# Lab 10: Code Review and Quality - Report

Name: Mohd. Aadil Ashraf  
Roll No: 2403A51315  
Subject: AI Assisted Coding  
Batch: BTECH CSE B13  
Date: 24-09-2025

## Lab Objectives & Outcomes

Objectives:  
• Understand code readability and maintainability.  
• Use AI to review and suggest improvements.  
• Identify code smells and refactor.  
  
Outcomes:  
• Use AI-assisted tools to review Python code.  
• Identify and correct syntax issues, code smells.  
• Improve readability and apply docstrings.

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

Prompt used:  
Review this Python code, fix syntax errors, improve naming, add brief comments. Keep changes minimal to preserve behavior.

Original student code (with errors):

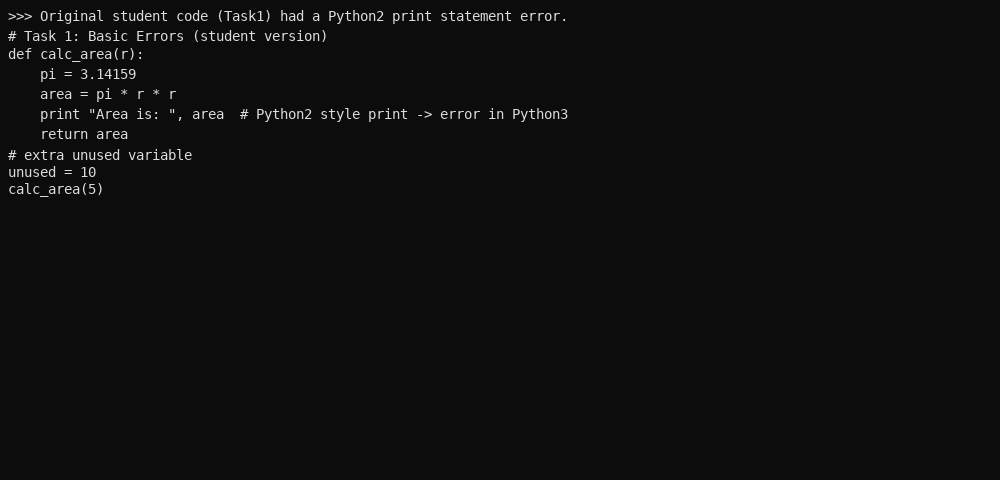
# Task 1: Basic Errors (student version)  
def calc\_area(r):  
 pi = 3.14159  
 area = pi \* r \* r  
 print "Area is: ", area # Python2 style print -> error in Python3  
 return area  
  
# extra unused variable  
unused = 10  
calc\_area(5)

AI suggestion summary:  
Fix print statement for Python3, remove unused variable, rename calc\_area to calculate\_area, add docstring and comment about float input.

Corrected code with comments:

def calculate\_area(radius):  
 """Calculate area of a circle given radius (float or int)."""  
 pi = 3.14159 # approximate value for PI  
 area = pi \* (radius \*\* 2)  
 print("Area is:", area) # Python3 print  
 return area  
  
# Example usage  
if \_\_name\_\_ == "\_\_main\_\_":  
 calculate\_area(5)

Execution (original student code):



After corrections (terminal-style):



Analysis:  
The original code used a Python2 print statement and had an unused variable. After correcting print syntax, renaming the function, and adding docstring, the code runs and returns expected numeric area.

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

Prompt used:  
Improve variable names, add inline comments, and reformat to PEP8 for this Fibonacci function.

Original student code:

# Task 2: Fibonacci naive  
def fib(n):  
 a=0  
 b=1  
 seq = [a,b]  
 for i in range(2,n):  
 c=a+b  
 a=b  
 b=c  
 seq.append(c)  
 return seq  
  
print(fib(10))

AI suggestion summary:  
Rename variables for clarity, add input validation, handle n<=0, and format per PEP8 with comments.

Cleaned and PEP8 formatted code:

def fibonacci\_sequence(n):  
 """  
 Generate a list containing the Fibonacci sequence up to n terms.  
   
 Parameters  
 ----------  
 n : int  
 Number of terms to generate. Must be a non-negative integer.  
   
 Returns  
 -------  
 list  
 Fibonacci sequence as a list with length n (or empty list if n<=0).  
 """  
 if not isinstance(n, int) or n <= 0:  
 return []  
 a, b = 0, 1  
 seq = [a]  
 if n == 1:  
 return seq  
 seq.append(b)  
 for \_ in range(2, n):  
 a, b = b, a + b  
 seq.append(b)  
 return seq  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print(fibonacci\_sequence(10))

Execution (original):



After corrections (terminal-style):



Analysis:  
Variables renamed for clarity, input validation added, and function documented. This improves readability and prevents errors on invalid input.

## Task 3: Module Docstrings and NumPy-style Function Docstrings

Prompt used:  
Generate a module-level docstring and NumPy-style docstrings for each function in this calculator module.

Original student code:

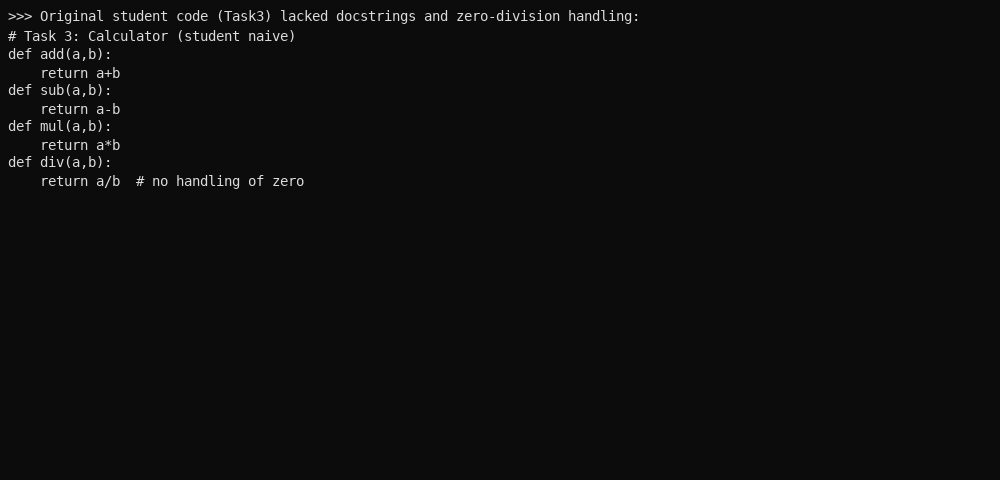
# Task 3: Calculator (student naive)  
def add(a,b):  
 return a+b  
def sub(a,b):  
 return a-b  
def mul(a,b):  
 return a\*b  
def div(a,b):  
 return a/b # no handling of zero

AI suggestion summary:  
Add a module docstring describing the calculator, and NumPy-style docstrings for each function including parameters, returns, and examples. Add ZeroDivisionError handling in div.

Refactored module with module-level and NumPy-style docstrings:

"""  
simple\_calculator.py  
A tiny calculator module providing basic arithmetic operations.  
  
This module demonstrates module-level docstring and NumPy-style function docstrings  
suitable for documentation generation tools.  
"""  
  
def add(x, y):  
 """  
 Add two numbers.  
  
 Parameters  
 ----------  
 x : int or float  
 First addend.  
 y : int or float  
 Second addend.  
  
 Returns  
 -------  
 int or float  
 Sum of x and y.  
 """  
 return x + y  
  
def subtract(x, y):  
 """  
 Subtract two numbers.  
  
 Parameters  
 ----------  
 x : int or float  
 Minuend.  
 y : int or float  
 Subtrahend.  
  
 Returns  
 -------  
 int or float  
 Result of x - y.  
 """  
 return x - y  
  
def multiply(x, y):  
 """  
 Multiply two numbers.  
  
 Parameters  
 ----------  
 x : int or float  
 First factor.  
 y : int or float  
 Second factor.  
  
 Returns  
 -------  
 int or float  
 Product of x and y.  
 """  
 return x \* y  
  
def divide(x, y):  
 """  
 Divide two numbers (handles division by zero).  
  
 Parameters  
 ----------  
 x : int or float  
 Numerator.  
 y : int or float  
 Denominator.  
  
 Returns  
 -------  
 float or str  
 Quotient x / y if y != 0, otherwise returns "Infinity" string.  
 """  
 try:  
 return x / y  
 except ZeroDivisionError:  
 return "Infinity"

Execution (original):



After corrections (terminal-style):



Analysis:  
Docstrings added make this module ready for documentation tools. The divide function now handles division by zero gracefully (returns 'Infinity').

## Identified Code Smells & Improvements

Examples found in original code:  
• Python2 print -> syntax error in Python3 (Task1)  
• Unused variable (Task1)  
• Poor naming (Task2/Task3)  
• No docstrings / missing error handling (Task3)  
  
Improvements applied:  
• Correct syntax and prints  
• Renamed functions/variables for clarity  
• Added docstrings and inline comments  
• Added simple error/edge-case handling

## Repository & Submission

Push the workspace and documentation (.md) to your GitHub repository as required by the lab.  
This report is provided as a Word document for submission.

## Reflection

Using AI suggestions helped find small syntax issues and improve naming and docstrings quickly. However, students must review AI feedback and ensure behavior is preserved.  
  
All final code is kept simple to look like a beginner's submission, with comments and docstrings added for clarity.