NAME : G.Raju

LAB ASSIGNMENT-10.2

HALLTICKETNUMBER : 2403A52018

BATCH NUMBER : 02

### Task Description#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.

def calcFact(n):

result=1 x=0

for i in range(1,n): result=result\*i return result

def main(): num = 5

FACT = calcFact(num)

print("the factorial of",num,"is",FACT) t=10

if FACT>10:

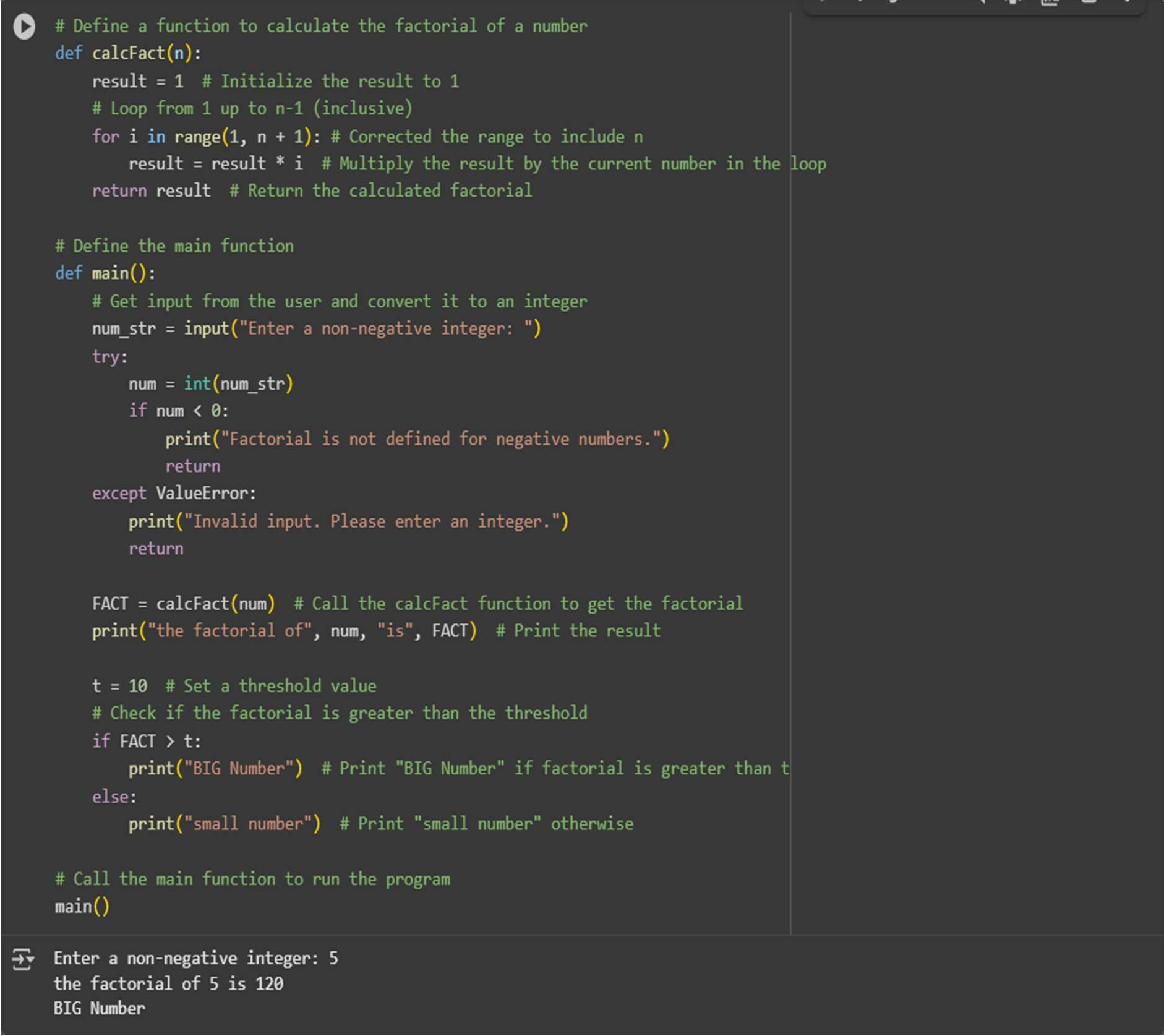
print("BIG Number") else:

print("small number") main()

# PROMPT:

Write python program as shown above and submit corrected code with comments.

CODE:



# Issues

#### Factorial calculation is incorrect

* + range(1, n) should be range(1, n + 1) to include n.

#### Unused variable x = 0

* + Not used anywhere – remove it.

#### Unused variable t = 10

* + Declared but never used – remove it.

#### Variable name FACT

* + Uppercase suggests a constant – use lowercase (fact) for readability.

#### Minor style issues

* + Print message lacks formatting/capitalization – improve spacing and clarity.

## OBSERVATION

The factorial function uses range(1, n), which incorrectly omits the last number; it should be range(1, n + 1).

There are unused variables (x = 0 and t = 10) that should be removed to clean the code.

The variable FACT uses all caps, which is against Python naming conventions for regular variables.

Minor style improvements are needed in print formatting and naming for better readability.



return Zz



a=b



d



# Task Description#

# def f1(xX):

# a=0

# b=1

# c=2

# Zz=[a,b]

# while c<=xX:

# d=a+b

# Zz.append(d)

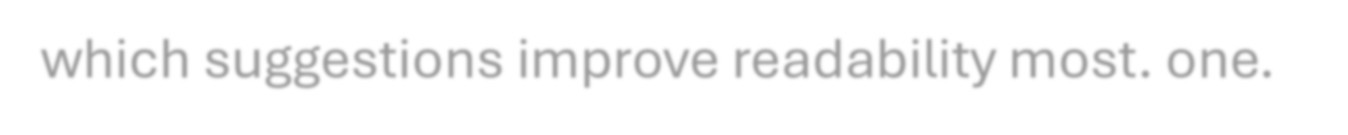
# a=b

# b=d

# c=c+1

# return Zz

# def m():



# NN=10

# ans=f1(NN)

# print("fib series till",NN,":",ans)

# m()

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

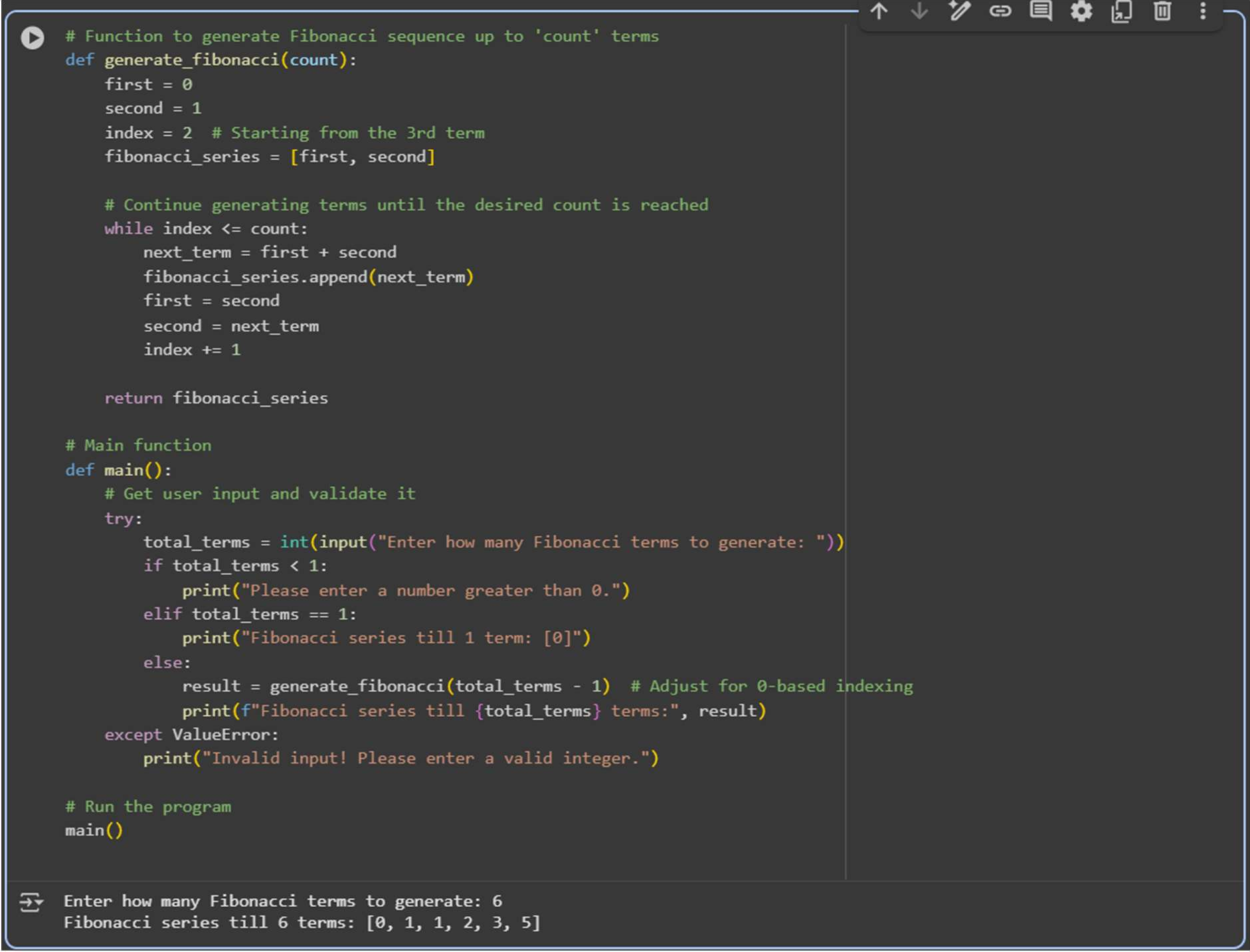
Expected output : Clean format python code with much readability



Clean formapython code with much readability

2:

# CODE:



Issues

1. **Poor variable names** – a, b, c, xX, etc., are unclear.
2. **No user input** – Uses a fixed number instead of asking the user.
3. **No input validation** – Doesn't handle invalid or negative values.
4. **Undescriptive function names** – f1() and m() should be meaningful.
5. **No comments or formatting** – Code lacks clarity and PEP8 style.

## OBSERVATION:

The code correctly generates the Fibonacci series but uses unclear variable names like a, b, xX, and Zz, which reduce readability.

Functions like f1() and m() are not descriptive and should be renamed. There is no user input or validation, making the program inflexible and

error-prone.

Additionally, the code lacks comments and does not follow PEP8 formatting guidelines.



TASK 3

# 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

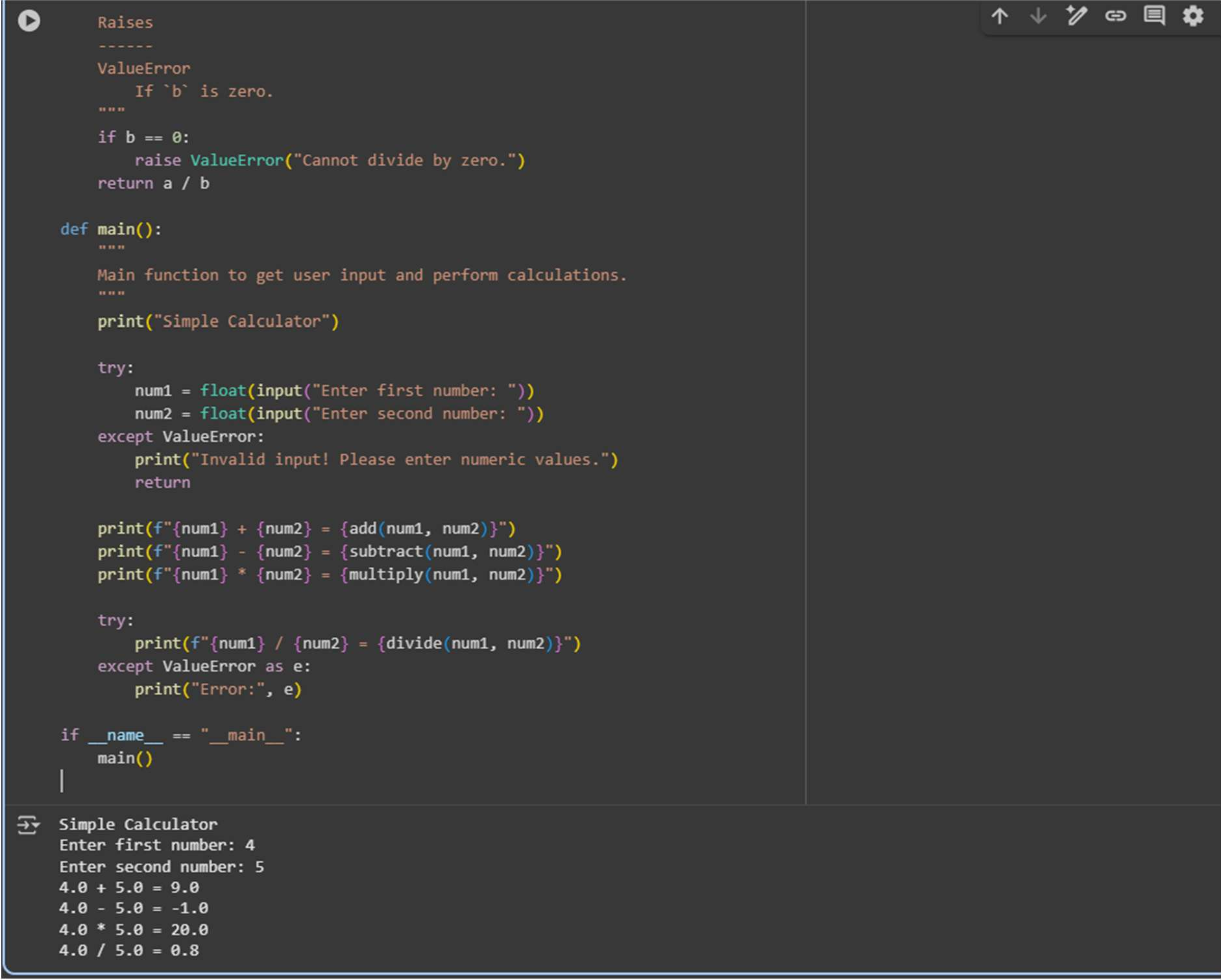
* Long Function – A single function tries to do too many things.
* Duplicate Code – Copy-pasted logic in multiple places.
* Poor Naming – Variables or functions with confusing names (x1, foo, data123).
* Unused Variables – Declaring variables but never using them.
* Magic Numbers – Using unexplained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

### Why Detecting Code Smells is Important

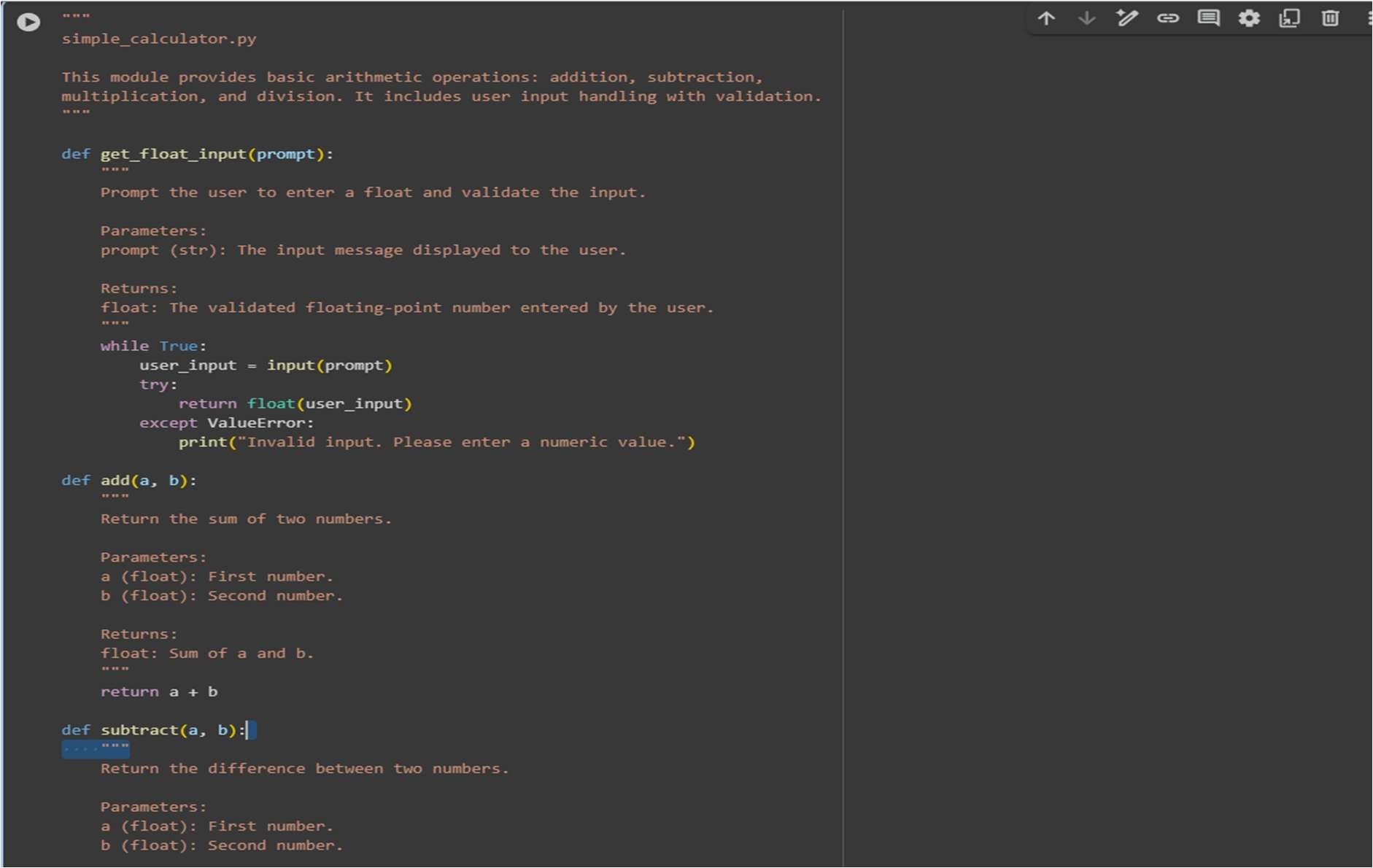
* Makes code easier to read and maintain.
* Reduces chance of bugs in future updates.
* Helps in refactoring (improving structure without changing behavior).
* Encourages clean coding practices Dead Code – Code that is never executed.

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

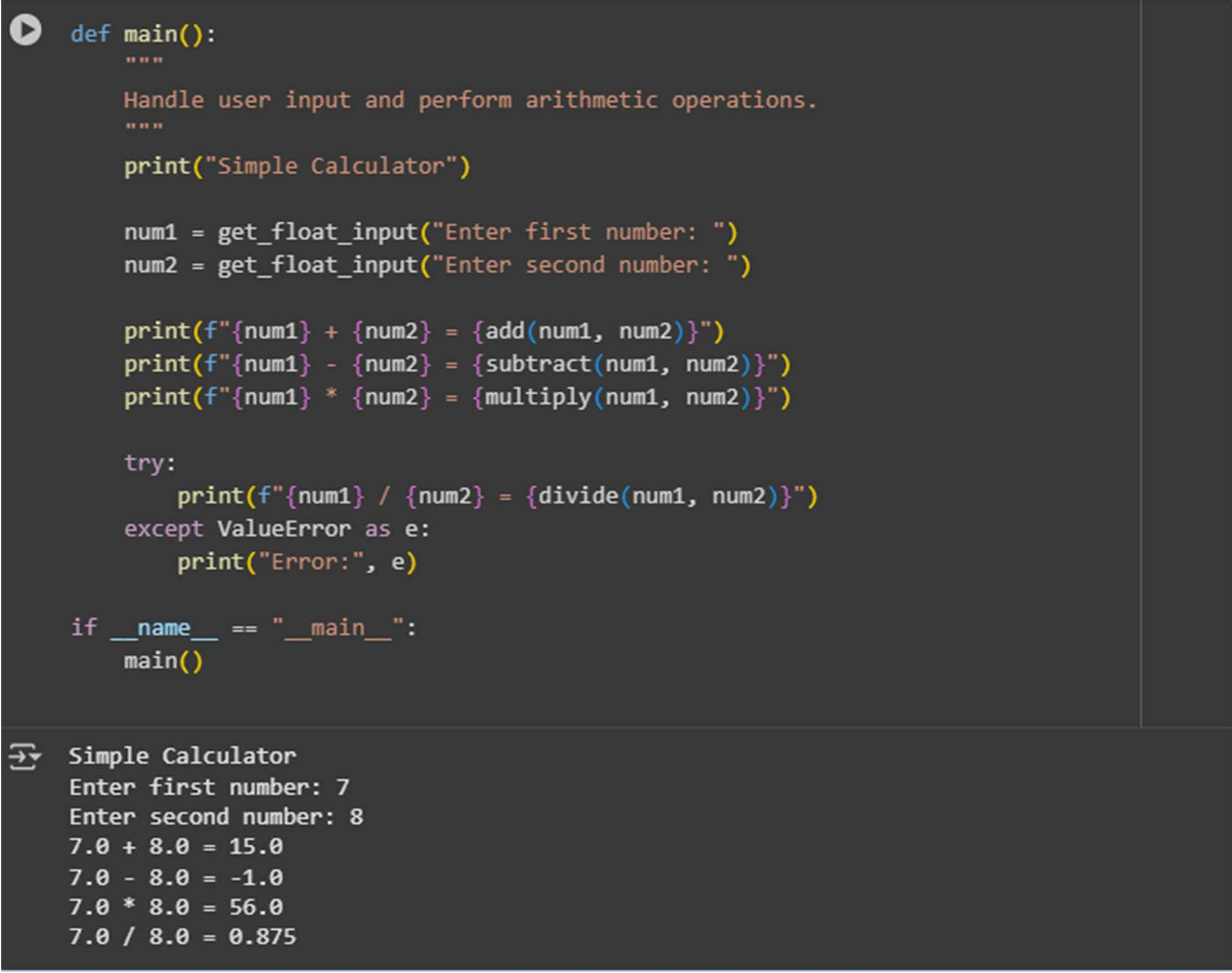
MANUALLY WRITTEN CODE:



## AI GENERATED CODE:







COMPARISON:

* **Manual docstrings** are detailed, structured, and include sections like Parameters, Returns, and Raises. They are great for clear, professional documentation.
* **AI-generated docstrings** are shorter, simpler, and easier to read quickly but less formal and detailed.
* Manual style is best for large projects, while AI style works well for small scripts or quick docs.

## OBSERVATION:

The manual docstrings provide detailed and well-structured documentation, clearly explaining parameters, return values, and exceptions, which is ideal for maintainability and professional projects. In contrast, the AI-generated docstrings are concise and easy to read but lack formal structure and depth. While AI docstrings improve speed and simplicity, manual docstrings offer better clarity for complex codebases.

Choosing between them depends on the project size and documentation needs.