AI ASSISTED CODING

Program :B.tech(CSE)

Name :ANANTHA MANIDEEP

En No. :2403A52078

Batch No. :02

LAB ASSIGNMENT-10.2

Task Description -1:

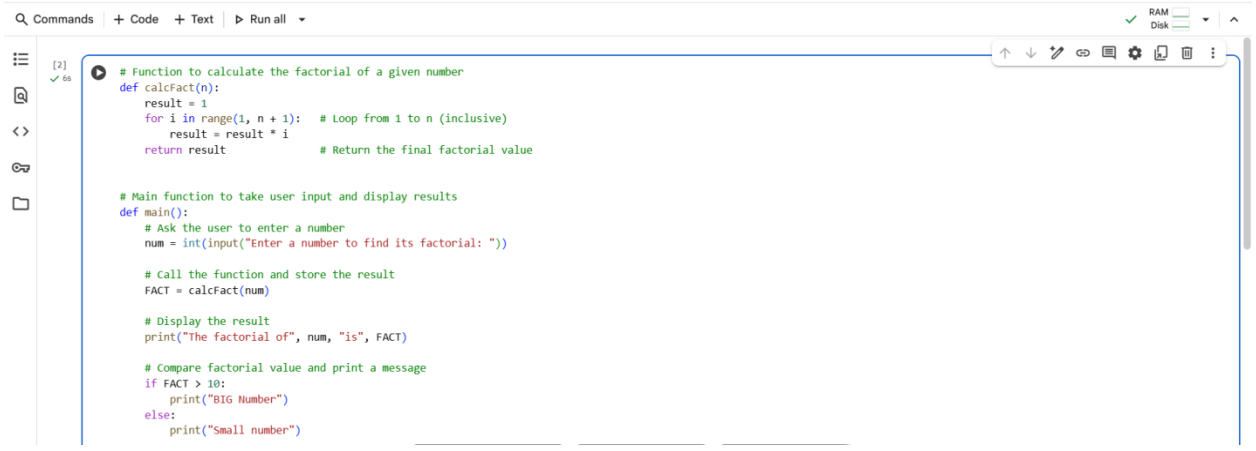
Write python program as shown below.
Use an AI assistant to review and suggest corrections..

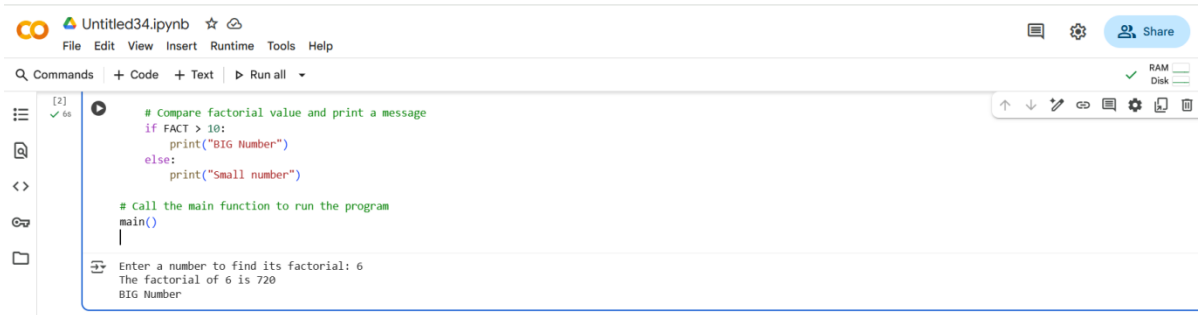
PROMPT:

- Review and correct the following Python program for calculating the factorial of a number. Identify and fix all logical or syntax errors. Add comments explaining each correction.

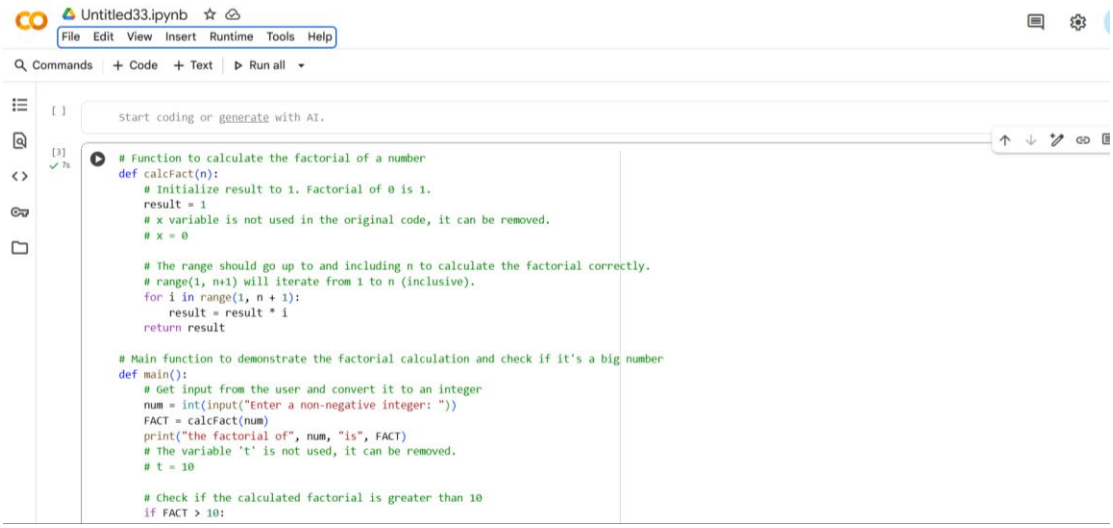
```
def calcFact(n):
  result=1
  x=0
  for i in range(1,n):
       result=result*i
  return result

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()
```

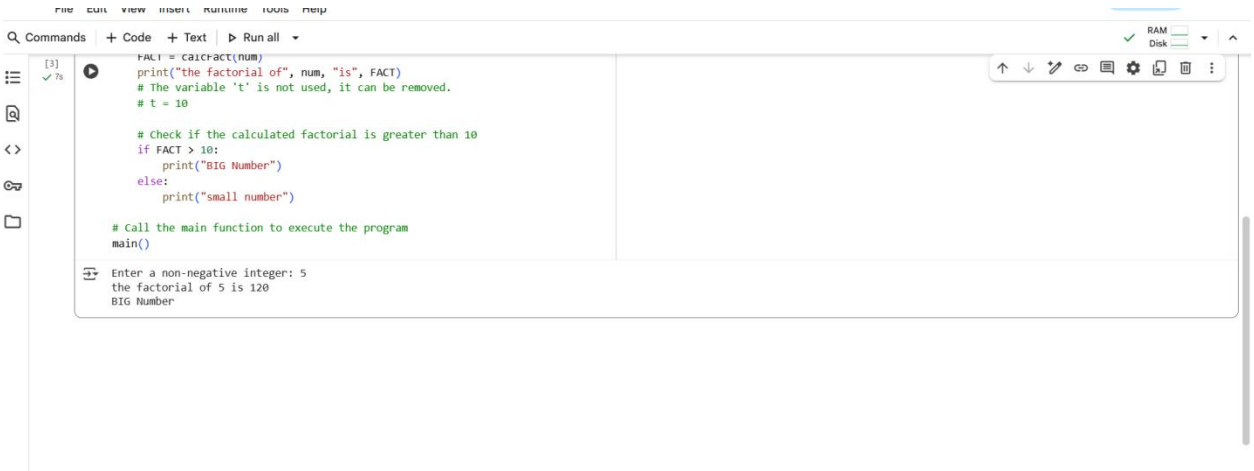
Code with output corrected by manual:





Code & output corrected with AI:





Comparing both codes:

By comparing both codes Ai written well but which code I written it is more understandable compare to AI.

EXPLANATION:

- `generate_fibonacci_series(num_terms)`: This function takes the desired number of terms as input. It initializes a list with the first two Fibonacci numbers (0 and 1). It then iteratively calculates the next number in the series by adding the previous two, appending each new number to the list until the desired number of terms is reached.
- `main()`: This function prompts the user to enter the number of terms, calls `generate_fibonacci_series()` to get the series, and then prints the resulting Fibonacci series.

TASK DESCRIPTION-2:

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.

```
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()
```

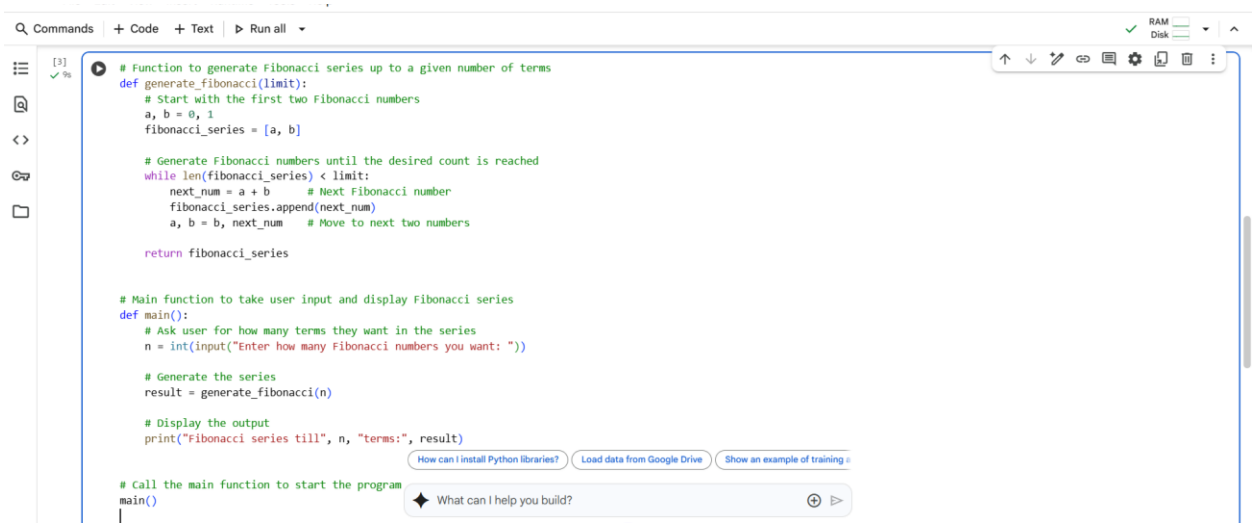
PROMPT:

Review and improve the following Python Fibonacci program.

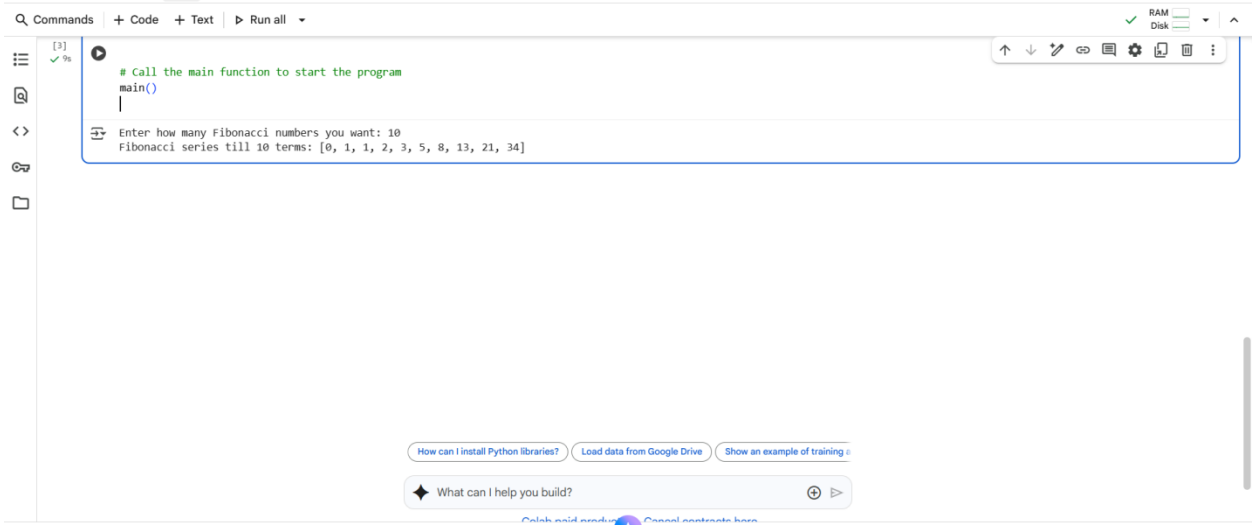
- Use meaningful variable names.
- Add clear comments explaining each step.
- Format the code properly according to PEP8 style guidelines.
- Make the code clean and easy to read.

```
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()
```

Code &output corrected by manual:



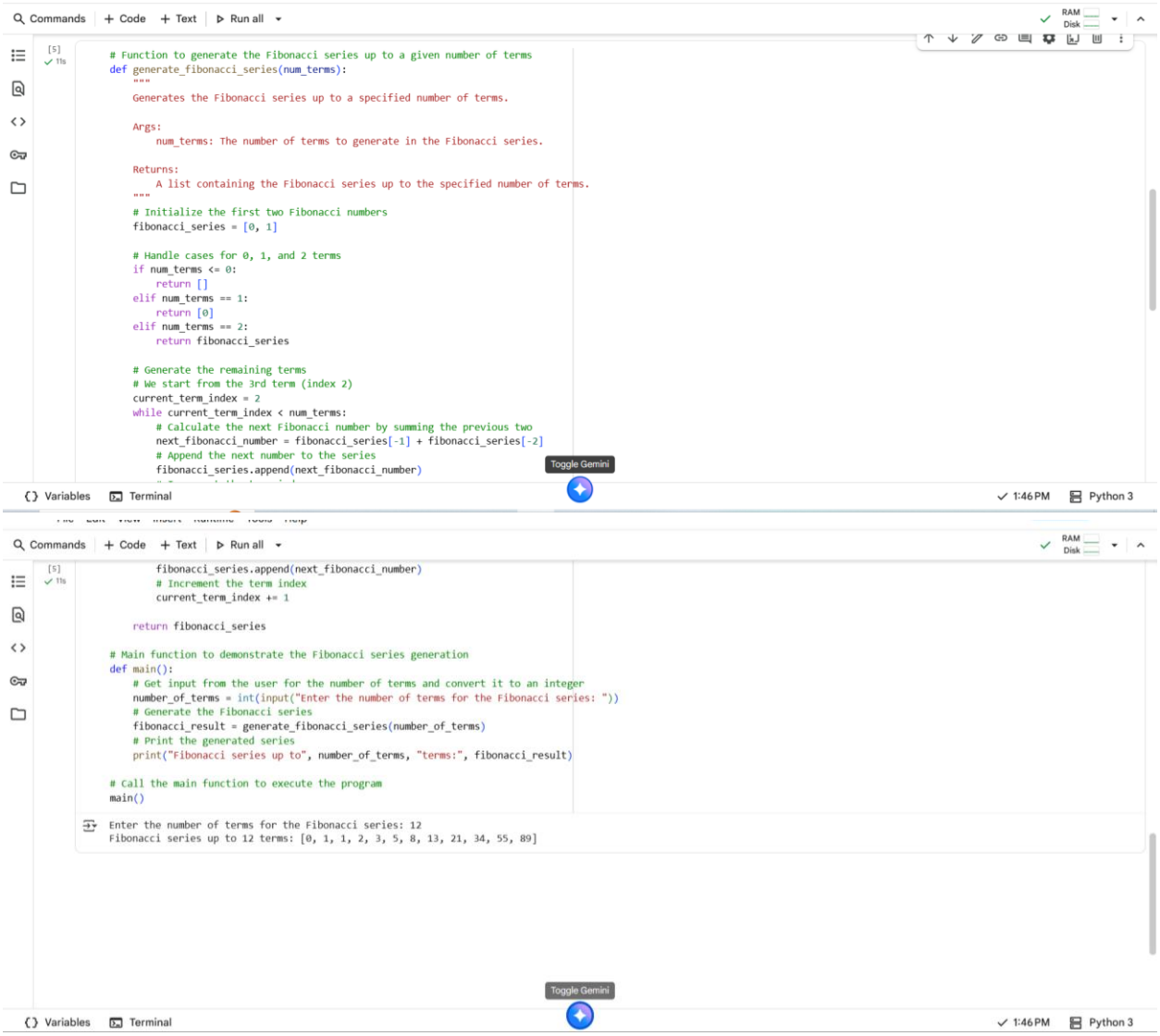
```
# Function to generate Fibonacci series up to a given number of terms
def generate_fibonacci(limit):
    # Start with the first two Fibonacci numbers
    a, b = 0, 1
    fibonacci_series = [a, b]

    # Generate Fibonacci numbers until the desired count is reached
    while len(fibonacci_series) < limit:
        next_num = a + b      # Next Fibonacci number
        fibonacci_series.append(next_num)
        a, b = b, next_num    # Move to next two numbers

    return fibonacci_series


# Main function to take user input and display Fibonacci series
def main():
    # Ask user for how many terms they want in the series
    n = int(input("Enter how many Fibonacci numbers you want: "))

    # Generate the series
    result = generate_fibonacci(n)

    # Display the output
    print("Fibonacci series till", n, "terms:", result)

# Call the main function to start the program
main()
```



```
# Call the main function to start the program
main()
```

```
Enter how many Fibonacci numbers you want: 10
Fibonacci series till 10 terms: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Code &output corrected with AI:



```
# Function to generate the Fibonacci series up to a given number of terms
def generate_fibonacci_series(num_terms):
    """
    Generates the Fibonacci series up to a specified number of terms.

    Args:
        num_terms: The number of terms to generate in the Fibonacci series.

    Returns:
        A list containing the Fibonacci series up to the specified number of terms.
    """
    # Initialize the first two Fibonacci numbers
    fibonacci_series = [0, 1]

    # Handle cases for 0, 1, and 2 terms
    if num_terms <= 0:
        return []
    elif num_terms == 1:
        return [0]
    elif num_terms == 2:
        return fibonacci_series

    # Generate the remaining terms
    # We start from the 3rd term (index 2)
    current_term_index = 2
    while current_term_index < num_terms:
        # Calculate the next Fibonacci number by summing the previous two
        next_fibonacci_number = fibonacci_series[-1] + fibonacci_series[-2]
        # Append the next number to the series
        fibonacci_series.append(next_fibonacci_number)
        # Increment the term index
        current_term_index += 1

    return fibonacci_series

# Main function to demonstrate the Fibonacci series generation
def main():
    # Get input from the user for the number of terms and convert it to an integer
    number_of_terms = int(input("Enter the number of terms for the Fibonacci series: "))
    # Generate the Fibonacci series
    fibonacci_result = generate_fibonacci_series(number_of_terms)
    # Print the generated series
    print("Fibonacci series up to", number_of_terms, "terms:", fibonacci_result)

# Call the main function to execute the program
main()
```

```
Enter the number of terms for the Fibonacci series: 12
Fibonacci series up to 12 terms: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Compare both codes :

- By comparing both codes almost both codes are similar code and similar output I got , but AI gives the diffent ways for output.

EXPLANATION:

- `generate_fibonacci_series(num_terms)`: This function is designed to produce the Fibonacci sequence. It takes one argument, `num_terms`, which is the desired length of the series. It

starts by creating a list `fibonacci_series` with the initial two numbers of the sequence, 0 and 1. It includes checks for edge cases where `num_terms` is 0, 1, or 2. If `num_terms` is greater than 2, it enters a `while` loop that continues until the list contains the specified number of terms. Inside the loop, it calculates the next Fibonacci number by adding the last two numbers in the list and then appends this new number to the list.

- `main()`: This function is the main execution part of the script. It prompts the user to enter the number of terms for the Fibonacci series using the `input()` function and converts this input to an integer. It then calls the `generate_fibonacci_series()` function with the user's input to get the Fibonacci sequence. Finally, it prints the generated Fibonacci series to the console.

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:

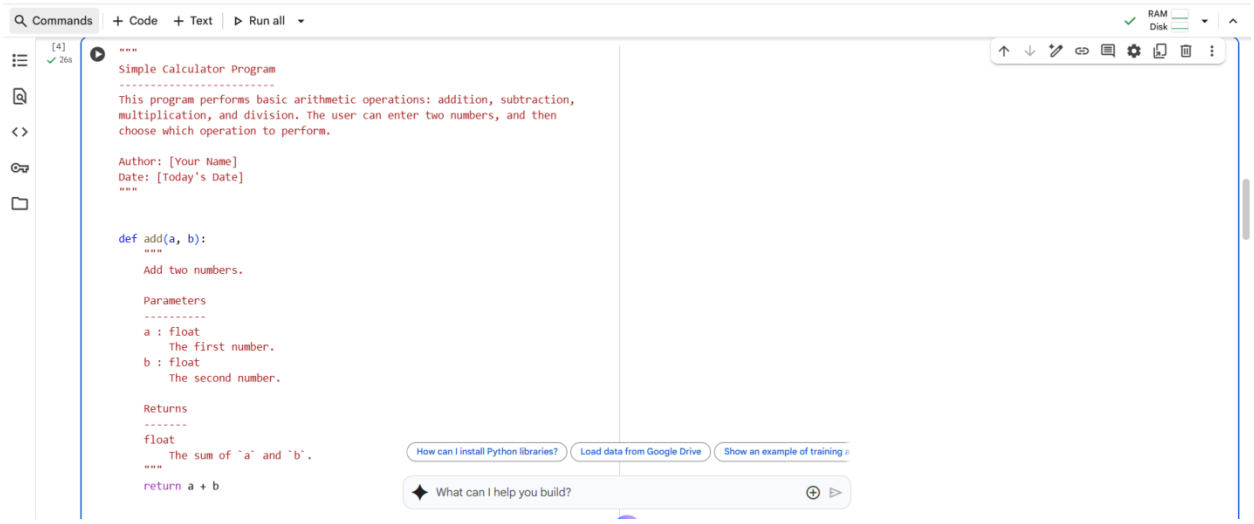
Write a Python program that acts as a simple calculator with 3–4 functions (add, subtract, multiply, divide).

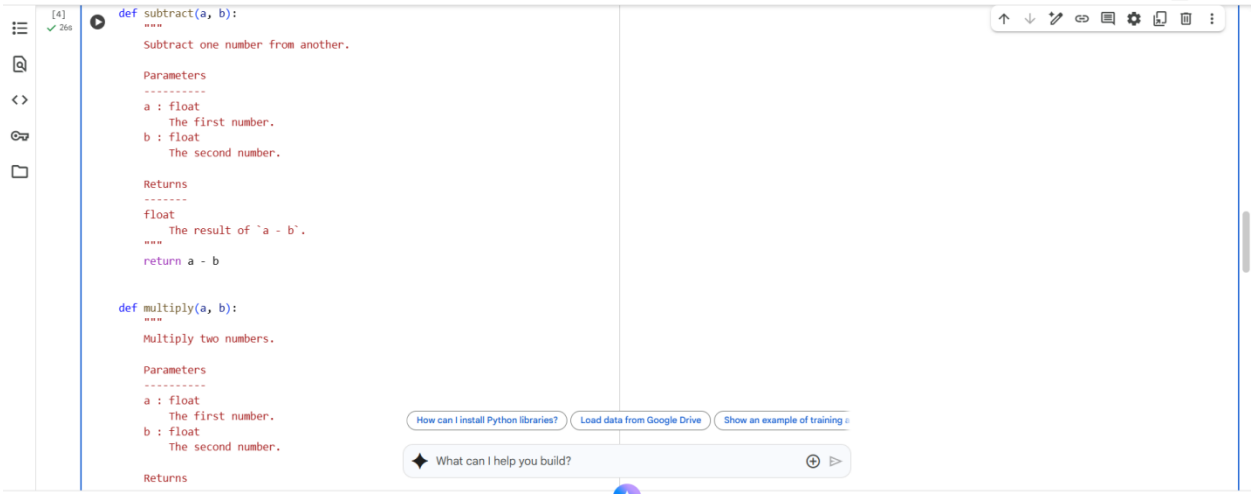
- Include **manual NumPy-style docstrings** for the module and each function.
- Ensure each function has clear parameter and return type descriptions.
- Use **AI assistance** to generate docstrings and compare them with your own manual ones.
- Make the code clean, readable, and properly formatted (PEP8 style).

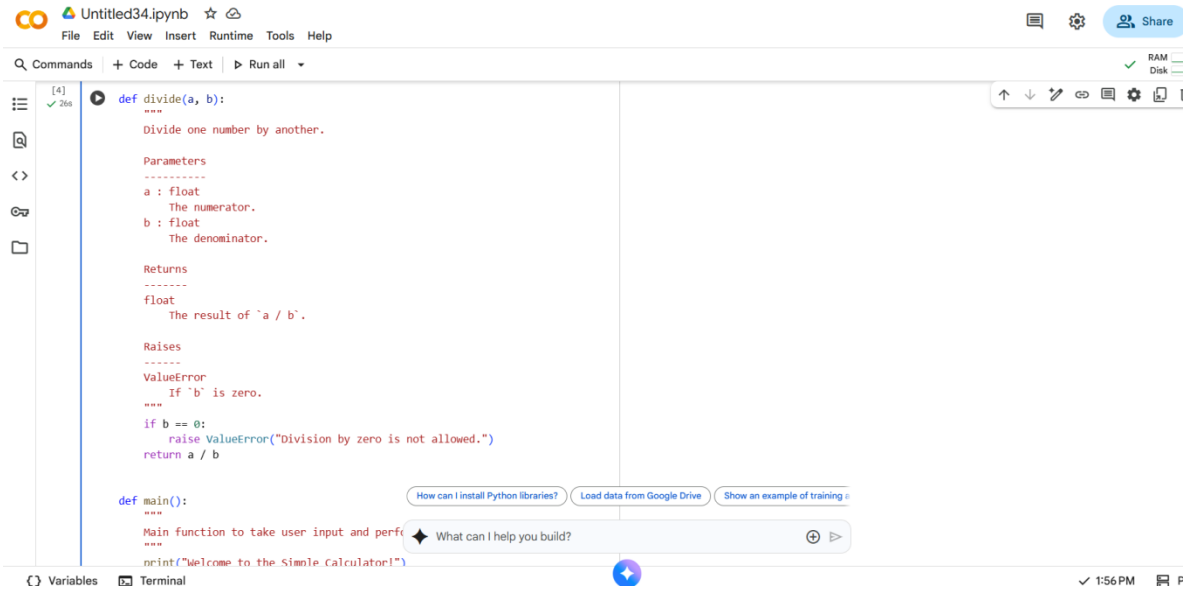
Example functions to include:

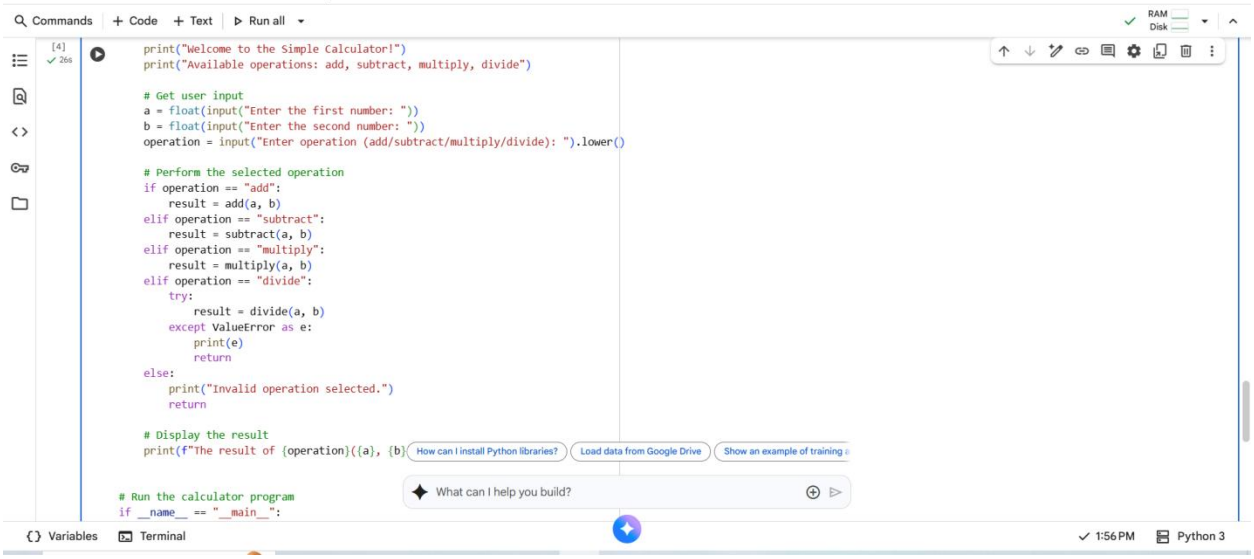
```
def add(a, b):
    pass


def subtract(a, b):
    pass


def multiply(a, b):
    pass


def divide(a, b):
    pass
```

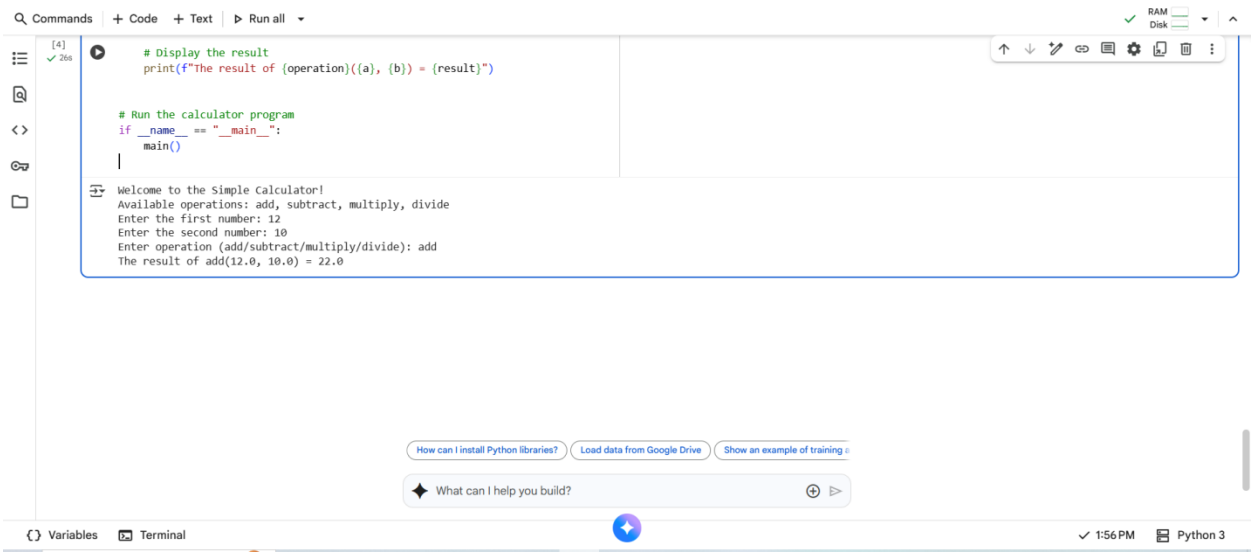
Code & output written by manual:


Commands
+ Code
+ Text
Run all
RAM
Disk
[4]
26s
"""
Simple Calculator Program
This program performs basic arithmetic operations: addition, subtraction,
multiplication, and division. The user can enter two numbers, and then
choose which operation to perform.
Author: [Your Name]
Date: [Today's Date]
"""
def add(a, b):
"""
Add two numbers.
Parameters
a : float
The first number.
b : float
The second number.
Returns
float
The sum of `a` and `b`.
"""
return a + b
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?

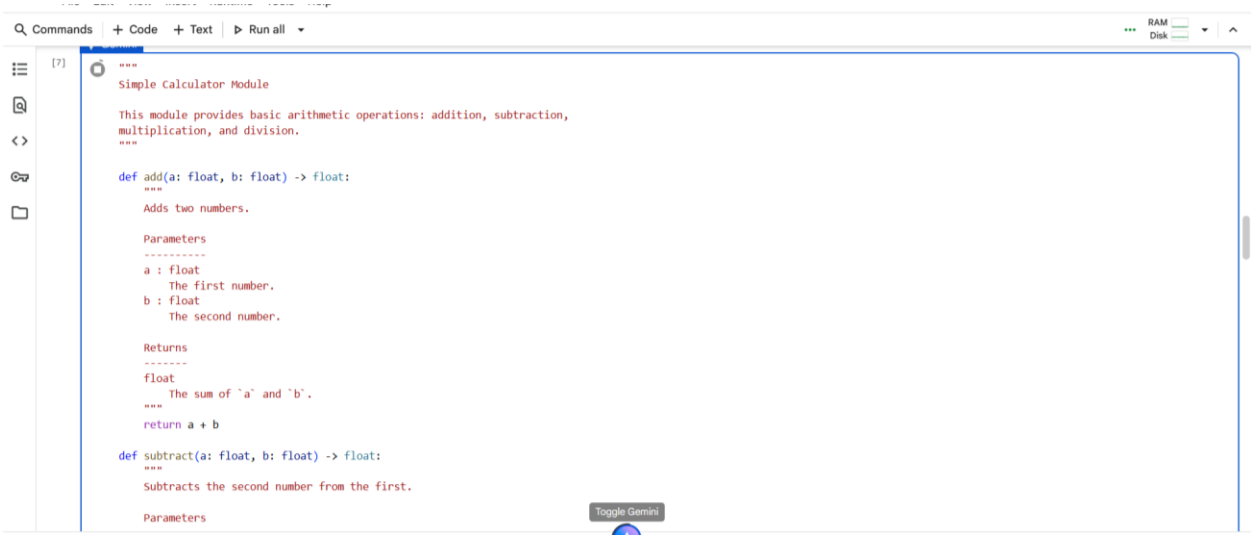

[4]
26s
def subtract(a, b):
"""
Subtract one number from another.
Parameters
a : float
The first number.
b : float
The second number.
Returns
float
The result of `a - b`.
"""
return a - b
def multiply(a, b):
"""
Multiply two numbers.
Parameters
a : float
The first number.
b : float
The second number.
Returns
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?


Untitled34.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands
+ Code
+ Text
Run all
RAM
Disk
[4]
26s
def divide(a, b):
"""
Divide one number by another.
Parameters
a : float
The numerator.
b : float
The denominator.
Returns
float
The result of `a / b`.
Raises
ValueError
If `b` is zero.
"""
if b == 0:
raise ValueError("Division by zero is not allowed.")
return a / b
def main():
"""
Main function to take user input and perf
"""
print("Welcome to the Simple Calculator!")
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?
Variables
Terminal
1:56 PM



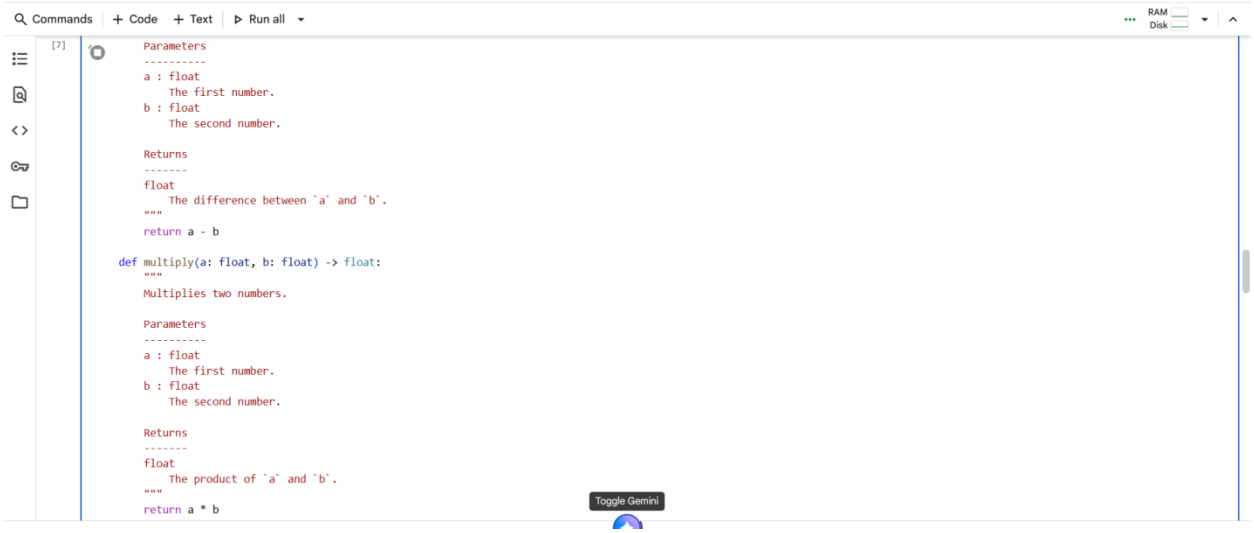


Code & output using AI:



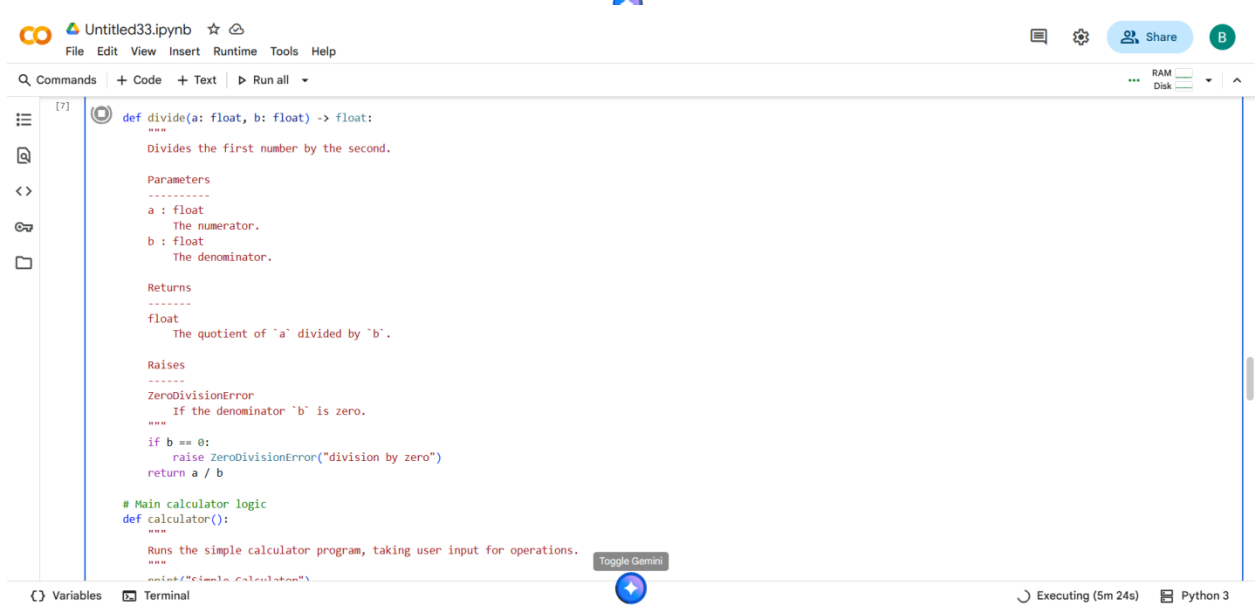
```
"""
Simple Calculator Module

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

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

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

    Returns
    -------
    float
        The sum of `a` and `b`.
    """
    return a + b

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

    Parameters
```

Toggle Gemini



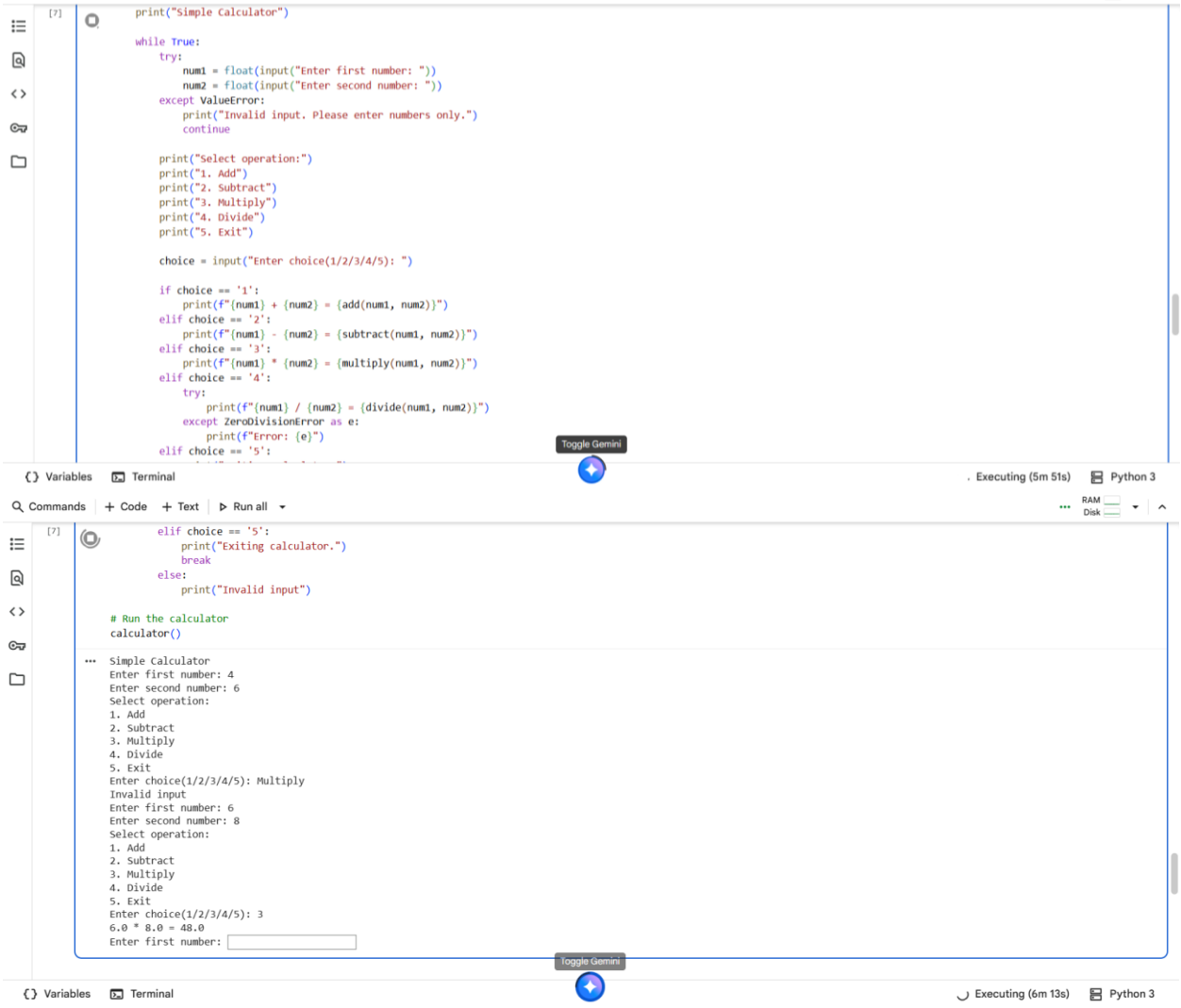
```
    Parameters
    ----------
    a : float
        The first number.
    b : float
        The second number.

    Returns
    -------
    float
        The difference between `a` and `b`.
    """
    return a - b

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

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

    Returns
    -------
    float
        The product of `a` and `b`.
    """
    return a * b
```

Toggle Gemini



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

    Parameters
    ----------
    a : float
        The numerator.
    b : float
        The denominator.

    Returns
    -------
    float
        The quotient of `a` divided by `b`.

    Raises
    ------
    ZeroDivisionError
        If the denominator `b` is zero.
    """
    if b == 0:
        raise ZeroDivisionError("division by zero")
    return a / b

# Main calculator logic
def calculator():
    """
    Runs the simple calculator program, taking user input for operations.
    """
```

Toggle Gemini



```
print("Simple Calculator")

while True:
    try:
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))
    except ValueError:
        print("Invalid input. Please enter numbers only.")
        continue

    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")
    print("5. Exit")

    choice = input("Enter choice(1/2/3/4/5): ")

    if choice == '1':
        print(f"{num1} + {num2} = {add(num1, num2)}")
    elif choice == '2':
        print(f"{num1} - {num2} = {subtract(num1, num2)}")
    elif choice == '3':
        print(f"{num1} * {num2} = {multiply(num1, num2)}")
    elif choice == '4':
        try:
            print(f"{num1} / {num2} = {divide(num1, num2)}")
        except ZeroDivisionError as e:
            print(f"Error: {e}")
    elif choice == '5':
```

Toggle Gemini

Variables Terminal Executing (5m 51s) Python 3

Commands + Code + Text Run all RAM Disk

```
    elif choice == '5':
        print("Exiting calculator.")
        break
    else:
        print("Invalid input")

# Run the calculator
calculator()
```

```
Simple Calculator
Enter first number: 4
Enter second number: 6
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Enter choice(1/2/3/4/5): Multiply
Invalid input
Enter first number: 6
Enter second number: 8
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Enter choice(1/2/3/4/5): 3
6.0 * 8.0 = 48.0
Enter first number:
```

Toggle Gemini

Variables Terminal Executing (6m 13s) Python 3

Comparing both AI and Manual code:

-By comparing both codes AI generated best code than manual code and easily understandable code.

EXPLANATION:

- **Module Docstring:** The code starts with a triple-quoted string at the beginning of the file. This is a module-level docstring that explains the overall purpose of the script – providing basic arithmetic operations.

- **`add(a, b)` function:** This function takes two floating-point numbers (`a` and `b`) as input and returns their sum. It includes a NumPy-style docstring explaining its parameters and what it returns.
- **`subtract(a, b)` function:** Similar to `add`, this function takes two floating-point numbers and returns their difference (`a - b`). It also has a NumPy-style docstring.
- **`multiply(a, b)` function:** This function takes two floating-point numbers and returns their product (`a * b`), with a corresponding docstring.
- **`divide(a, b)` function:** This function takes two floating-point numbers and returns the result of dividing `a` by `b`. It includes a check to prevent division by zero, raising a `ZeroDivisionError` if the denominator is zero. Its docstring explains the parameters, return value, and the potential exception.
- **`calculator()` function:** This is the main function that runs the interactive calculator.
    - It enters an infinite `while True` loop to keep the calculator running until the user chooses to exit.
    - Inside the loop, it prompts the user to enter two numbers and handles potential `ValueError` if the input is not a valid number.
    - It then presents a menu of operations (Add, Subtract, Multiply, Divide, Exit) and prompts the user to select one.
    - Based on the user's choice, it calls the appropriate arithmetic function and prints the result.
    - It includes specific error handling for the `divide` function to catch `ZeroDivisionError`.
    - If the user enters '5', the loop breaks, and the program exits.
    - If the user enters an invalid choice, it prints an "Invalid input" message.
- **`calculator()` call:** The last line `calculator()` calls the main calculator function to start the program.

---END---