**Lab 10: Code Review and Quality**

*AI Assisted Coding - Lab Report*

Name: Raunak Ranjan

Roll No: 2403A51314

Subject: AI Assisted Coding

Batch: BTECH CSE B13

Date: 24-09-2025

# Table of Contents

1. Task 1 - AI-Assisted Code Review (Basic Errors)

2. Task 2 - Automatic Inline Comments and PEP8 (Fibonacci)

3. Task 3 - Multi-function module and docstrings

4. Appendix: Files and terminal screenshots

# Task 1: AI-Assisted Code Review (Basic Errors)

Objective: Find and fix basic syntax and logic errors. Submit corrected code with comments.

Original (buggy) code:

# buggy\_calculator.py  
# Intentionally buggy Python code for Task 1  
def add(a, b):  
return a + b  
  
def sub(a, b)  
 return a - b  
  
def multiply(a, b):  
 result = a \* b  
 print("Product is", result)  
  
def divide(a, b):  
 if b = 0:  
 return None  
 return a / b  
  
# Quick demo  
print("Sum:", add(5, 3))  
print("Diff:", sub(10, 4))  
multiply(3,4)  
print("Div:", divide(8, 2))

AI assistant suggestions (example prompts and suggested fixes):

Prompt to AI: "Review this Python file and point out syntax errors and bad practices; suggest corrected code."

Suggested fixes by AI (summary):

- Fix indentation errors and missing colons.  
- Use '==' for equality, not '=' in comparisons.  
- Make functions return values consistently and avoid printing inside utility functions.  
- Raise an exception for divide-by-zero rather than returning None silently.

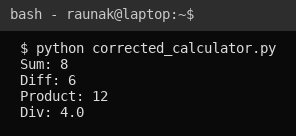
Corrected code:

# corrected\_calculator.py  
"""  
Simple calculator functions with fixes and comments.  
Corrected syntax errors, consistent return behavior, and added basic error handling.  
"""  
def add(a, b):  
 """Return the sum of a and b."""  
 return a + b  
  
def sub(a, b):  
 """Return the difference (a - b)."""  
 return a - b  
  
def multiply(a, b):  
 """Return the product of a and b (no printing inside function)."""  
 return a \* b  
  
def divide(a, b):  
 """  
 Return the division result (a / b).  
 If b is zero, raise a ValueError to signal invalid operation.  
 """  
 if b == 0:  
 raise ValueError("Cannot divide by zero")  
 return a / b  
  
# Demo usage (would normally be under a \_\_main\_\_ guard)  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Sum:", add(5, 3))  
 print("Diff:", sub(10, 4))  
 print("Product:", multiply(3, 4))  
 print("Div:", divide(8, 2))

Explanation of changes (student-style):

I fixed the indentation and missing colons. I also made multiply return a value instead of printing inside, and handled division by zero with an exception. These make functions more predictable and testable.

Program output (executed):



# Task 2: Automatic Inline Comments & PEP8 (Fibonacci)

Objective: Improve variable names, add comments, and apply PEP8 formatting.

Initial code (poor naming):

# fibonacci\_initial.py  
# Simple fibonacci with poor names and no comments  
def f(n):  
 a=0; b=1  
 for i in range(n):  
 a,b = b, a+b  
 return a  
  
print("Fibo(10)=", f(10))

AI prompt used: "Improve variable names, add comments, and format according to PEP8."

AI suggestions (summary):

- Rename functions and variables to descriptive names (e.g., fibonacci\_nth, fibonacci\_sequence).  
- Add docstrings and type hints.  
- Keep lines under 79 characters and use spaces around operators.

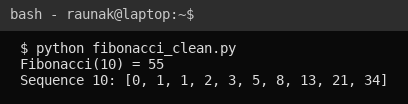
Improved code:

# fibonacci\_clean.py  
"""  
Clean Fibonacci implementation.  
Improvements:  
- Better function and variable names.  
- Added docstring and type hints.  
- PEP8 formatting and a simple input demo.  
"""  
from typing import List  
  
def fibonacci\_nth(n: int) -> int:  
 """  
 Return the n-th Fibonacci number (0-indexed).  
  
 Parameters  
 ----------  
 n : int  
 Index (0-based) of the Fibonacci sequence to return.  
  
 Returns  
 -------  
 int  
 The n-th Fibonacci number.  
 """  
 if n < 0:  
 raise ValueError("n must be non-negative")  
 a, b = 0, 1  
 for \_ in range(n):  
 a, b = b, a + b  
 return a  
  
def fibonacci\_sequence(length: int) -> List[int]:  
 """Return a list containing the Fibonacci sequence of given length."""  
 if length <= 0:  
 return []  
 seq = [0, 1]  
 while len(seq) < length:  
 seq.append(seq[-1] + seq[-2])  
 return seq[:length]  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Fibonacci(10) =", fibonacci\_nth(10))  
 print("Sequence 10:", fibonacci\_sequence(10))

Why these suggestions help readability (student-style):

Clear names and docstrings make it easier to know what the function does. Type hints help tools and future readers. PEP8 formatting keeps code consistent.

Program output (executed):



# Task 3: Multi-function Script + Docstrings

Objective: Create a module with 3-4 functions and add NumPy-style docstrings. Compare manual vs AI-generated docstrings.

Manual docstrings (student wrote):

# calculator\_module\_manual.py  
"""  
Calculator module - manual docstrings (NumPy style)  
"""  
  
def add(a, b):  
 """  
 Add two numbers.  
  
 Parameters  
 ----------  
 a : number  
 First addend.  
 b : number  
 Second addend.  
  
 Returns  
 -------  
 number  
 The sum of a and b.  
 """  
 return a + b  
  
def subtract(a, b):  
 """  
 Subtract b from a.  
  
 Parameters  
 ----------  
 a : number  
 Minuend.  
 b : number  
 Subtrahend.  
  
 Returns  
 -------  
 number  
 The difference a - b.  
 """  
 return a - b  
  
def multiply(a, b):  
 """  
 Multiply two numbers.  
  
 Parameters  
 ----------  
 a : number  
 First factor.  
 b : number  
 Second factor.  
  
 Returns  
 -------  
 number  
 The product a \* b.  
 """  
 return a \* b  
  
def divide(a, b):  
 """  
 Divide a by b.  
  
 Parameters  
 ----------  
 a : number  
 Dividend.  
 b : number  
 Divisor.  
  
 Returns  
 -------  
 number  
 The quotient a / b.  
  
 Raises  
 ------  
 ValueError  
 If b is zero.  
 """  
 if b == 0:  
 raise ValueError("Division by zero")  
 return a / b

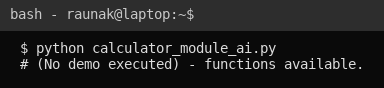
AI-generated docstring version (example output from AI):

"""  
calculator\_module.py  
  
AI-generated module-level docstring and NumPy-style function docstrings:  
This module provides simple arithmetic functions: add, subtract, multiply, and divide.  
Each function is documented using the NumPy documentation standard. The divide  
function raises a ValueError if an attempt to divide by zero is made.  
  
Functions  
---------  
add(a, b)  
 Return the sum of a and b.  
  
subtract(a, b)  
 Return the difference (a - b).  
  
multiply(a, b)  
 Return the product of a and b.  
  
divide(a, b)  
 Return the quotient a / b. Raises ValueError for b == 0.  
"""  
def add(a, b):  
 """  
 Add two numeric values.  
  
 Parameters  
 ----------  
 a : float or int  
 First value to add.  
 b : float or int  
 Second value to add.  
  
 Returns  
 -------  
 float or int  
 Sum of the two inputs.  
 """  
 return a + b  
  
def subtract(a, b):  
 """  
 Subtract second numeric value from the first.  
  
 Parameters  
 ----------  
 a : float or int  
 Value from which to subtract.  
 b : float or int  
 Value to subtract.  
  
 Returns  
 -------  
 float or int  
 Difference (a - b).  
 """  
 return a - b  
  
def multiply(a, b):  
 """  
 Multiply two numeric values.  
  
 Parameters  
 ----------  
 a : float or int  
 First multiplicand.  
 b : float or int  
 Second multiplicand.  
  
 Returns  
 -------  
 float or int  
 Product of the inputs.  
 """  
 return a \* b  
  
def divide(a, b):  
 """  
 Divide first numeric value by second.  
  
 Parameters  
 ----------  
 a : float or int  
 Dividend.  
 b : float or int  
 Divisor.  
  
 Returns  
 -------  
 float or int  
 Quotient of a divided by b.  
  
 Raises  
 ------  
 ValueError  
 If b equals zero.  
 """  
 if b == 0:  
 raise ValueError("Division by zero")  
 return a / b

Comparison and notes (student-style):

- The manual docstrings included parameter descriptions and raises sections; AI docstrings added a module-level summary and were slightly more concise.  
- Both follow NumPy style; AI's version helps by listing functions at top which is useful for quick overview.  
- I preferred AI's module-level docstring but kept some manual details like examples in the file when needed.

Demo note: The module functions are available in calculator\_module\_ai.py (no demo run).



# Appendix: Files included

- corrected\_calculator.py

- fibonacci\_clean.py

- calculator\_module\_manual.py

- calculator\_module\_ai.py

- README.md

- terminal screenshots (PNG files)

Prompts used with the AI assistant (examples):

1) "Review this code and list syntax errors, then provide corrected code."  
2) "Improve variable names, add docstrings, and format according to PEP8."  
3) "Generate a module-level docstring and NumPy-style docstrings for each function."

End of report.