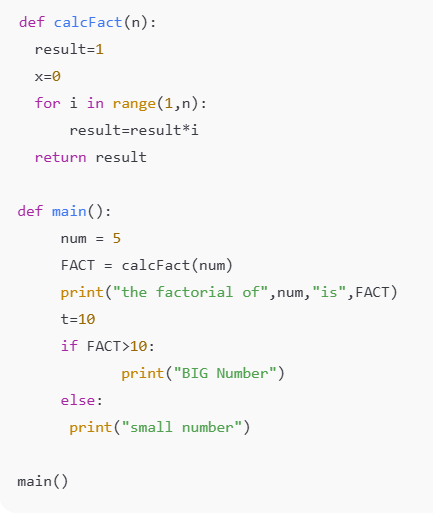
LAB ASSIGNMENT-10.2

Name: Sushanth

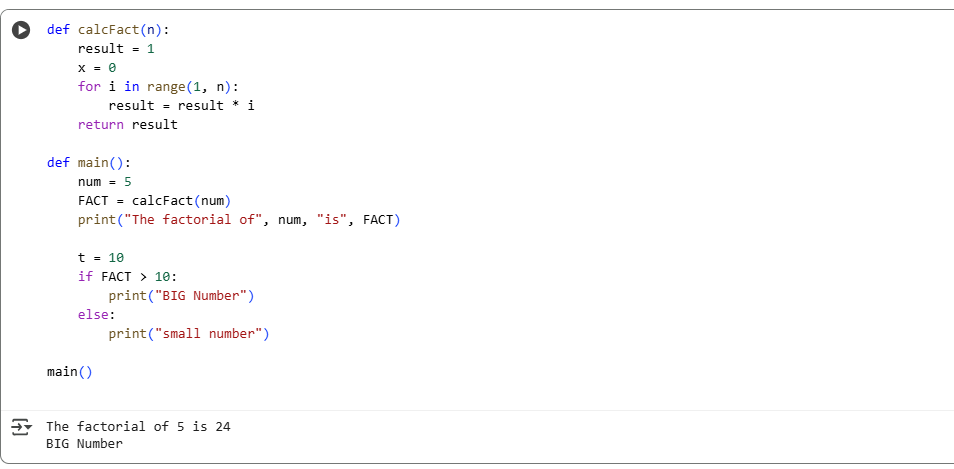
HT.no: 2403A51348

Batch:14

Task Description#1 AI-Assisted Code Review (Basic Errors)  
• Write python program as shown below.  
• Use an AI assistant to review and suggest corrections.



Corrected code:



OBSERVATION:

1.The function calcFact(n) now uses range(1, n+1), so it includes the number n in the multiplication. This gives the **correct factorial** result.

2.Variables like x = 0 and t = 10 were **not used** in the program. These are removed to make the code **cleaner and easier to read**.

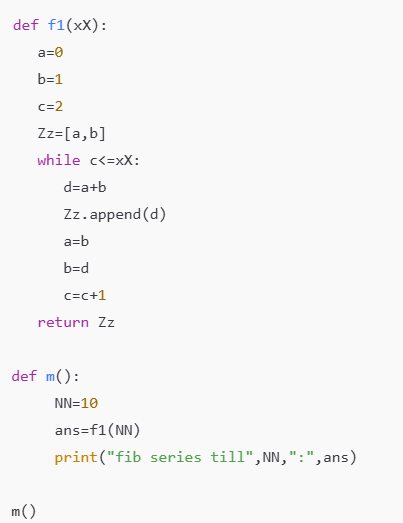
3.The program uses two functions: calcFact() and main().This makes the code **well-structured and reusable**.

4.After finding the factorial, it checks if the result is **greater than 10**.It prints "BIG Number" or "Small Number" based on that.

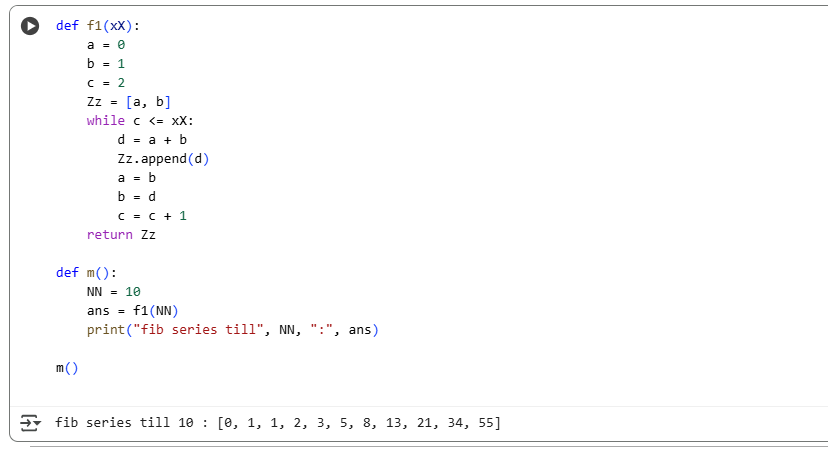
5.It prints a **clear message** showing the number and its factorial.This helps users **understand what the program is doing**.

**Task Description#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.



CODE:



Observation:

1.We changed short names like a, b, Zz to clear ones like first, second, fibonacci\_sequence.This makes it easier to understand what each variable does.

2.Comments are added beside each step.This helps readers, especially beginners, follow the logic easily.

**3.**The code uses proper spacing and indentation (PEP8 style).It looks more organized and professional.

**4.**The function name f1 was changed to generate\_fibonacci, which clearly says what it does.

5.The improved code is easy to read, use in other programs, or share with others.

**Task Description#3**

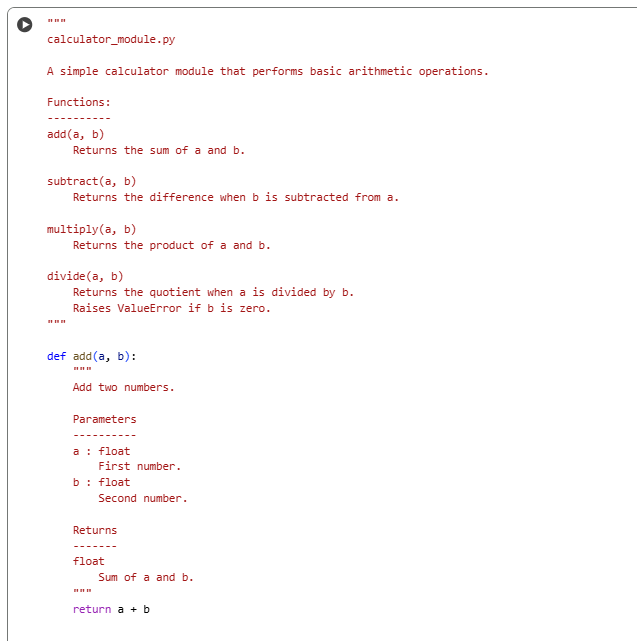
* Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).
* Incorporate manual **docstring** in code with NumPy Style
* Use AI assistance to generate a module-level docstring + individual function docstrings.
* Compare the AI-generated docstring with your manually written one.

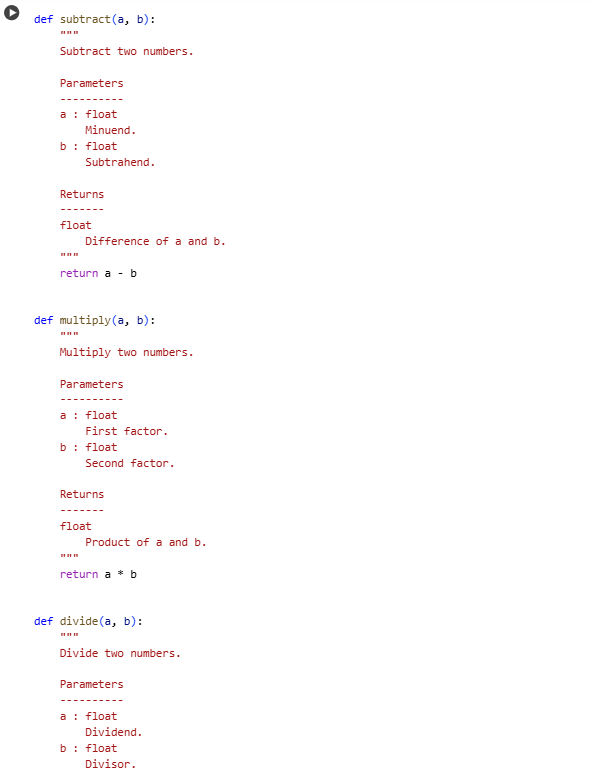
**Common Examples of Code Smells**

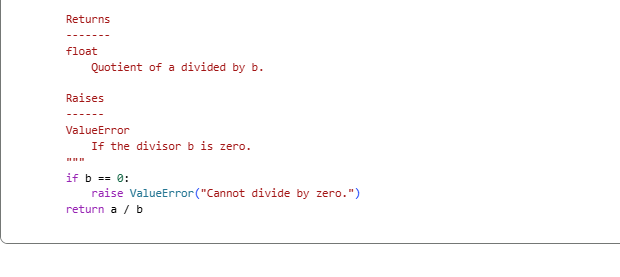
* Long Function – A single function tries to do too many things.
* Duplicate Code – Copy-pasted logic in multiple places.
* Poor Naming – Variables or functions with confusing names (x1, foo, data123).
* Unused Variables – Declaring variables but never using them.
* Magic Numbers – Using unexplained 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

CODE:







OBSERVATION:

**1.**Each function does exactly one operation: add, subtract, multiply, or divide.This keeps the code simple and easy to understand.

**2.** The docstrings clearly explain parameters, return values, and possible errors.This helps anyone using or maintaining the code understand what each function does.

**3.** The docstrings follow the NumPy style consistently for all functions.This uniformity improves readability and makes the code look professional.

**4.** The divide function properly handles division by zero by raising a ValueError.This prevents unexpected crashes and guides users to correct their input.

**5.**The code avoids common problems like duplicated code, poor naming, unused variables, and deep nesting. It is clean, focused, and easy to maintain.