Name: K.Likitha

Hall Ticket: 2303A52144

Batch: 41

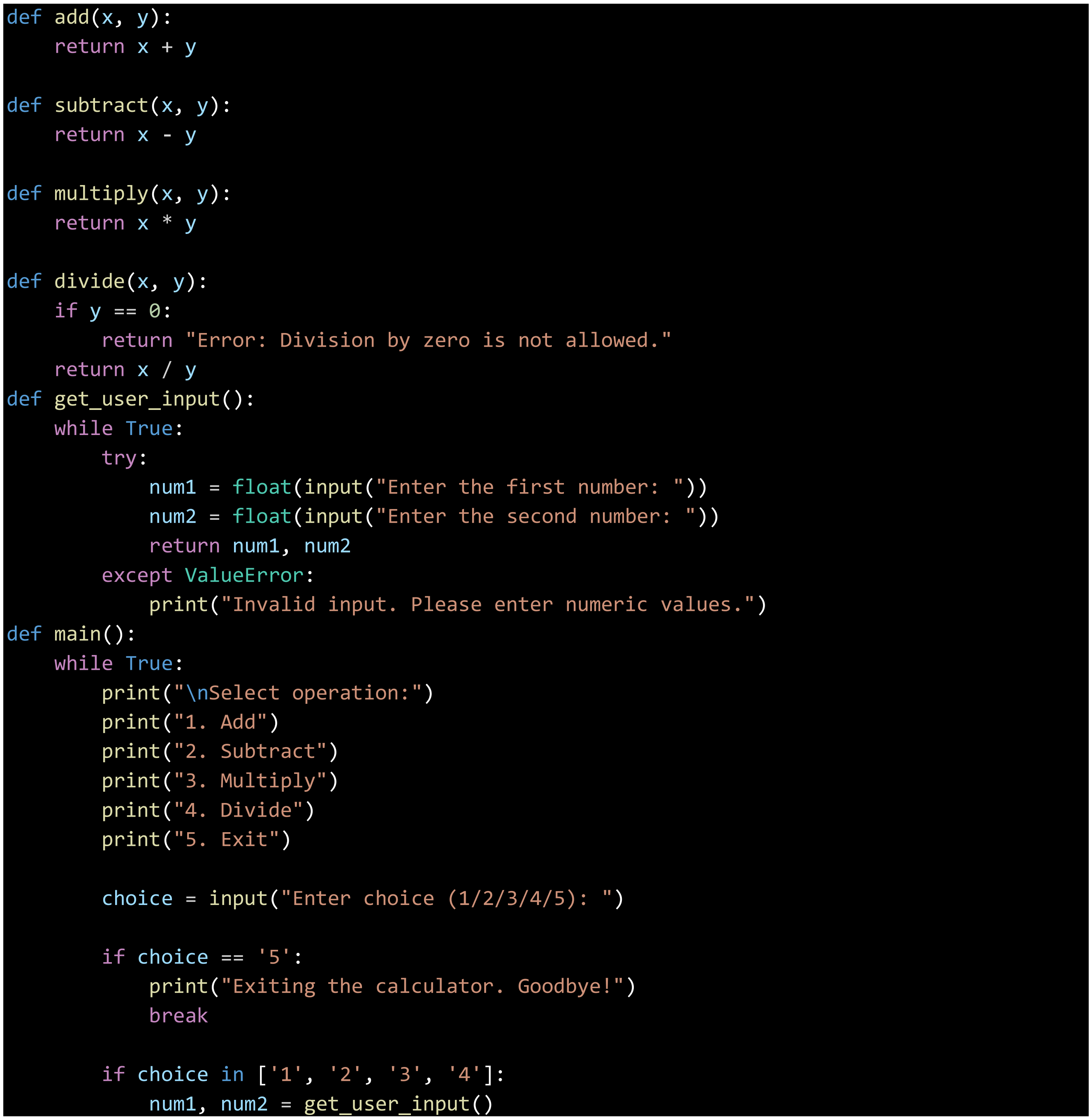
Task 1: Auto-Generating Function Documentation in a Shared Codebase

Python script containing multiple functions without any docstrings.

Prompt:

#generate python code for calculator that can perform basic arithmetic operations (addition, subtraction, multiplication, division based on user input. The calculator should continue to prompt the user for operations until they choose to exit. should contain multiple functions for each operation and a main function to run the calculator. include error handling for invalid inputs and division by zero. without any docstrings or comments.

Code:



```python
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    if y == 0:
        return "Error: Division by zero is not allowed."
    return x / y
def get_user_input():
    while True:
        try:
            num1 = float(input("Enter the first number: "))
            num2 = float(input("Enter the second number: "))
            return num1, num2
        except ValueError:
            print("Invalid input. Please enter numeric values.")
def main():
    while True:
        print("\nSelect 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 == '5':
            print("Exiting the calculator. Goodbye!")
            break

        if choice in ['1', '2', '3', '4']:
            num1, num2 = get_user_input()
            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':
                result = divide(num1, num2)
                print(f"{num1} / {num2} = {result}")
        else:
            print("Invalid choice. Please select a valid operation.")
if __name__ == "__main__":
    main()
```

Google Doc String Prompt:

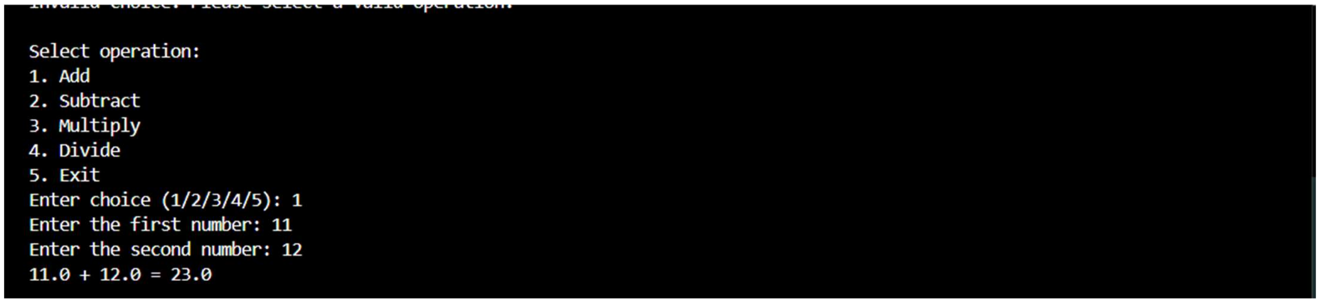
Add Google-style function docstrings to all functions in the given Python calculator code. Each docstring should be only 2–3 lines long and must clearly describe what the function does, list all parameters with their data types, specify the return value with its data type, and include at least one example usage. Do not modify the existing logic, structure, or output of the program; only insert appropriate docstrings above each function definition.

Code:

```
def add(x, y):
    """Add two numbers.

    Args:
        x (float): The first number.
        y (float): The second number.

    Returns:
        float: The sum of x and y.

    Example:
        >>> add(2, 3)
        5
    """
    return x + y
def subtract(x, y):
    """Subtract two numbers.

    Args:
        x (float): The first number.
        y (float): The second number.

    Returns:
        float: The difference of x and y.

    Example:
        >>> subtract(5, 3)
        2
    """
    return x - y
def multiply(x, y):
```

```python
    """Multiply two numbers.

    Args:
        x (float): The first number.
        y (float): The second number.

    Returns:
        float: The product of x and y.

    Example:
        >>> multiply(2, 3)
        6
    """
    return x * y
def divide(x, y):
    """Divide two numbers.

    Args:
        x (float): The numerator.
        y (float): The denominator.

    Returns:
        float or str: The quotient of x and y, or an error message if division by zero.

    Example:
        >>> divide(6, 3)
        2.0
    """
    if y == 0:
        return "Error: Division by zero is not allowed."
    return x / y
def get_user_input():
    """Get two numeric inputs from the user.

    Returns:
        tuple: A tuple containing two floats entered by the user.
    Example:
        >>> get_user_input()
        (2.0, 3.0)
    """
    while True:
        try:
            num1 = float(input("Enter the first number: "))
            num2 = float(input("Enter the second number: "))
            return num1, num2
        except ValueError:
            print("Invalid input. Please enter numeric values.")
def main():
    """Run the interactive calculator.

    Continuously prompts the user to select an arithmetic operation and input numbers
until they choose to exit.

    Example:
        >>> main()
        (Interactive session)
    """
    while True:
        print("\nSelect 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 == '5':
            print("Exiting the calculator. Goodbye!")
            break

        if choice in ['1', '2', '3', '4']:
            num1, num2 = get_user_input()

            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':
                result = divide(num1, num2)
                print(f"{num1} / {num2} = {result}")
        else:
            print("Invalid choice. Please select a valid operation.")
if __name__ == "__main__":
    main()
```

Output:



```
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Enter choice (1/2/3/4/5): 1
Enter the first number: 11
Enter the second number: 12
11.0 + 12.0 = 23.0
```

Explanation:

This code implements a simple menu-driven calculator that performs addition, subtraction, multiplication, and division based on user input. It uses functions for each operation and includes input validation to handle invalid values and division by zero.

Task 2: Enhancing Readability Through AI-Generated Inline Comments

Python Code for Bubble Sort:

Prompt:

Write a Python program for bubble sort. should take user input for the list of numbers to sort.

Code:

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

numbers = input("Enter the list of numbers to sort: ")
numbers = numbers.split()
numbers = [int(num) for num in numbers]
sorted_numbers = bubble_sort(numbers)
print("Sorted numbers are:", sorted_numbers)
```

Updated Prompt:

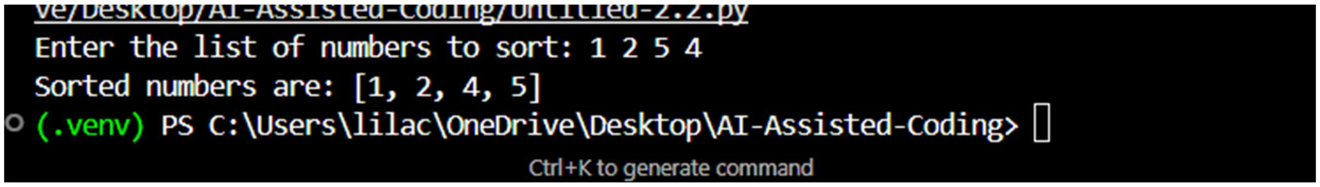
from the above program, Automatically insert inline comments only for complex or non-obvious logic

and also add docstrings for the function and the program.

Code:

```python
def bubble_sort(arr):
    """
    Bubble sort algorithm to sort the list of numbers.
    """
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

numbers = input("Enter the list of numbers to sort: ")
numbers = numbers.split()
numbers = [int(num) for num in numbers]
sorted_numbers = bubble_sort(numbers)
print("Sorted numbers are:", sorted_numbers)
```

Output:



```
ve/Desktop/AI-Assisted-Coding/Untitled-2.2.py
Enter the list of numbers to sort: 1 2 5 4
Sorted numbers are: [1, 2, 4, 5]
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding>
```

Explanation:

AI assistance is used to add inline comments only where the logic is complex or hard to understand, while skipping obvious code. This helps improve readability without overcrowding the program with unnecessary comments.

Task 3: Generating Module-Level Documentation for a Python Package

Prompt:

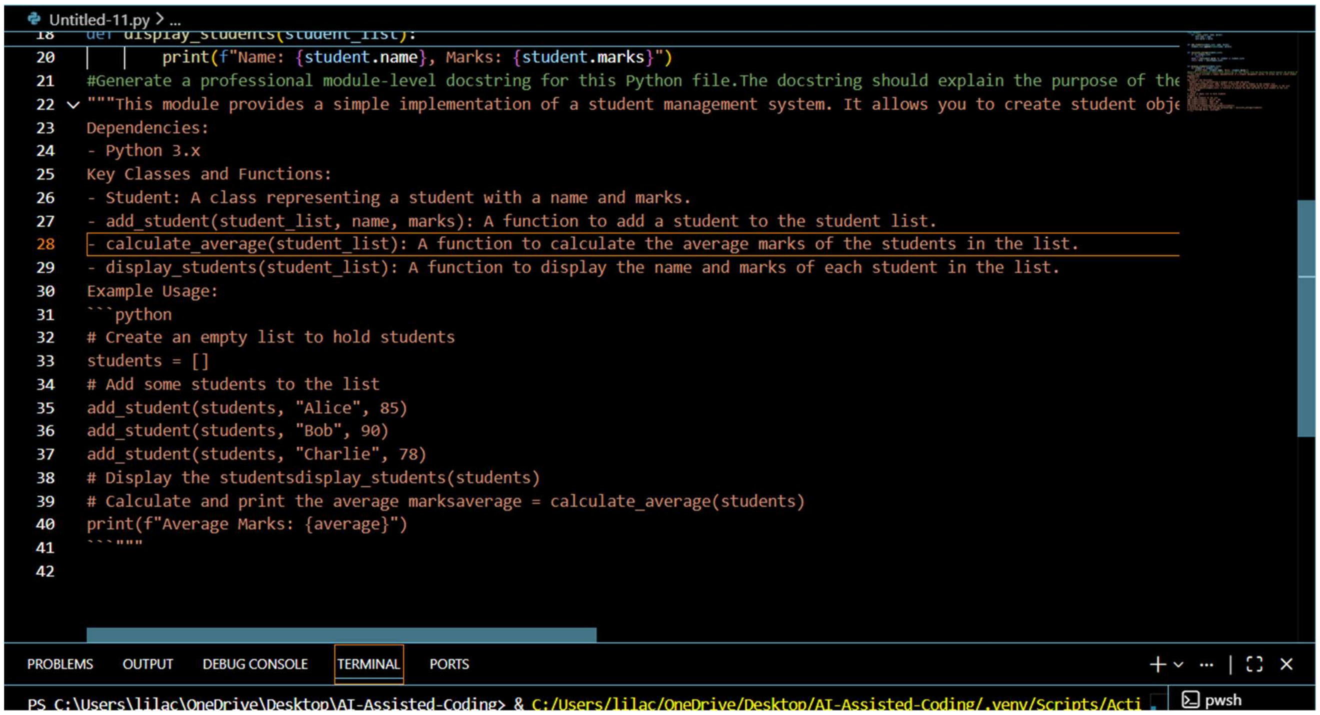
Generate a professional module-level docstring for this Python file.The docstring should explain the purpose of the module, required dependencies, describe the key classes and functions, and include a short example of how the module can be used.

Code:

~~~
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks


def add_student(student_list, name, marks):
    student_list.append(Student(name, marks))


def calculate_average(student_list):
    if not student_list:
        return 0
    total = sum(student.marks for student in student_list)
    return total / len(student_list)


def display_students(student_list):
    for student in student_list:
        print(f"Name: {student.name}, Marks: {student.marks}")
#Generate a professional module-level docstring for this Python file.The docstring should
explain the purpose of the module, required dependencies, describe the key classes and
functions, and include a short example of how the module can be used.
"""This module provides a simple implementation of a student management system. It allows
you to create student objects, add them to a list, calculate the average marks of the
students, and display their information.
Dependencies:
- Python 3.x
Key Classes and Functions:
- Student: A class representing a student with a name and marks.
- add_student(student_list, name, marks): A function to add a student to the student list.
- calculate_average(student_list): A function to calculate the average marks of the
students in the list.
- display_students(student_list): A function to display the name and marks of each student
in the list.
Example Usage:
```python
# Create an empty list to hold students
students = []
# Add some students to the list
add_student(students, "Alice", 85)
~~~

~~~
add_student(students, "Bob", 90)
add_student(students, "Charlie", 78)
# Display the studentsdisplay_students(students)
# Calculate and print the average marksaverage = calculate_average(students)
print(f"Average Marks: {average}")
```"""
~~~

Output:



~~~
def display_students(student_list):
        print(f"Name: {student.name}, Marks: {student.marks}")
#Generate a professional module-level docstring for this Python file.The docstring should explain the purpose of the
"""This module provides a simple implementation of a student management system. It allows you to create student obje
Dependencies:
- Python 3.x
Key Classes and Functions:
- Student: A class representing a student with a name and marks.
- add_student(student_list, name, marks): A function to add a student to the student list.
- calculate_average(student_list): A function to calculate the average marks of the students in the list.
- display_students(student_list): A function to display the name and marks of each student in the list.
Example Usage:
```python
# Create an empty list to hold students
students = []
# Add some students to the list
add_student(students, "Alice", 85)
add_student(students, "Bob", 90)
add_student(students, "Charlie", 78)
# Display the studentsdisplay_students(students)
# Calculate and print the average marksaverage = calculate_average(students)
print(f"Average Marks: {average}")
```"""
~~~

```
PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> & C:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/.venv/Scripts/Acti
```

Explanation:

This task demonstrates how to design a simple Python module and use an AI tool to generate clear, professional documentation.

Task 4: Converting Developer Comments into Structured Docstrings

Prompt:

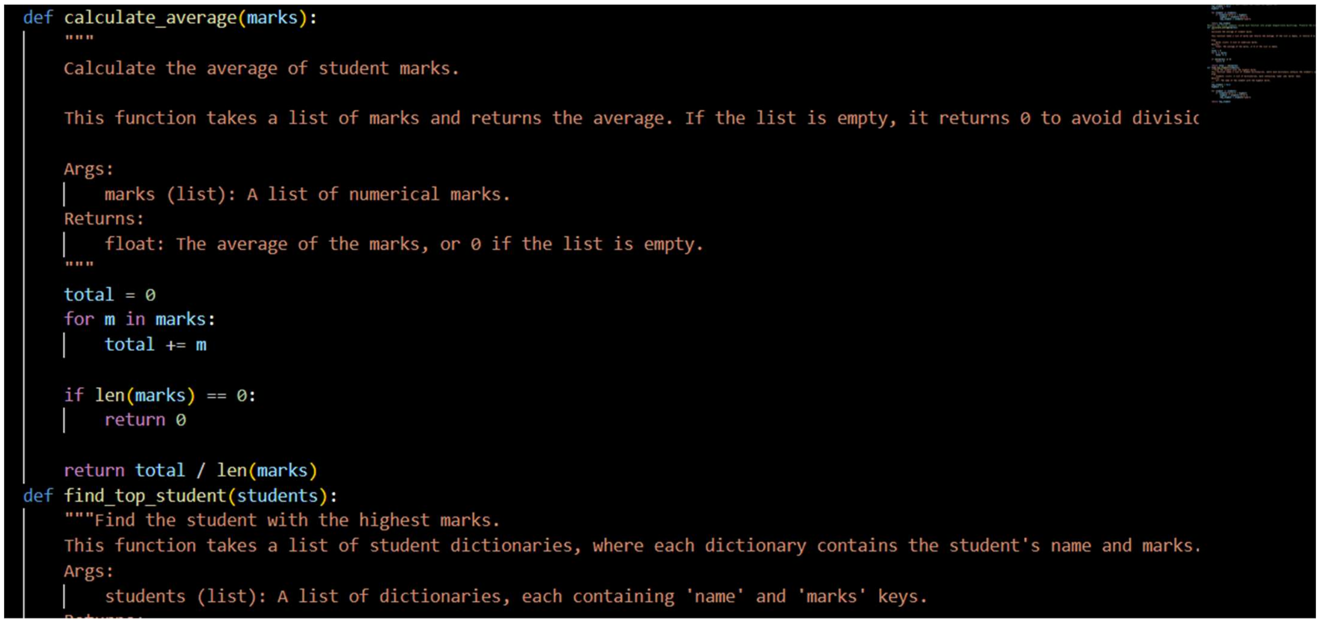
Convert the inline comments inside each function into proper Google-style docstrings. Preserve the original meaning, remove redundant inline comments, and keep the function logic unchanged.

Code:

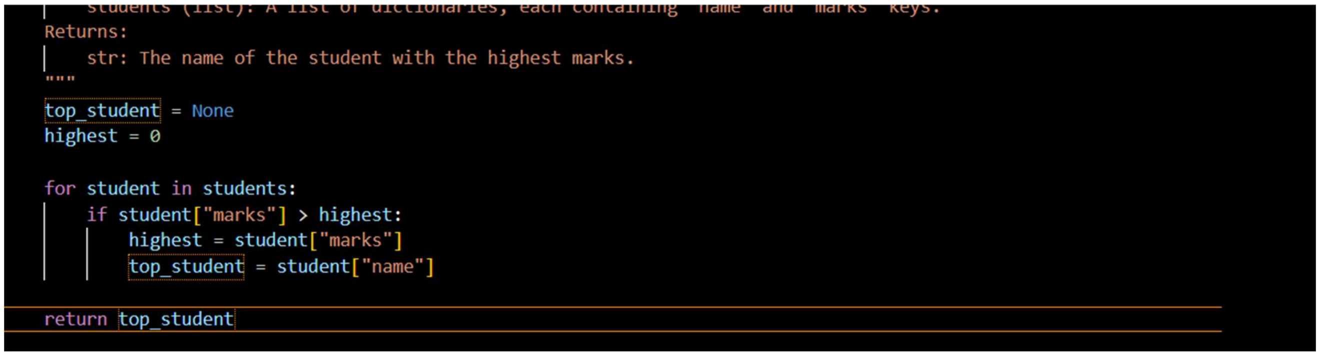
```python
def calculate_average(marks):
    # This function calculates the average of student marks
    # First, it checks whether the marks list is empty
    # If the list is empty, it returns 0 to avoid division errors
    # Then it adds all the marks in the list
    # Finally, it divides the total by the number of marks
    total = 0
    for m in marks:
        total += m

    if len(marks) == 0:
        return 0
```

```python
    return total / len(marks)


def find_top_student(students):
    # This function finds the student with the highest marks
    # It assumes students is a list of dictionaries
    # Each dictionary contains the student's name and marks
    # It compares marks of each student and tracks the highest one
    top_student = None
    highest = 0

    for student in students:
        if student["marks"] > highest:
            highest = student["marks"]
            top_student = student["name"]

    return top_student
#Convert the inline comments inside each function into proper Google-style docstrings.
Preserve the original meaning, remove redundant inline comments, and keep the function
logic unchanged.
def calculate_average(marks):
    """
    Calculate the average of student marks.

    This function takes a list of marks and returns the average. If the list is empty, it
returns 0 to avoid division errors.

    Args:
        marks (list): A list of numerical marks.
    Returns:
        float: The average of the marks, or 0 if the list is empty.
    """
    total = 0
    for m in marks:
        total += m

    if len(marks) == 0:
        return 0

    return total / len(marks)
def find_top_student(students):
    """Find the student with the highest marks.
    This function takes a list of student dictionaries, where each dictionary contains the
student's name and marks. It compares the marks of each student and returns the name of
the student with the highest marks.
    Args:
        students (list): A list of dictionaries, each containing 'name' and 'marks' keys.
    Returns:
        str: The name of the student with the highest marks.
    """
    top_student = None
    highest = 0
    for student in students:
        if student["marks"] > highest:
            highest = student["marks"]
            top_student = student["name"]

    return top_student
```

Output:



```
def calculate_average(marks):
    """
    Calculate the average of student marks.

    This function takes a list of marks and returns the average. If the list is empty, it returns 0 to avoid divisic

    Args:
        marks (list): A list of numerical marks.
    Returns:
        float: The average of the marks, or 0 if the list is empty.
    """
    total = 0
    for m in marks:
        total += m

    if len(marks) == 0:
        return 0

    return total / len(marks)
def find_top_student(students):
    """Find the student with the highest marks.
    This function takes a list of student dictionaries, where each dictionary contains the student's name and marks.
    Args:
        students (list): A list of dictionaries, each containing 'name' and 'marks' keys.
```



```
        students (list): A list of dictionaries, each containing 'name' and 'marks' keys.
```

```
    Returns:
        str: The name of the student with the highest marks.
    """
    top_student = None
    highest = 0

    for student in students:
        if student["marks"] > highest:
            highest = student["marks"]
            top_student = student["name"]

    return top_student
```

Explanation:

The code processes student data by calculating average marks and identifying the top-performing student. Inline comments are replaced with structured docstrings to improve readability and maintain consistent documentation.

Task 5: Building a Mini Automatic Documentation Generator

Prompt:

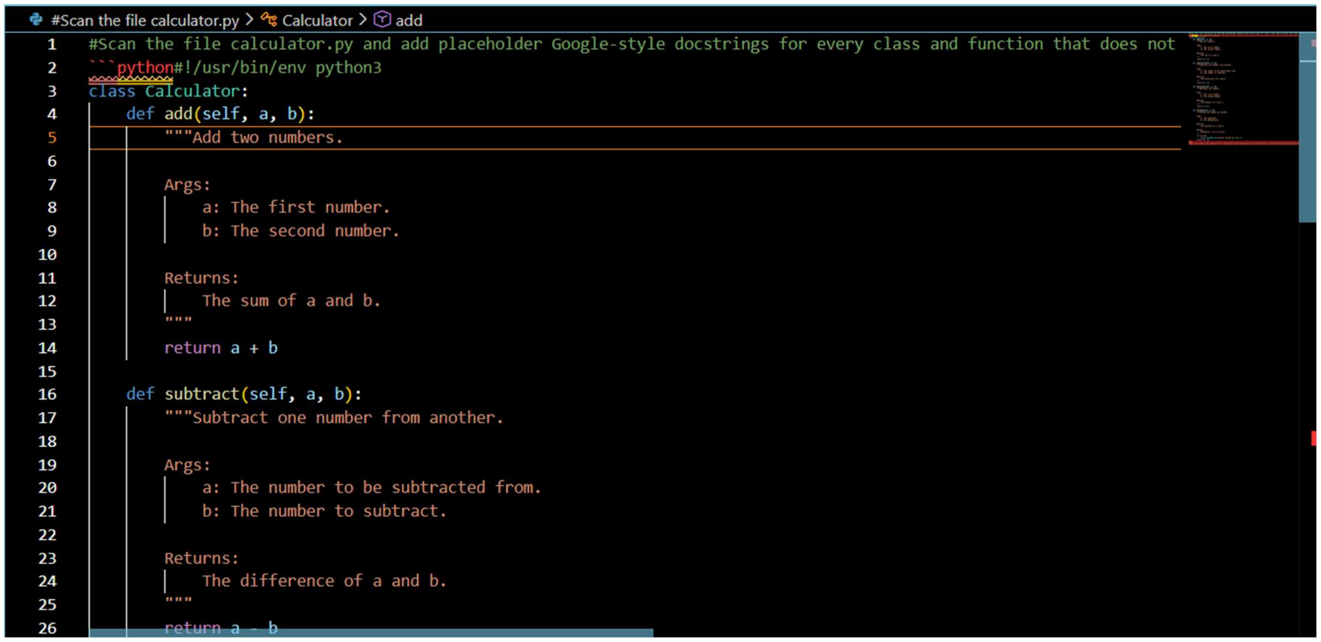
Scan the file calculator.py and add placeholder Google-style docstrings for every class and function that does not already have one. Do not change any existing logic or behavior; only insert clean, structured docstrings.

Code:

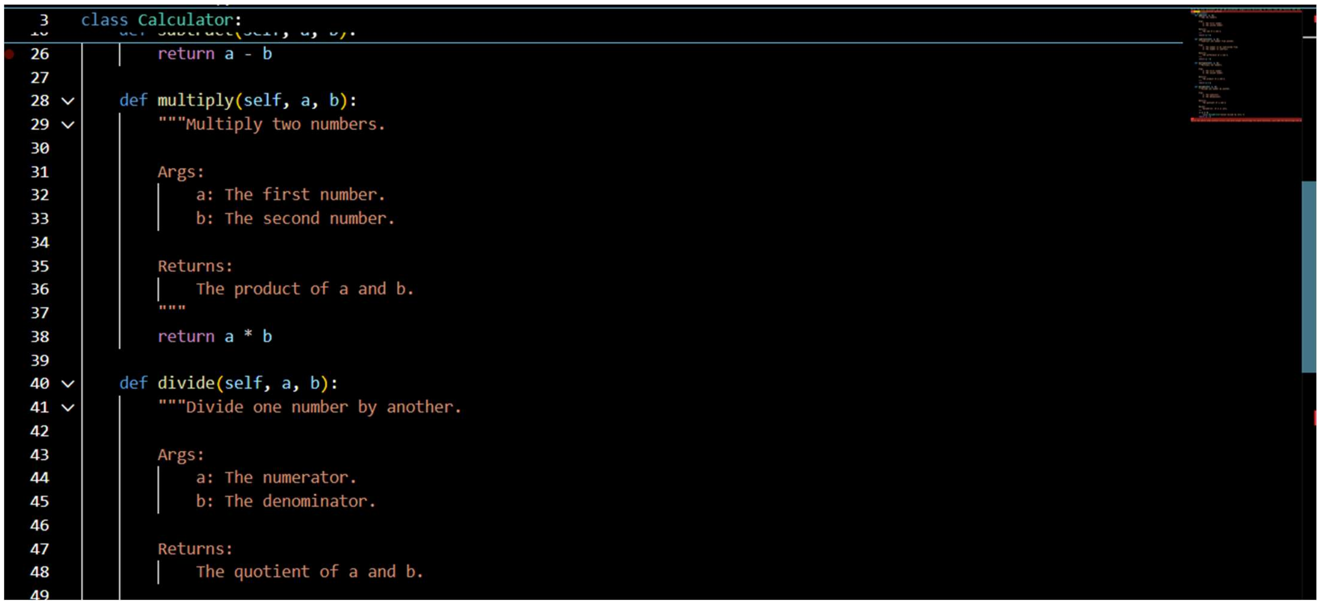
```
class Calculator:
    def add(self, a, b):
        return a + b
```

```python
    def subtract(self, a, b):
        return a - b


def multiply(a, b):
    return a * b


def divide(a, b):
    if b == 0:
        return None
    return a / b
```

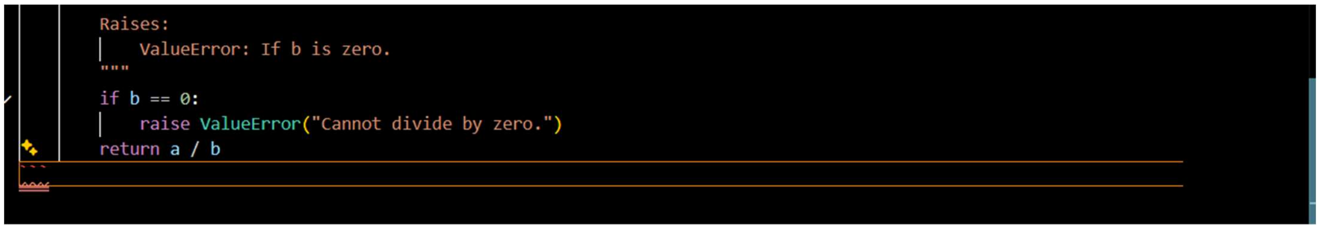
Output:



~~~python
#Scan the file calculator.py and add placeholder Google-style docstrings for every class and function that does not
```python#!/usr/bin/env python3
class Calculator:
    def add(self, a, b):
        """Add two numbers.

        Args:
            a: The first number.
            b: The second number.

        Returns:
            The sum of a and b.
        """
        return a + b

    def subtract(self, a, b):
        """Subtract one number from another.

        Args:
            a: The number to be subtracted from.
            b: The number to subtract.

        Returns:
            The difference of a and b.
        """
        return a - b
~~~



```python
class Calculator:
        return a - b

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

        Args:
            a: The first number.
            b: The second number.

        Returns:
            The product of a and b.
        """
        return a * b

    def divide(self, a, b):
        """Divide one number by another.

        Args:
            a: The numerator.
            b: The denominator.

        Returns:
            The quotient of a and b.
```



```
        Raises:
            ValueError: If b is zero.
        """
        if b == 0:
            raise ValueError("Cannot divide by zero.")
        return a / b
```
```

Explanation:

The code defines a Calculator class that performs basic arithmetic operations with clear Google-style docstrings. Each method documents its purpose, parameters, return values, and errors, improving readability and maintainability.
```