AI Assisted Coding

## Assignment-10.4 K.Sindhu meenan(2403A51250)-Batch11

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:

TASK: Fix the syntax errors, indentation, and variable names in the code below.

The corrected function should return the sum of two numbers. Also correct the function call to print the result of adding 10 and 20.

CODE AND THE OUTPUT:

A screenshot of a computer program

AI-generated content may be incorrect.

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:

TASK: Optimize the find\_duplicates function below.

Requirements:

- Return the same duplicates as before

- Use sets or other efficient logic to improve performance to O(n)

- Keep output as a list of duplicates.

Buggy code:

A black background with blue and yellow text

AI-generated content may be incorrect.

Corrected code:

A black screen with yellow and green text

AI-generated content may be incorrect.

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:

TASK: Refactor the factorial function below for readability:

- Rename function to calculate\_factorial

- Follow PEP 8 formatting and indentation

- Use descriptive variable names

- Add a Google-style docstring and type hints

- Include error handling for negative input.

Messy code:

A black rectangular object with white text

AI-generated content may be incorrect.

Corrected code:

A screenshot of a computer program

AI-generated content may be incorrect.

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:

TASK: Enhance security and error handling in the function below:

- Use parameterized SQL instead of f-strings to prevent injection

- Add try/except/finally for database connection and errors

- Validate that user input is an integer before executing the query

- Use clear function and variable names, plus docstrings.

Messy code:

A computer screen shot of text

AI-generated content may be incorrect.

Problems with messy code:

* **SQL injection risk** — f-string directly inserting user\_id.
* **No input validation** — user\_id might not be an integer.
* **No exception handling** — DB errors could crash the program.

Updated code:

A screen shot of a computer

AI-generated content may be incorrect.

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:

TASK: Perform an automated code review of the following function.

- Identify missing docstrings

- Identify inconsistent formatting and indentation

- Identify missing error handling (like division by zero)

- Identify non-descriptive names

- Suggest improvements for readability and PEP 8 compliance.

Messy code:

A black rectangular object with white text

AI-generated content may be incorrect.

Updated code:

A screenshot of a computer program

AI-generated content may be incorrect.

Report:

1. **Missing Docstrings**
   * The function [calc](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) lacks a docstring explaining its purpose, parameters, and return value.
2. **Inconsistent Formatting and Indentation**
   * The function body is not properly indented.
   * The [elif](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") and [else](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) statements are not consistently formatted.
   * The [print](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) statements are not separated from the function definition.
3. **Missing Error Handling**
   * Division by zero is not handled, which will raise a ZeroDivisionError if [y](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) is zero and [z](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) is "div".
4. **Non-descriptive Names**
   * The function name [calc](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) and parameter names [x](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [y](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), and [z](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) are not descriptive. More meaningful names like a, b, and operation would improve readability.
5. **Other Suggestions**
   * Use return instead of [print](vscode-file://vscode-app/c:/Users/siris/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) for error messages to keep function output consistent.
   * Consider using a dictionary to map operations to functions for scalability.
   * Follow PEP 8: add spaces after commas, use lowercase with underscores for function names, and add two blank lines before function definitions.

**Suggested Improvements**

* Add a docstring.
* Use descriptive parameter names.
* Properly indent the function.
* Add error handling for division by zero.
* Use PEP 8 formatting.
* Return error messages instead of printing them inside the function.