of tuples containing user data.
    """
    try:
        # Ensure input is an integer
        user_id = int
```

OBSERVATION:'

 SQL Injection Fixed → Replaced f-string with parameterized query (?).

	Input Validation → Converted user_input to int; handled ValueError if input isn't numeric. **Error Handling Added** → Wrapped DB operations in try-except-finally. **Connection Safety** → finally ensures database connection closes properly. **Docstring Added** → Clear explanation of function purpose and parameters.	

	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
	Prompt:
	You are given a messy Python script that performs basic arithmetic operations. Perform an automated code review and generate a clean, PEP 8–compliant version. The review should address:	

- Adding **docstrings**.
- Improving **function and variable naming**.
- Consistent **indentation and formatting**.
- Adding **error handling** (e.g., division by zero).
- Enhancing **readability** and **maintainability**.

CODE:

```
# Refactored version of buggy_code_task4.py

import sqlite3

def get_user_data(user_id):
    """
    Fetch user data securely from the database using parameterized queries

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

    Returns:
        list: List of tuples containing user data.
    """
    try:
        # Ensure input is an integer
        user_id = int
```

OBSERVATION:

 Missing docstrings → Added detailed docstring with Args, Returns, and Raises.

 Non-descriptive names → Renamed calc → calculate and z → operation for clarity.

 Inconsistent formatting → Fixed indentation and avoided inline returns.

 Error handling added → Handled **division by zero** using ZeroDivisionError.

 Readability improved → Code now follows **PEP 8** with spacing and blank lines.

 Maintainability → Structured errors using raise ValueError for invalid operations.