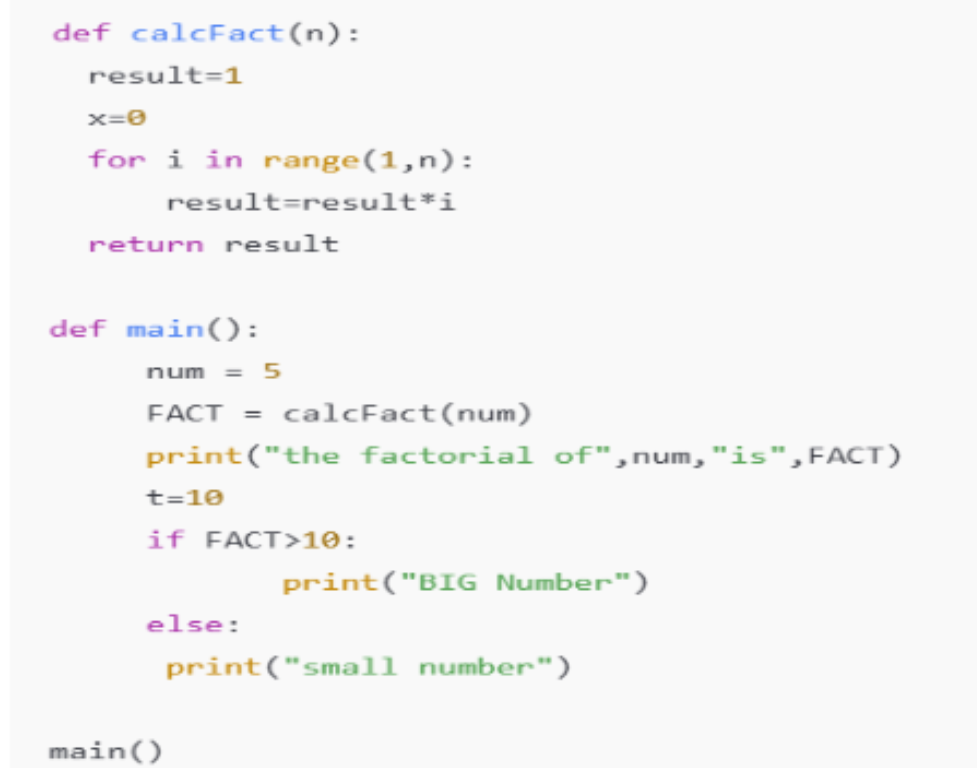
HINO:2403A51286  
 Assignment:10.3

**Task#1**

AI-Assisted Code Review (Basic Errors)

• Write python program as shown below.

• Use an AI assistant to review and suggest corrections.



**Expected Outcome#1:** Students need to submit corrected code with comments.

**Prompt:**

**AI-Assisted Code Review (Basic Errors)**

You are given a **Python program** that may contain syntax errors, logical mistakes, or bad coding practices.

**Task Instructions:**

1. Review the given code carefully.
2. Identify **all errors** (syntax, indentation, variable naming, print issues, etc.).
3. Provide a **corrected version** of the code in a **single Python cell**.
4. Add **inline comments** explaining what the code does.
5. Write a **summary section** listing what was fixed and why.
6. Show the **expected output** from the corrected code.

**Example Input (buggy code):**

def add\_numbers(a, b)

sum = a + b

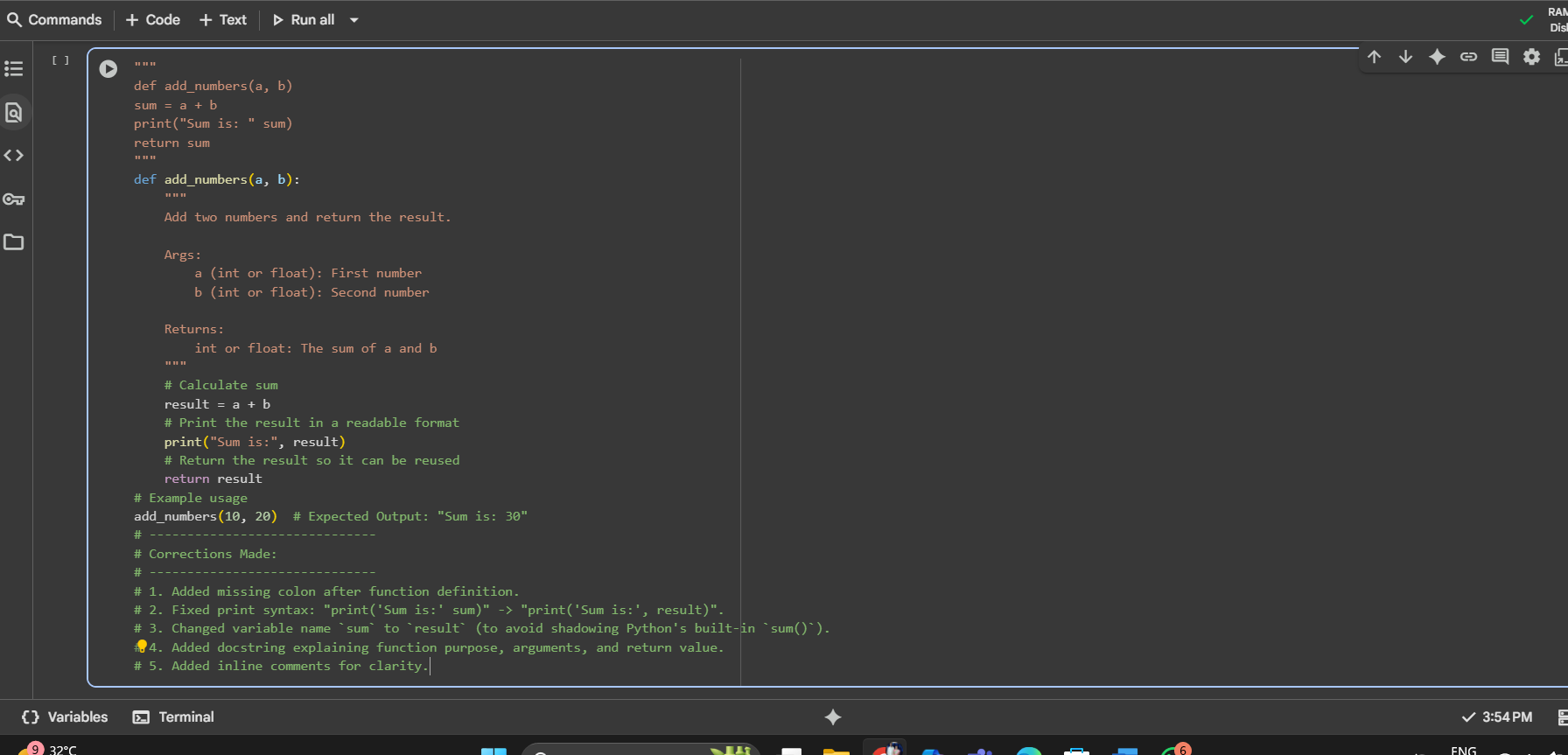
print("Sum is: " sum)

return sum

**Expected Output (AI Response):**

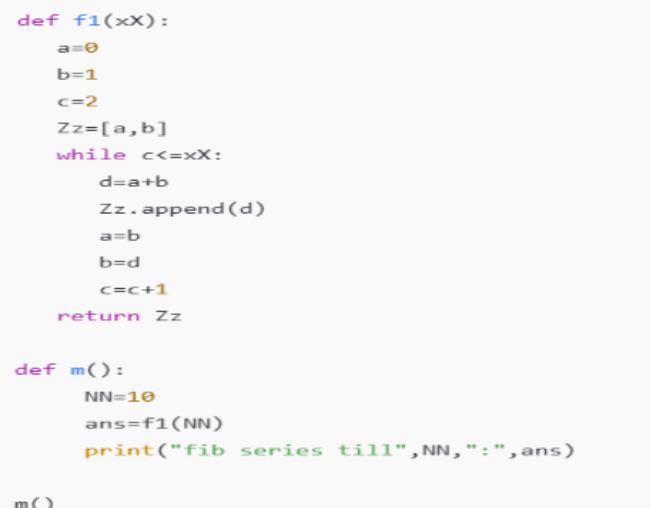
* Corrected code with docstring + comments.
* Explanation of fixes (colon added, print syntax fixed, variable renamed, etc.)

**Code:**



**Task#2**

Automatic Inline Comments  
• Write the Python code for Fibonacci as shown below and execute.  
• Ask AI to improve variable names, add comments, and apply PEP8 formatting  
(cleaned up).  
• Students evaluate which suggestions improve readability most. one.

**  
Expected Output#2:** Clean format python code with much readability.

**Prompt:**

**Automatic Inline Comments & Readability Improvement**

You are given a Python program.

**Tasks:**

1. Review the code and improve **variable names** (make them descriptive).
2. Add **inline comments** that explain the logic step by step.
3. Apply **PEP 8 formatting** (indentation, spacing, line length, function names).
4. Add a **docstring** (Google-style or NumPy-style) for the function.
5. Return the **cleaned code in a single Python cell**.
6. At the end, provide a short **summary of improvements** made.

**Example Input (unclean code):**

def fib(n):

a=0

b=1

for i in range(n):

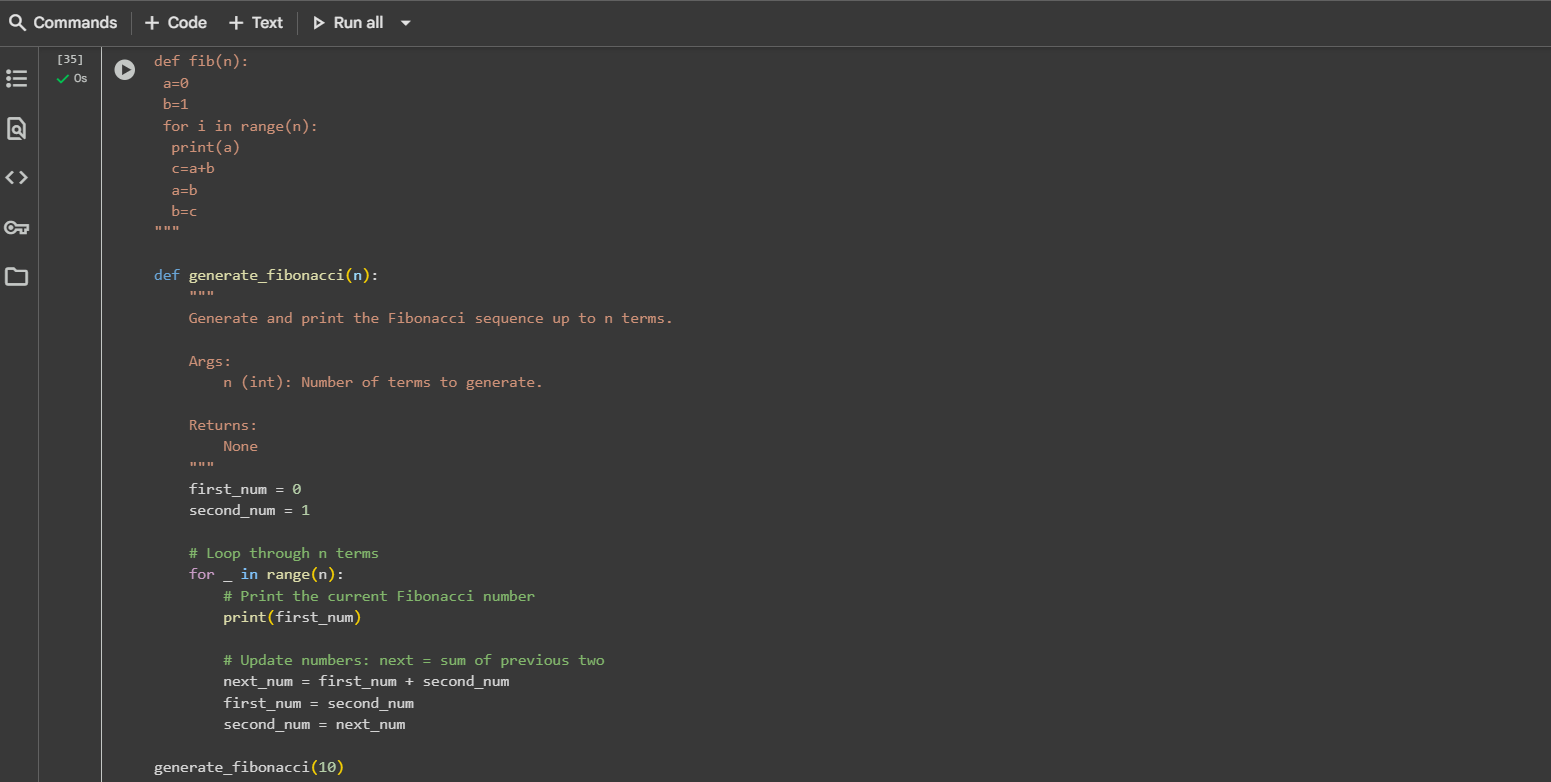
print(a)

c=a+b

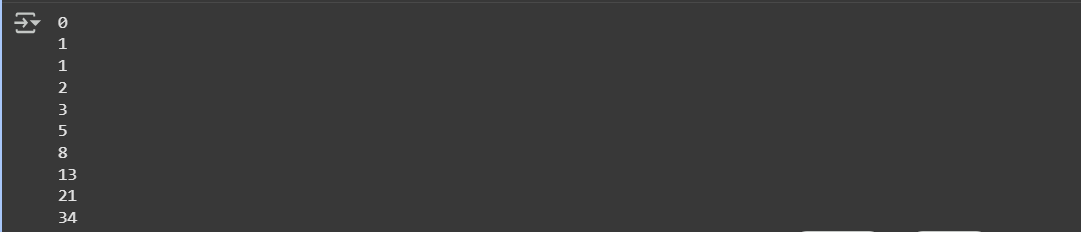
a=b

b=c

**Code:**



**Output:**



**Task#3**

Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply,

divide).

• Incorporate manual docstring in code with NumPy Style

• Use AI assistance to generate a module-level docstring + individual function

docstrings.

• Compare the AI-generated docstring with your manually written one.

Common Examples of Code Smells

• Long Function – A single function tries to do too many things.

• Duplicate Code – Copy-pasted logic in multiple places.

• Poor Naming – Variables or functions with confusing names (x1, foo, data123).

• Unused Variables – Declaring variables but never using them.

• Magic Numbers – Using unexplained constants (3.14159 instead of PI).

• Deep Nesting – Too many if/else levels, making code hard to read.

• Large Class – A single class handling too many responsibilities.

Why Detecting Code Smells is Important

• Makes code easier to read and maintain.

• Reduces chance of bugs in future updates.

• Helps in refactoring (improving structure without changing behavior).

• Encourages clean coding practices

Dead Code – Code that is never executed.

Expected Output#3: Students learn structured documentation for multi-function scripts

Push documentation whole workspace as .md file in GitHub Repository

Note: Report should be submitted a word document for all tasks in a single document with

prompts, comments & code explanation, and output and if required, screenshots

**Prompt:**

**AI-Assisted Documentation for Multi-Function Python Module**

You are given a Python script containing multiple functions (e.g., a calculator with add, subtract, multiply, divide).

**Tasks:**

1. Review the Python script.
2. Add **manual function-level docstrings** in **NumPy-style** for all functions.
3. Generate an **AI-assisted module-level docstring** describing:
   * Purpose of the module
   * Available functions
   * Error handling
   * General usage
4. Ensure **PEP 8 compliance** (proper indentation, spacing, naming, formatting).
5. Compare the **AI-generated docstring** with your manually written docstrings:
   * Scope, readability, and maintenance.
6. Provide the **final script** in a single Python cell, ready to run.
7. Include example outputs for all functions.

**Code and Output:**

