# ASSIGNMENT 10.3

NAME:P. Umesh Reddy ROLL NO: 2403A510F9

BATCH NO: 06 COURSE: AI Assisted Coding

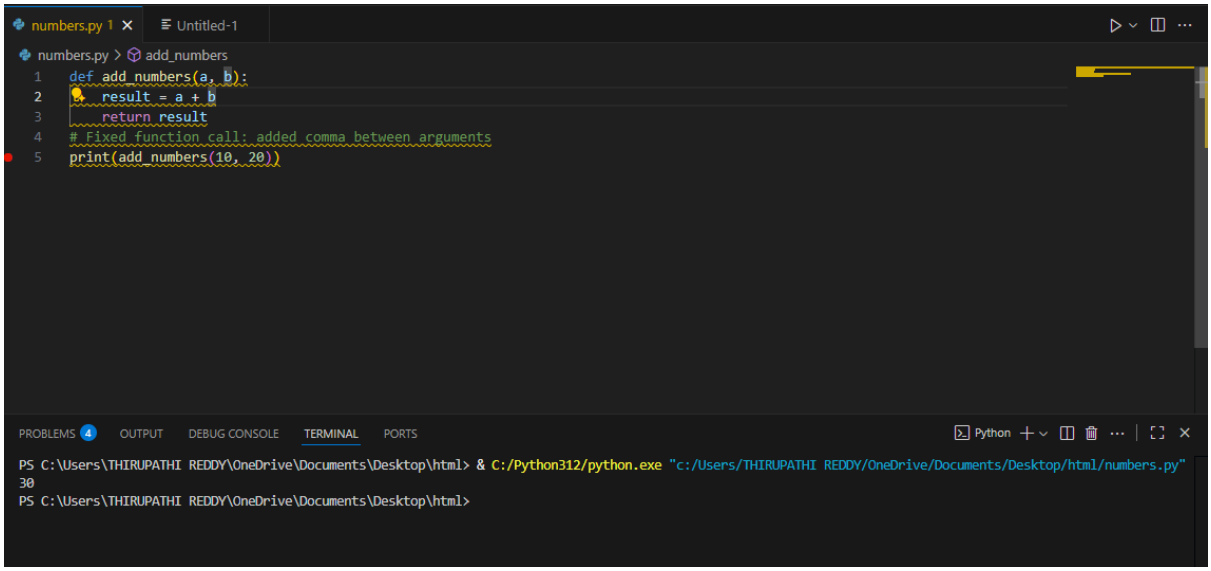
*Task 1: Syntax and Error Detection
Task: Identify and fix syntax, indentation, and variable errors in the given script.

```
# 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.

#Out Put:




Explanation:

*Added a colon (:) after the function definition.

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

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

*Indented the function body properly.

*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.

#Out Put:

```python
def find_duplicates(nums):
    # Use sets for efficient duplicate detection
    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))
```

```
PROBLEMS 9   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/test_square.py"
[1, 2]
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html>
```

Explanation:

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

*The optimized version uses sets to track seen numbers and duplicates, reducing the time complexity to O(n).

*This approach ensures each duplicate is only added once and improves performance for large lists.

*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.

#Out Put:

```python
def calculate_factorial(n):
    """
    Calculate the factorial of a given number n.
    Returns the factorial as an integer.
    """
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

print(calculate_factorial(5))
```

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/test_square.py"
120
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html>
```

Explanation:

*Renamed function to calculate_factorial for clarity.

*Added a docstring describing the function.

*Improved variable names (result instead of x).

*Fixed indentation and formatting to follow PEP 8 standards.

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: Add security practices and exception handling to the code.

#Out Put:

```python
import sqlite3

def get_user_data(user_id):
    # Input validation: ensure user_id is an integer
    try:
        user_id = int(user_id)
    except ValueError:
        print("Invalid user ID. Must be an integer.")
        return []

    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 = []
    finally:
        conn.close()
    return result

user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```

```
PROBLEMS 9   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/test_square.py"
Enter user ID:
```

Explanation:

*Uses parameterized SQL queries to prevent SQL injection.

*Validates input to ensure it’s an integer.

*Adds try-except blocks for error handling and safe resource management.

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: Generate a review report for this messy code.

#Out Put:

```python
def calculate(num1, num2, operation):
    """
    Perform basic arithmetic operations: add, sub, mul, div.
    Returns the result or an 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"

# Example calls to see output
print(calculate(10, 5, "add"))   # Output: 15
print(calculate(10, 5, "sub"))   # Output: 5
print(calculate(10, 5, "mul"))   # Output: 50
print(calculate(10, 0, "div"))   # Output: Error: Division by zero
print(calculate(10, 5, "mod"))   # Output: Error: Unknown operation
```

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
15
5
50
Error: Division by zero
Error: Unknown operation
```

Explanation:

- **Function Purpose:**
  The calculate function performs basic arithmetic operations (add, sub, mul, div) on two numbers and returns the result. If the operation is not recognized or division by zero is attempted, it returns an error message.
- **Parameters:**
  - num1: The first number (integer or float).
  - num2: The second number (integer or float).

    - operation: A string specifying the operation ("add", "sub", "mul", "div").
- **Logic:**
    - Checks the value of operation and performs the corresponding arithmetic.
    - For division, it checks if num2 is zero to avoid a runtime error and returns an error message if so.
    - If the operation is not one of the recognized strings, it returns "Error: Unknown operation".
- **Example Calls:**
    - Demonstrates usage of the function with different operations and prints the results.
    - Shows correct results for valid operations and appropriate error messages for invalid operations or division by zero.
- **Comments:**
    - The code is well-structured, readable, and handles errors gracefully