**AI ASSISTED CODING LAB**

**ASSIGNMENT 10.3**

**ENROLLMENT NO :**2503A51L05

**BATCH NO:** 19

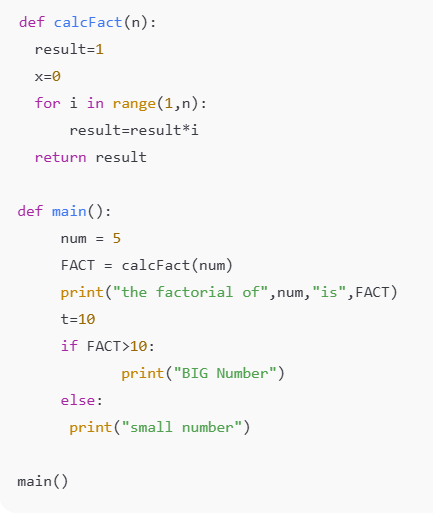
**NAME:** NARADALA SATYA SRI CHARAN

**TASK 1**

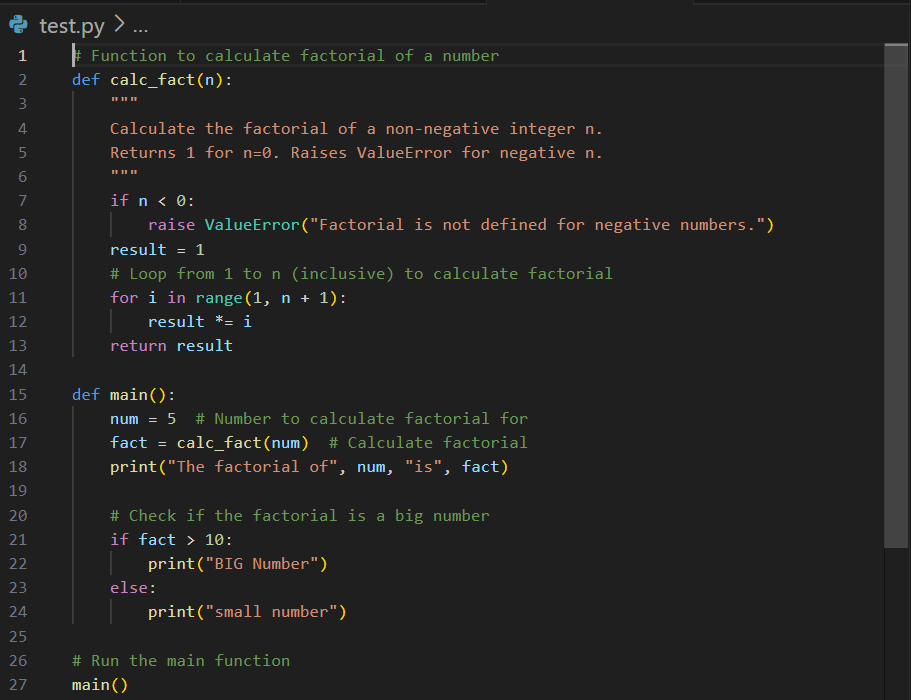
**TASK1:** (Basic Errors)

• Write python program as shown below.

• Use an AI assistant to review and suggest corrections**.**

****

**PROMPT : I give correct code**

****

**CODE generated :**

def calc\_fact(n):

"""

Calculate the factorial of a non-negative integer n.

Returns 1 for n=0. Raises ValueError for negative n.

"""

if n < 0:

raise ValueError("Factorial is not defined for negative numbers.")

result = 1

for i in range(1, n + 1): # range should include n

result \*= i

return result

def main():

num = 5

fact = calc\_fact(num) # Calculate factorial

print("The factorial of", num, "is", fact)

# Check if the factorial is a big number

if fact > 10:

print("BIG Number")

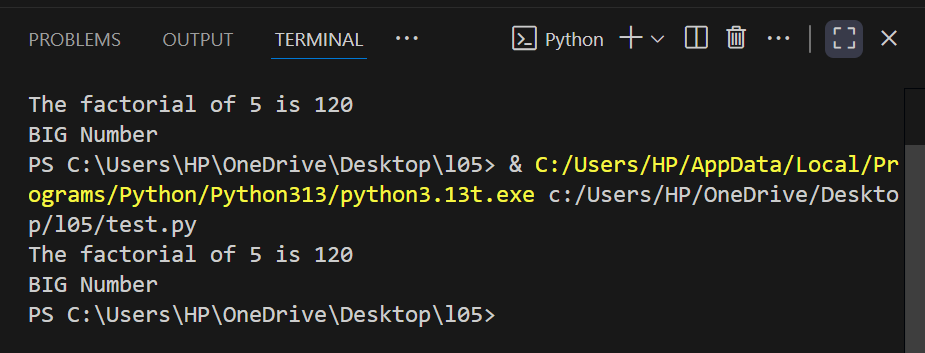
else:

print("small number")

# Run the main function

main()

**Expected output:**

****

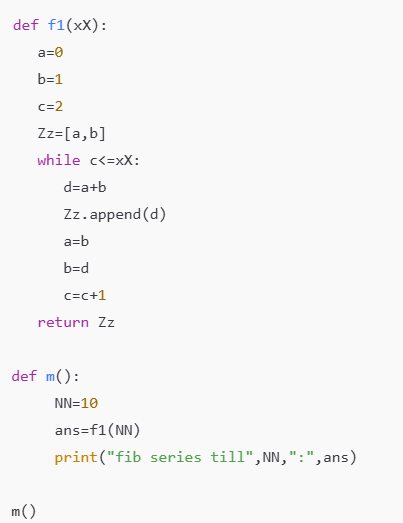
**OBSERVATION:**

The function calc\_fact(n) correctly calculates the factorial of a non-negative integer n.

It raises a ValueError for negative inputs, which is good error handling.

**Task 2**

**Task 2:** Automatic Inline Comments

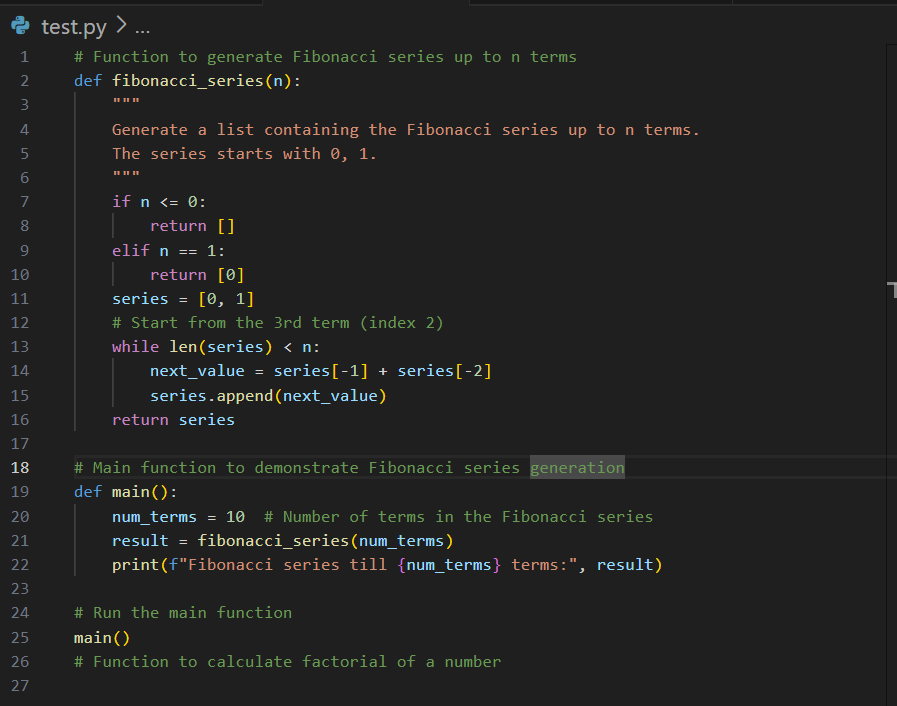


• Write the Python code for Fibonacci as shown below and execute.

• Ask AI to improve variable names, add comments, and apply PEP8 formatting (cleaned up).

• Students evaluate which suggestions improve readability most.

**PROMPT : this code correct**

****

**CODE generated :**

# Function to generate Fibonacci series up to n terms

def fibonacci\_series(n):

"""

Generate a list containing the Fibonacci series up to n terms.

The series starts with 0, 1.

"""

if n <= 0:

return []

elif n == 1:

return [0]

series = [0, 1]

# Generate the next terms until the list has n elements

while len(series) < n:

next\_value = series[-1] + series[-2]

series.append(next\_value)

return series

# Main function to demonstrate Fibonacci series generation

def main():

num\_terms = 10 # Number of terms in the Fibonacci series

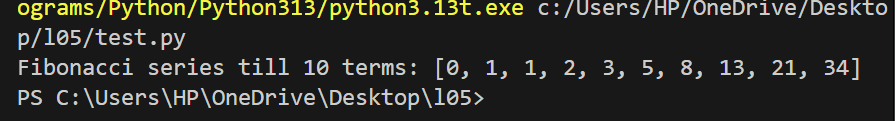
result = fibonacci\_series(num\_terms)

print(f"Fibonacci series till {num\_terms} terms:", result)

# Run the main function

main()

**OUTPUT:**

****

**OBSERVATION:** I observated

The function fibonacci\_series(n) correctly generates a list of the first n Fibonacci numbers, starting from 0 and 1.

**TASK3**

**TASK3:**

**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**

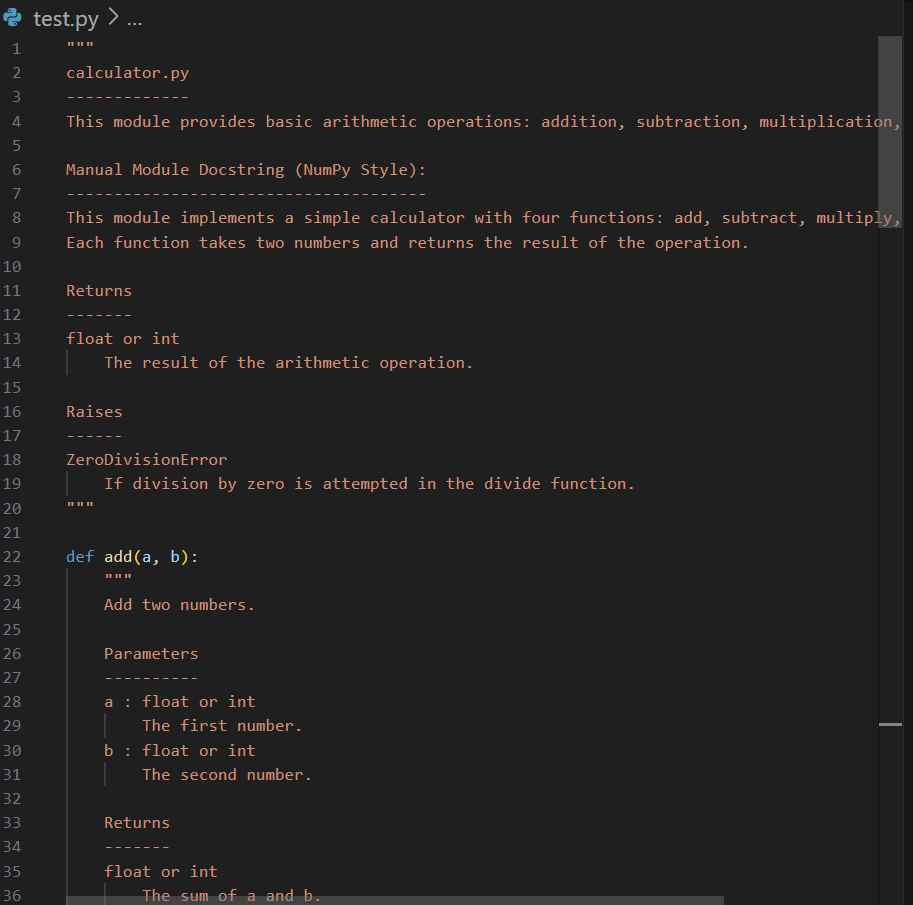
* Long Function – A single function tries to do too many things.
* Duplicate Code – Copy-pasted logic in multiple places.
* Poor Naming – Variables or functions with confusing names (x1, foo, data123).
* Unused Variables – Declaring variables but never using them.
* Magic Numbers – Using unexplained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

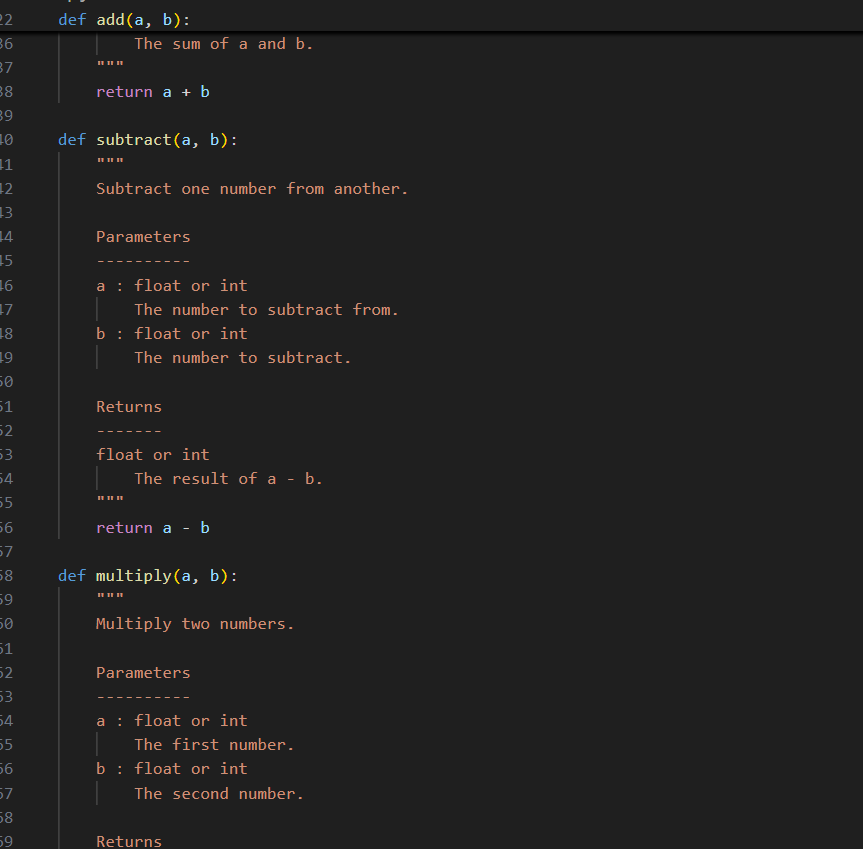
**Why Detecting Code Smells is Important**

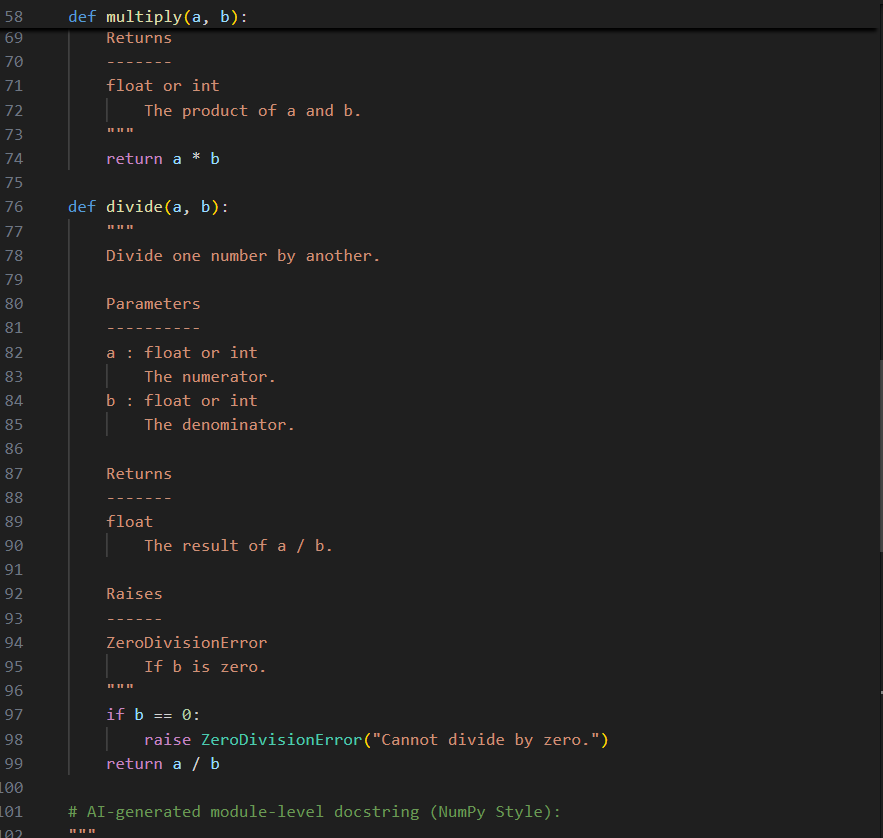
* Makes code easier to read and maintain.
* Reduces chance of bugs in future updates.
* Helps in refactoring (improving structure without changing behavior).
* Encourages clean coding practices

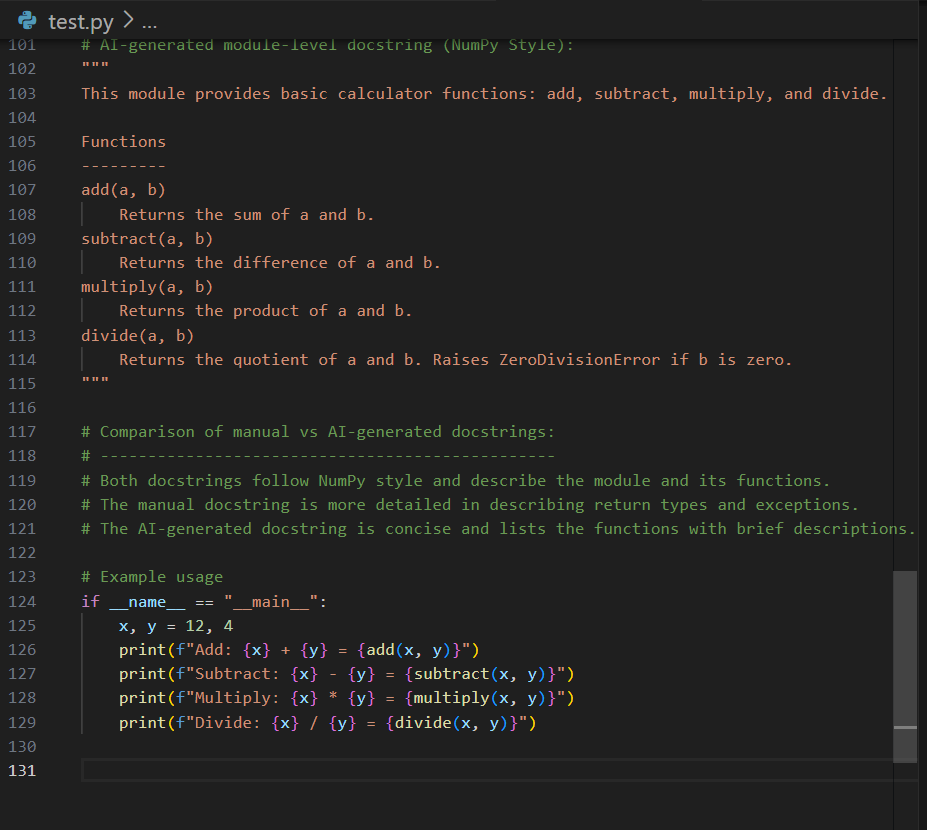
**Dead Code – Code that is never executed.**

**PROMPT:**

****

****

****

****

**CODE GENGRATE :**

"""

calculator.py

-------------

This module provides basic arithmetic operations: addition, subtraction, multiplication, and division.

Manual Module Docstring (NumPy Style):

--------------------------------------

This module implements a simple calculator with four functions: add, subtract, multiply, and divide.

Each function takes two numbers and returns the result of the operation.

Returns

-------

float or int

The result of the arithmetic operation.

Raises

------

ZeroDivisionError

If division by zero is attempted in the divide function.

""”

def add(a, b):

"""

Add two numbers.

Parameters

----------

a : float or int

The first number.

b : float or int

The second number.

Returns

-------

float or int

The sum of a and b.

"""

return a + b

def subtract(a, b):

"""

Subtract one number from another.

Parameters

----------

a : float or int

The number to subtract from.

b : float or int

The number to subtract.

Returns

-------

float or int

The result of a - b.

"""

return a - b

def multiply(a, b):

"""

Multiply two numbers.

Parameters

----------

a : float or int

The first number.

b : float or int

The second number.

Returns

-------

float or int

The product of a and b.

"""

return a \* b

def divide(a, b):

"""

Divide one number by another.

Parameters

----------

a : float or int

The numerator.

b : float or int

The denominator.

Returns

-------

float

The result of a / b.

Raises

------

ZeroDivisionError

If b is zero.

"""

if b == 0:

raise ZeroDivisionError("Cannot divide by zero.")

return a / b

# AI-generated module-level docstring (NumPy Style):

"""

This module provides basic calculator functions: add, subtract, multiply, and divide.

Functions

---------

add(a, b)

Returns the sum of a and b.

subtract(a, b)

Returns the difference of a and b.

multiply(a, b)

Returns the product of a and b.

divide(a, b)

Returns the quotient of a and b. Raises ZeroDivisionError if b is zero.

"""

# Comparison of manual vs AI-generated docstrings:

# ------------------------------------------------

# Both docstrings follow NumPy style and describe the module and its functions.

# The manual docstring is more detailed in describing return types and exceptions.

# The AI-generated docstring is concise and lists the functions with brief descriptions.

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

x, y = 12, 4

print(f"Add: {x} + {y} = {add(x, y)}")

print(f"Subtract: {x} - {y} = {subtract(x, y)}")

print(f"Multiply: {x} \* {y} = {multiply(x, y)}")

print(f"Divide: {x} / {y} = {divide(x, y)}")

**OBSERVATION** : he script provides four basic arithmetic functions: add, subtract, multiply, and divide.

Each function is simple, focused, and does exactly one thing.

**OUTPUT** : NEVER EXECUTE