ASSIGNMENT:10.2

HTNO:2403A51284

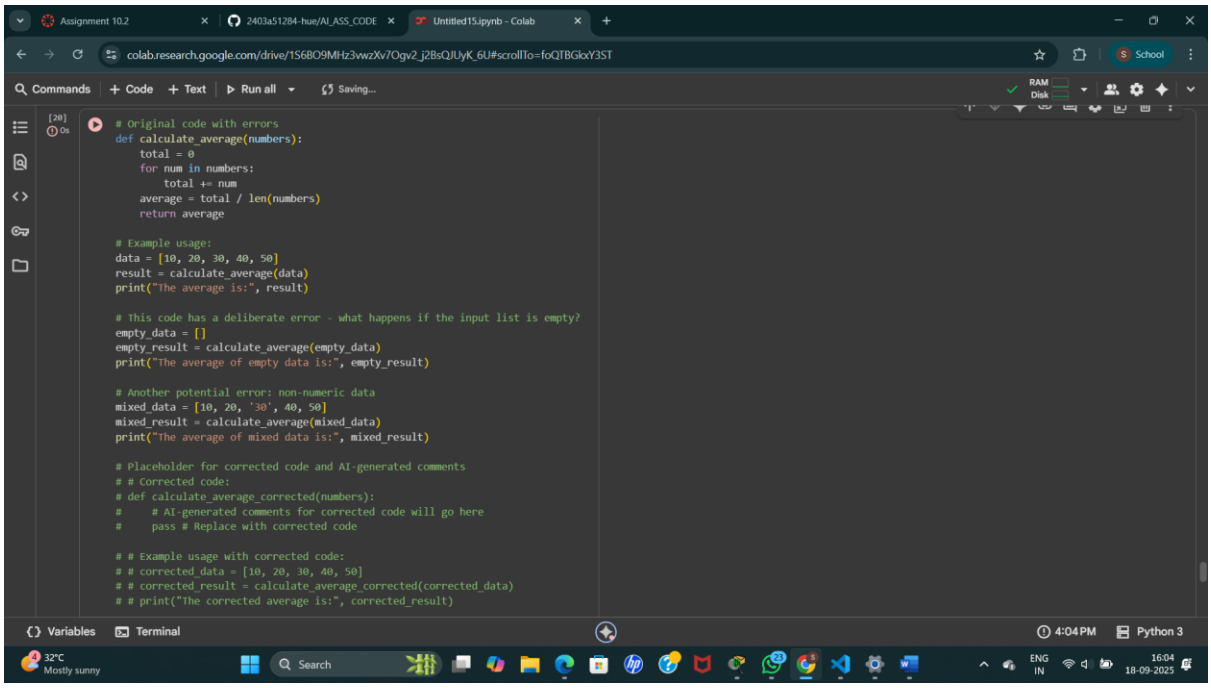
Task Description#1 AI-Assisted Code Review (Basic Errors)
• Write python program as shown below.
• Use an AI assistant to review and suggest corrections.
Expected Outcome#1: Students need to submit corrected code with comments

PROMPT:

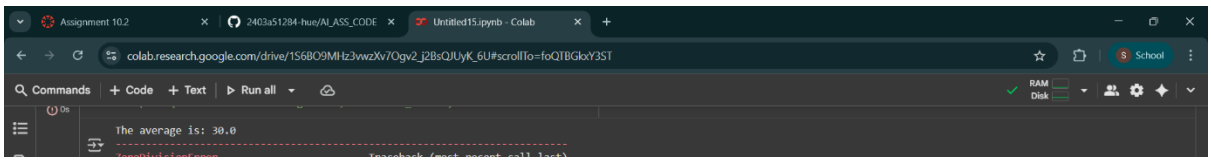
You've reached your monthly chat messages quota. Upgrade to Copilot Pro (30-day free trial) or wait for your allowance to renew.

Upgrade to GitHub

Code:



OUTPUT:



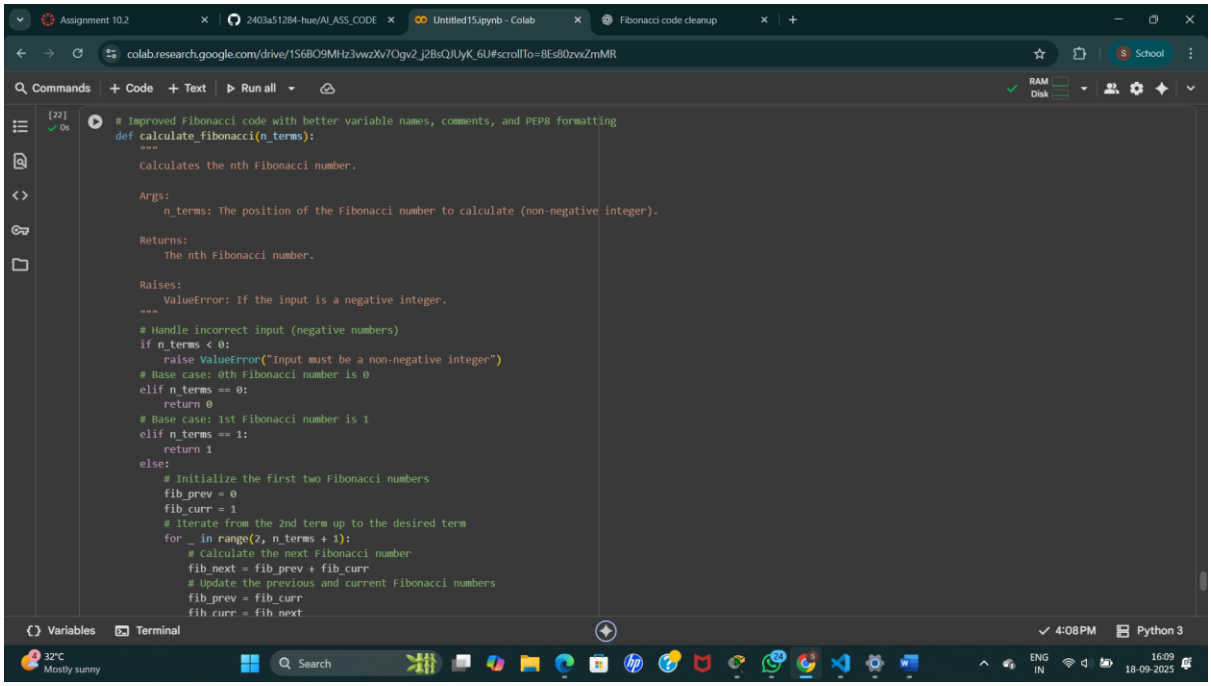
Task Description#2 Automatic Inline Comments
• Write the Python code for Fibonacci as shown below and execute.
• Ask AI to improve variable names, add comments, and apply PEP8 formatting (cleaned up).
• Students evaluate which suggestions improve readability most. one.
Expected Output#2: Clean format python code with much readability.

PROMPT:

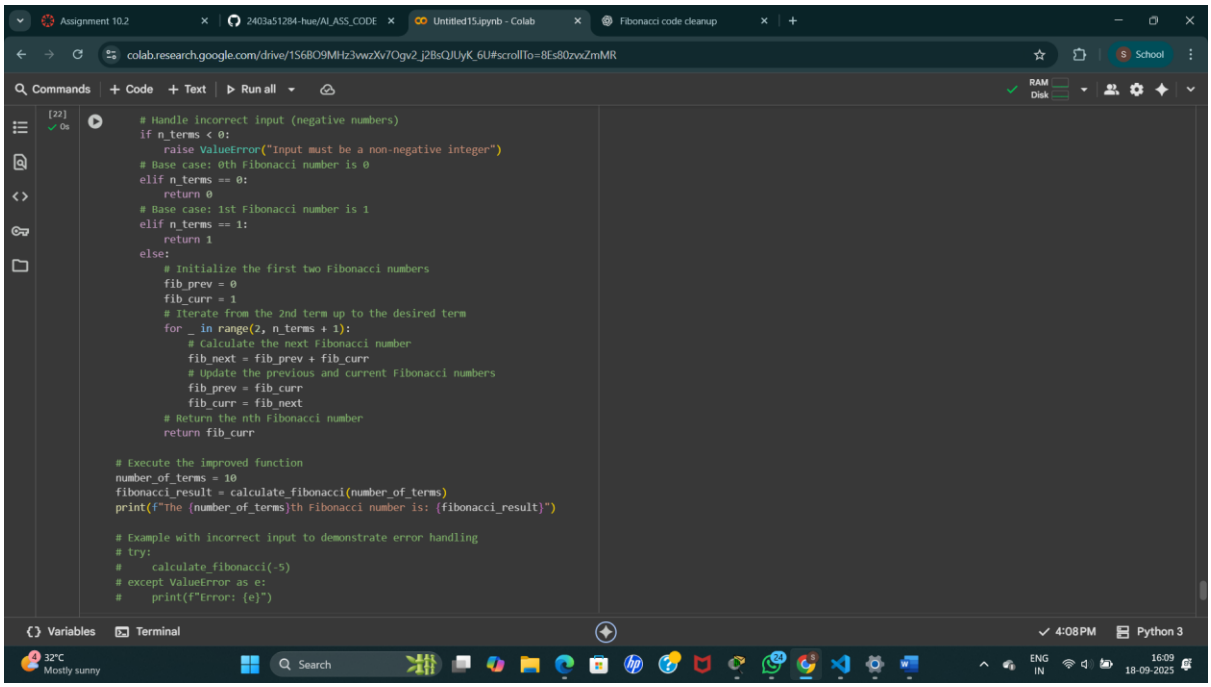
have a Python program for generating the Fibonacci sequence. Please:

1. Improve the variable names to be more descriptive.
2. Add inline comments explaining each step.
3. Apply proper PEP8 formatting (indentation, spacing, line length).
4. Return a cleaned-up version of the code that is easier to read and understand.

CODE:

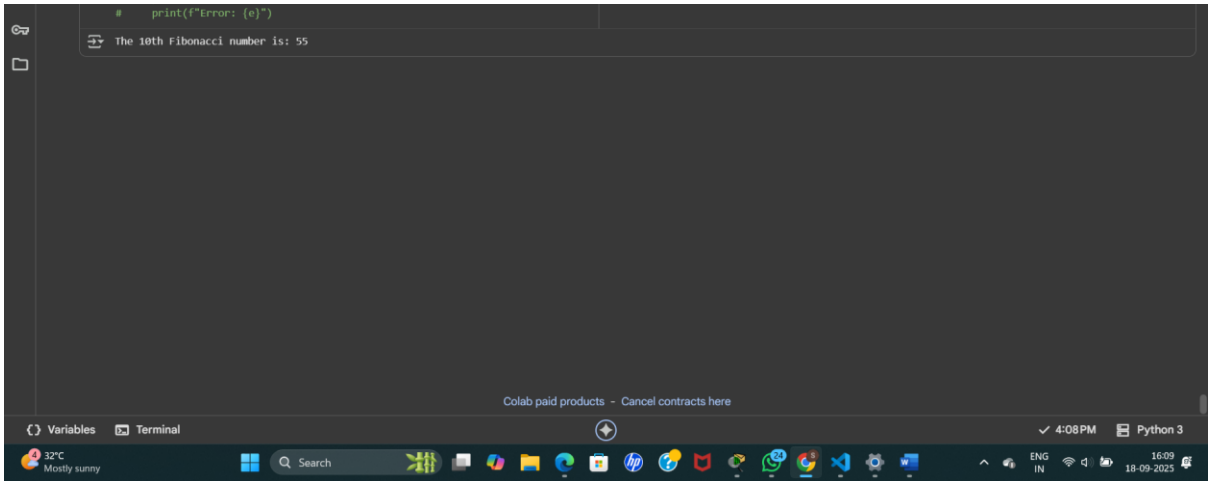


```python
# Improved Fibonacci code with better variable names, comments, and PEP8 formatting
def calculate_fibonacci(n_terms):
    """
    Calculates the nth Fibonacci number.

    Args:
        n_terms: The position of the Fibonacci number to calculate (non-negative integer).

    Returns:
        The nth Fibonacci number.

    Raises:
        ValueError: If the input is a negative integer.
    """
    # Handle incorrect input (negative numbers)
    if n_terms < 0:
        raise ValueError("Input must be a non-negative integer")
    # Base case: 0th Fibonacci number is 0
    elif n_terms == 0:
        return 0
    # Base case: 1st Fibonacci number is 1
    elif n_terms == 1:
        return 1
    else:
        # Initialize the first two Fibonacci numbers
        fib_prev = 0
        fib_curr = 1
        # Iterate from the 2nd term up to the desired term
        for _ in range(2, n_terms + 1):
            # Calculate the next Fibonacci number
            fib_next = fib_prev + fib_curr
            # Update the previous and current Fibonacci numbers
            fib_prev = fib_curr
            fib_curr = fib_next
```



```python
    # Handle incorrect input (negative numbers)
    if n_terms < 0:
        raise ValueError("Input must be a non-negative integer")
    # Base case: 0th Fibonacci number is 0
    elif n_terms == 0:
        return 0
    # Base case: 1st Fibonacci number is 1
    elif n_terms == 1:
        return 1
    else:
        # Initialize the first two Fibonacci numbers
        fib_prev = 0
        fib_curr = 1
        # Iterate from the 2nd term up to the desired term
        for _ in range(2, n_terms + 1):
            # Calculate the next Fibonacci number
            fib_next = fib_prev + fib_curr
            # Update the previous and current Fibonacci numbers
            fib_prev = fib_curr
            fib_curr = fib_next
        # Return the nth Fibonacci number
        return fib_curr

# Execute the improved function
number_of_terms = 10
fibonacci_result = calculate_fibonacci(number_of_terms)
print(f"The {number_of_terms}th Fibonacci number is: {fibonacci_result}")

# Example with incorrect input to demonstrate error handling
# try:
#     calculate_fibonacci(-5)
# except ValueError as e:
#     print(f"Error: {e}")
```

OUTPUT:



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

PROMPT:

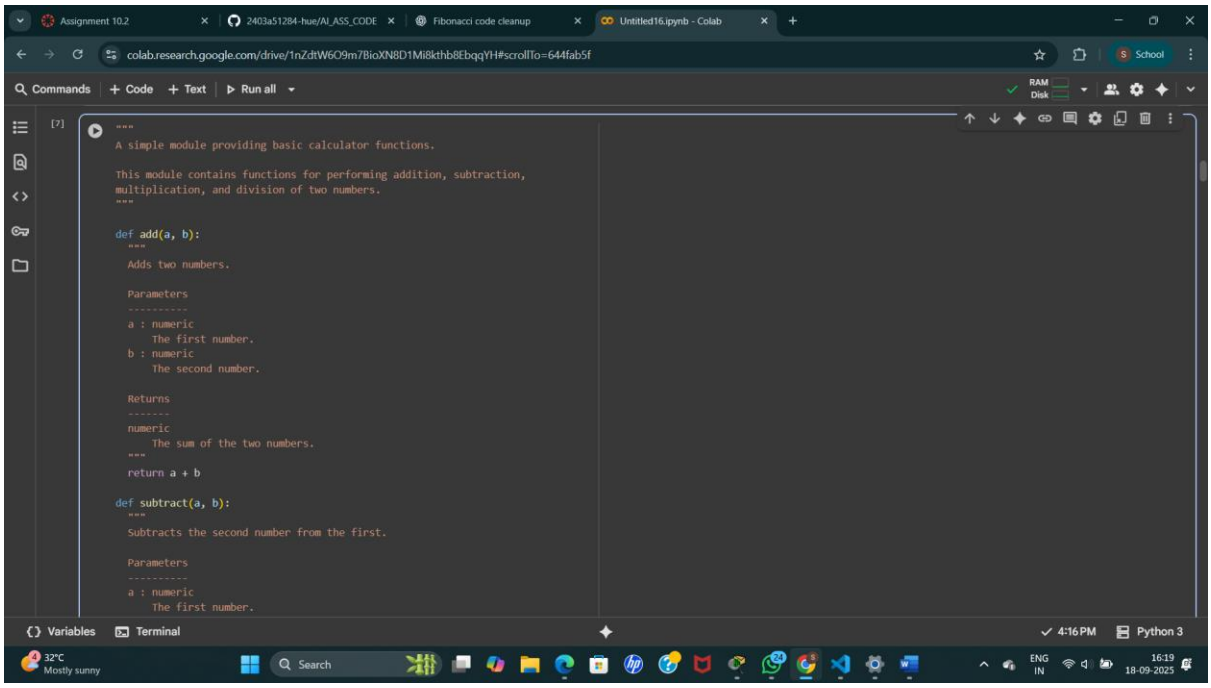
1 Generate a **module-level docstring** in **NumPy style**.

2 Add **function-level docstrings** for each function in NumPy style.

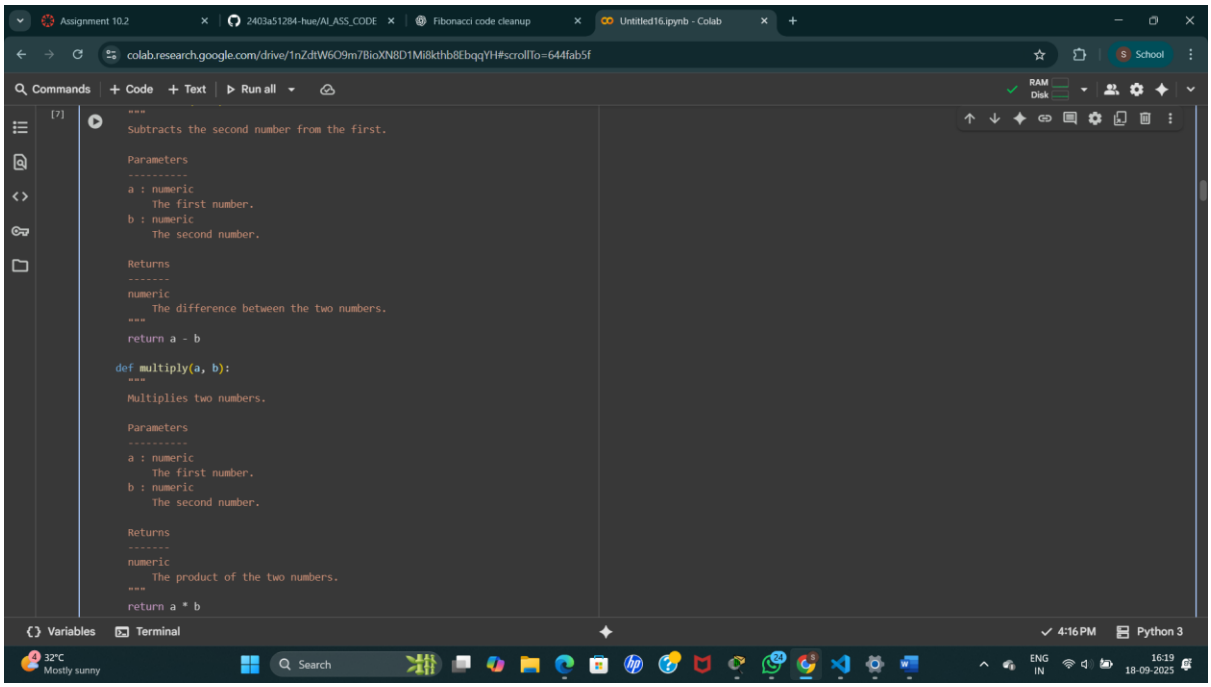
3 Ensure the script follows clean coding practices (avoid code smells like long functions, duplicate code, poor naming, unused variables, magic numbers, deep nesting, large classes, or dead code).

4 Compare the AI-generated docstrings with my manually written docstrings (to see which improves readability and clarity).

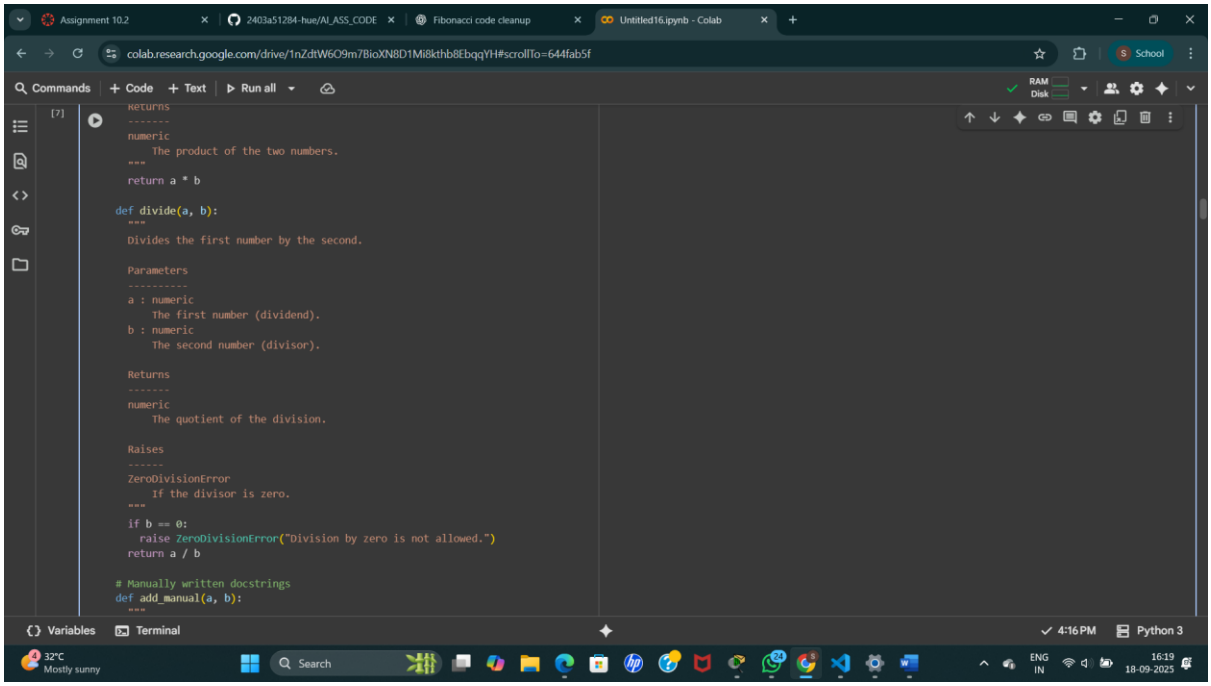
CODE:



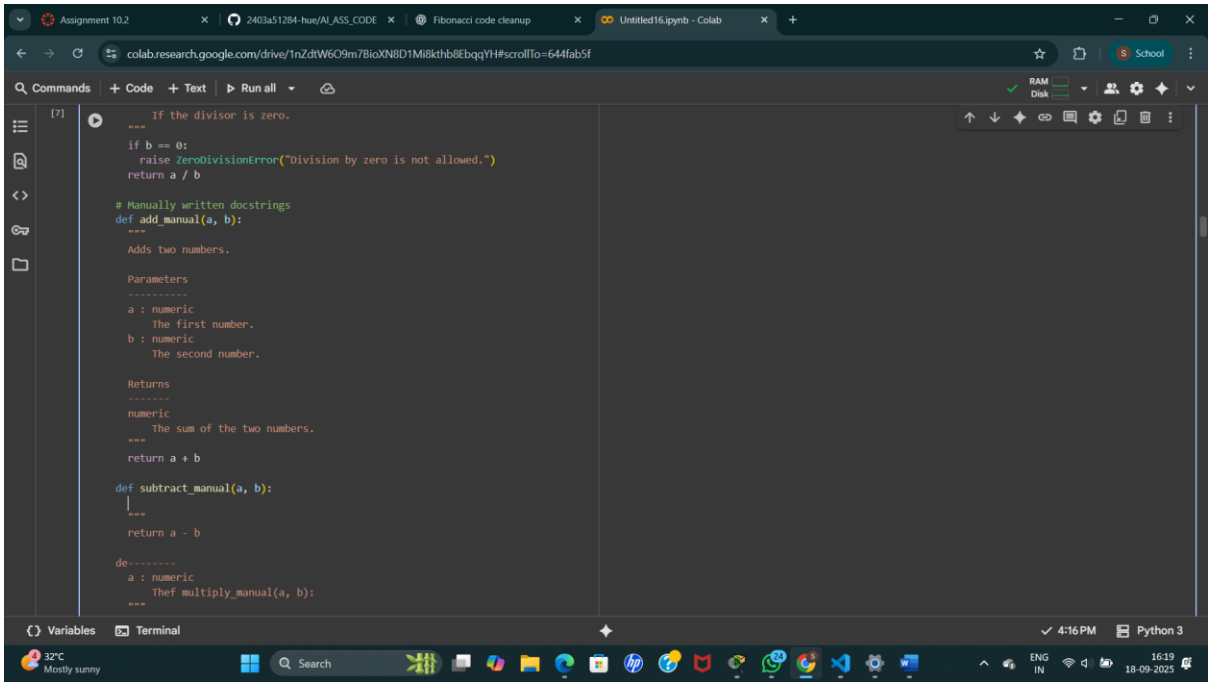
```
"""
A simple module providing basic calculator functions.

This module contains functions for performing addition, subtraction,
multiplication, and division of two numbers.
"""

def add(a, b):
  """
  Adds two numbers.

  Parameters
  ----------
  a : numeric
      The first number.
  b : numeric
      The second number.

  Returns
  -------
  numeric
      The sum of the two numbers.
  """
  return a + b

def subtract(a, b):
  """
  Subtracts the second number from the first.

  Parameters
  ----------
  a : numeric
      The first number.
```



```
  """
  Subtracts the second number from the first.

  Parameters
  ----------
  a : numeric
      The first number.
  b : numeric
      The second number.

  Returns
  -------
  numeric
      The difference between the two numbers.
  """
  return a - b

def multiply(a, b):
  """
  Multiplies two numbers.

  Parameters
  ----------
  a : numeric
      The first number.
  b : numeric
      The second number.

  Returns
  -------
  numeric
      The product of the two numbers.
  """
  return a * b
```



```
    Returns
    -------
    numeric
        The product of the two numbers.
    """
    return a * b

def divide(a, b):
    """
    Divides the first number by the second.

    Parameters
    ----------
    a : numeric
        The first number (dividend).
    b : numeric
        The second number (divisor).

    Returns
    -------
    numeric
        The quotient of the division.

    Raises
    ------
    ZeroDivisionError
        If the divisor is zero.
    """
    if b == 0:
        raise ZeroDivisionError("Division by zero is not allowed.")
    return a / b

# Manually written docstrings
def add_manual(a, b):
    """
```



```
        If the divisor is zero.
    """
    if b == 0:
        raise ZeroDivisionError("Division by zero is not allowed.")
    return a / b

# Manually written docstrings
def add_manual(a, b):
    """
    Adds two numbers.

    Parameters
    ----------
    a : numeric
        The first number.
    b : numeric
        The second number.

    Returns
    -------
    numeric
        The sum of the two numbers.
    """
    return a + b

def subtract_manual(a, b):
    |
    """
    return a - b

de--------
    a : numeric
        Thef multiply_manual(a, b):
    """
```

OUTPUT:

```
--- Comparing docstrings for add ---
Manual Docstring:
Adds two numbers.

Parameters
----------
a : numeric
    The first number.
b : numeric
    The second number.

Returns
-------
numeric
    The sum of the two numbers.

AI-Generated Docstring (Plausible):
Adds two numbers.

Parameters
----------
a : int or float
    The first number.
b : int or float
    The second number.

Returns
-------
int or float
    The sum of a and b.
--------------------------------
--- Comparing docstrings for subtract ---
Manual Docstring:
Subtracts the second number from the first.

Parameters
```

```
Parameters
----------
a : numeric
    The first number.
b : numeric
    The second number.

Returns
-------
numeric
    The difference between the two numbers.

AI-Generated Docstring (Plausible):
Subtracts the second number from the first.

Parameters
----------
a : int or float
    The number to subtract from.
b : int or float
    The number to subtract.

Returns
-------
int or float
    The difference between a and b.
------------------------------------
--- Comparing docstrings for multiply ---
Manual Docstring:
Multiplies two numbers.

Parameters
----------
a : numeric
    The first number.
b : numeric
```

Assignment 10.2 | 2403a51284-hue/AI_ASS_CODE | Fibonacci code cleanup | Untitled16.ipynb - Colab

colab.research.google.com/drive/1nZdtW6O9m7BioXN8D1Mi8kthb8EbqqYH#scrollTo=644fab5f

Commands | + Code | + Text | Run all

```
Manual Docstring:
Divides the first number by the second.

Parameters
----------
a : numeric
    The first number (dividend).
b : numeric
    The second number (divisor).

Returns
-------
numeric
    The quotient of the division.

Raises
------
ZeroDivisionError
    If the divisor is zero.

AI-Generated Docstring (Plausible):
Divides the first number by the second.

Parameters
----------
a : int or float
    The dividend.
b : int or float
    The divisor.

Returns
-------
int or float
    The result of the division.
```

Add manual docstrings

Variables | Terminal | 4:16 PM | Python 3

32°C Mostly sunny | Search | ENG IN | 16:21 18-09-2025