Assignment-9.3

2303A51304

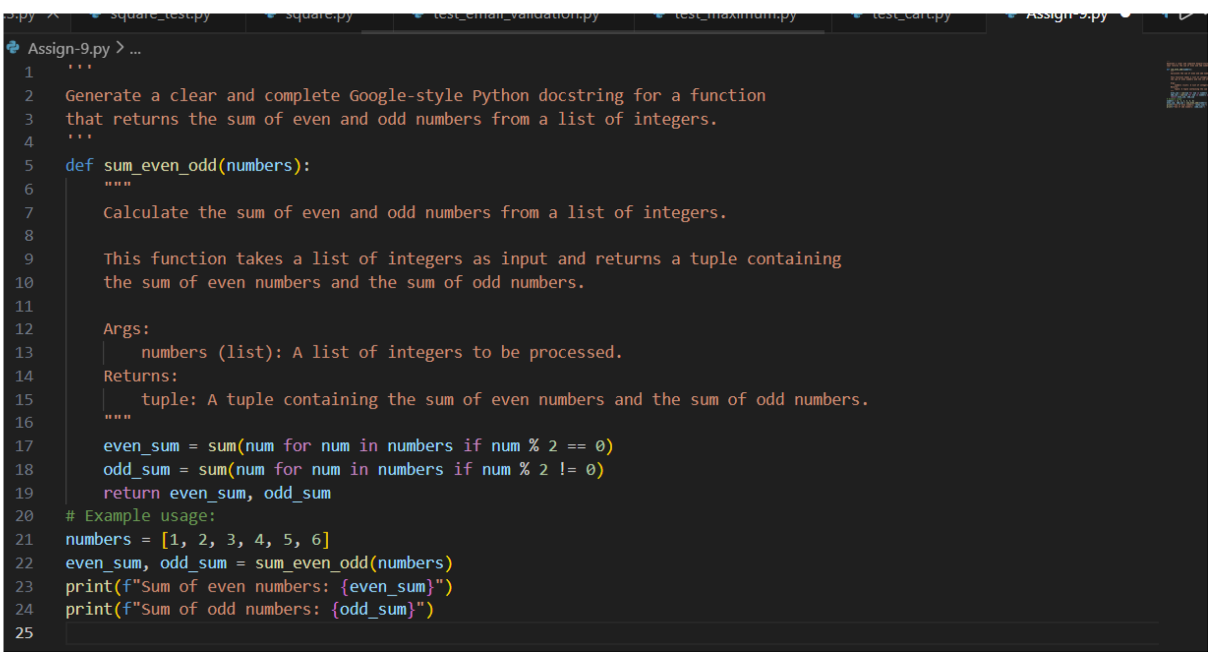
Batch-05

Task 1: Basic Docstring Generation

Prompt:

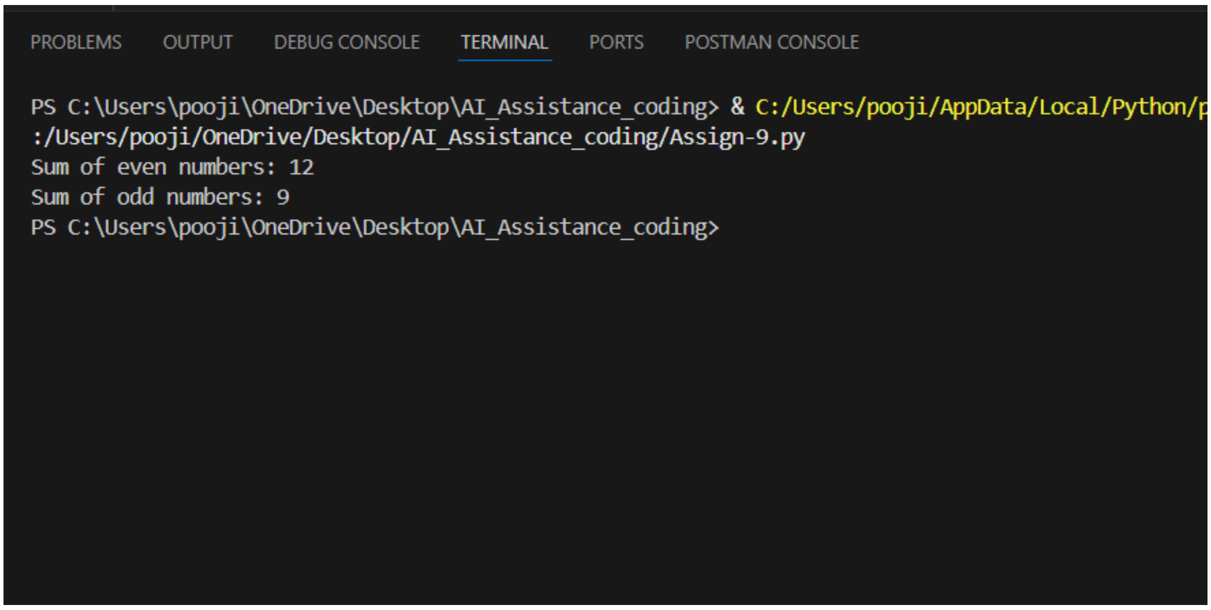
Generate a clear and complete Google-style Python docstring for a function that returns the sum of even and odd numbers from a list of integers.

Code:



```python
'''
Generate a clear and complete Google-style Python docstring for a function
that returns the sum of even and odd numbers from a list of integers.
'''
def sum_even_odd(numbers):
    """
    Calculate the sum of even and odd numbers from a list of integers.

    This function takes a list of integers as input and returns a tuple containing
    the sum of even numbers and the sum of odd numbers.

    Args:
        numbers (list): A list of integers to be processed.
    Returns:
        tuple: A tuple containing the sum of even numbers and the sum of odd numbers.
    """
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    return even_sum, odd_sum
# Example usage:
numbers = [1, 2, 3, 4, 5, 6]
even_sum, odd_sum = sum_even_odd(numbers)
print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

Output:



```
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/p
:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/Assign-9.py
Sum of even numbers: 12
Sum of odd numbers: 9
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

Manual documentation is more descriptive and easier for beginners to understand. AI-generated docstrings are concise and professional but may lack detailed explanation.
AI saves time and ensures consistency in format.
Manual docstrings provide better contextual clarity.
Best practice is AI-generated base + human refinement

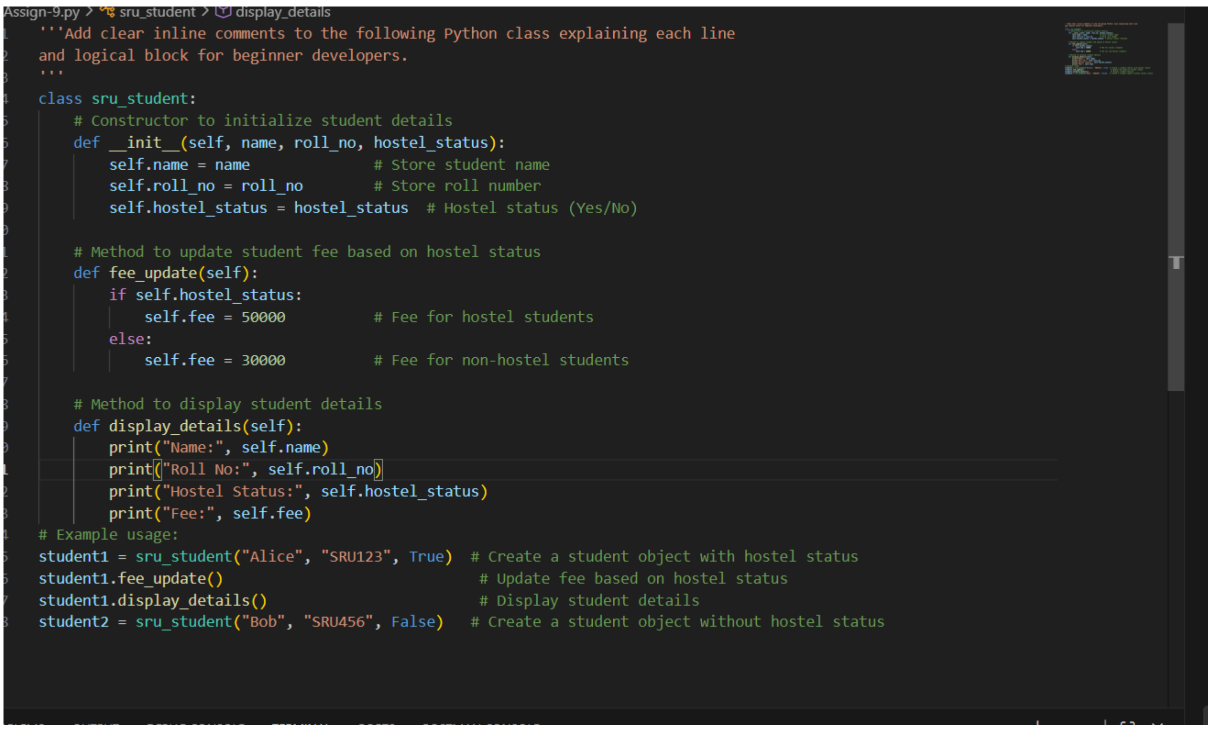
Task-02:

Prompt:

Add clear inline comments to the following Python class explaining each line and logical block for beginner developers.

Code:

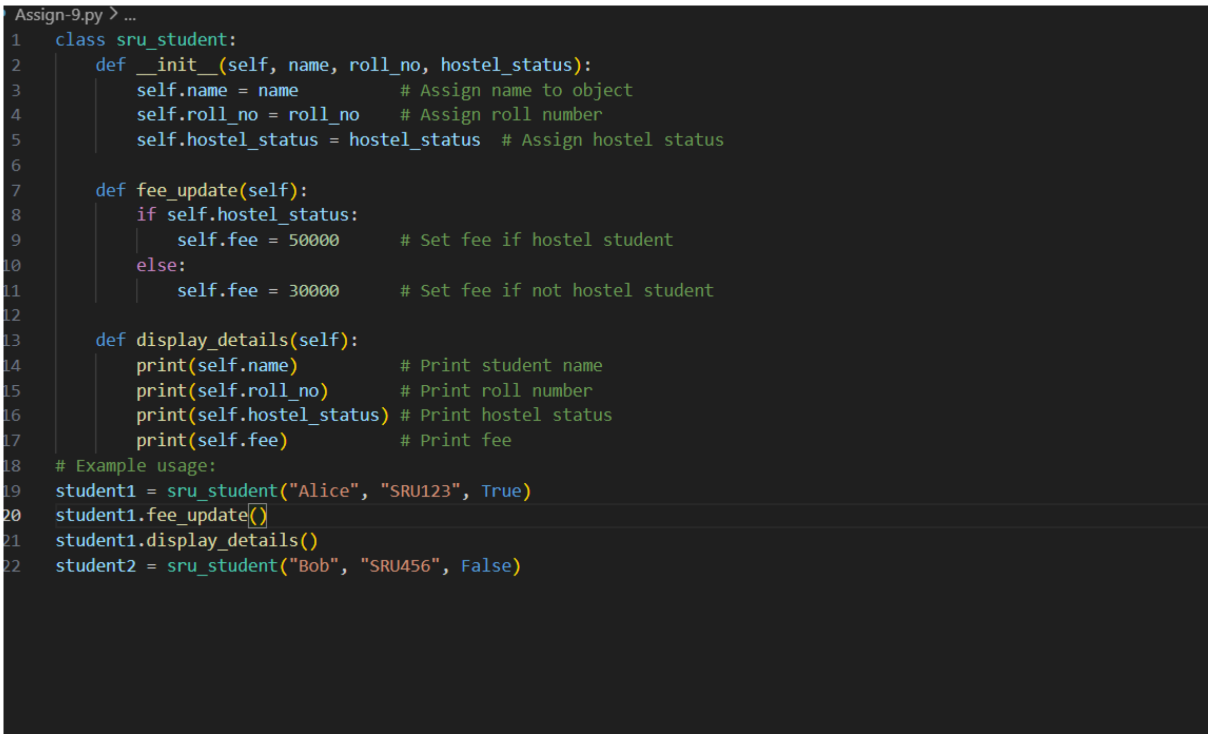
Python Class with Manual Inline Comments



Assign-9.py > sru_student > display_details

```python
'''Add clear inline comments to the following Python class explaining each line
and logical block for beginner developers.
'''
class sru_student:
    # Constructor to initialize student details
    def __init__(self, name, roll_no, hostel_status):
        self.name = name              # Store student name
        self.roll_no = roll_no        # Store roll number
        self.hostel_status = hostel_status  # Hostel status (Yes/No)

    # Method to update student fee based on hostel status
    def fee_update(self):
        if self.hostel_status:
            self.fee = 50000          # Fee for hostel students
        else:
            self.fee = 30000          # Fee for non-hostel students

    # Method to display student details
    def display_details(self):
        print("Name:", self.name)
        print("Roll No:", self.roll_no)
        print("Hostel Status:", self.hostel_status)
        print("Fee:", self.fee)
# Example usage:
student1 = sru_student("Alice", "SRU123", True)  # Create a student object with hostel status
student1.fee_update()                             # Update fee based on hostel status
student1.display_details()                        # Display student details
student2 = sru_student("Bob", "SRU456", False)   # Create a student object without hostel status
```

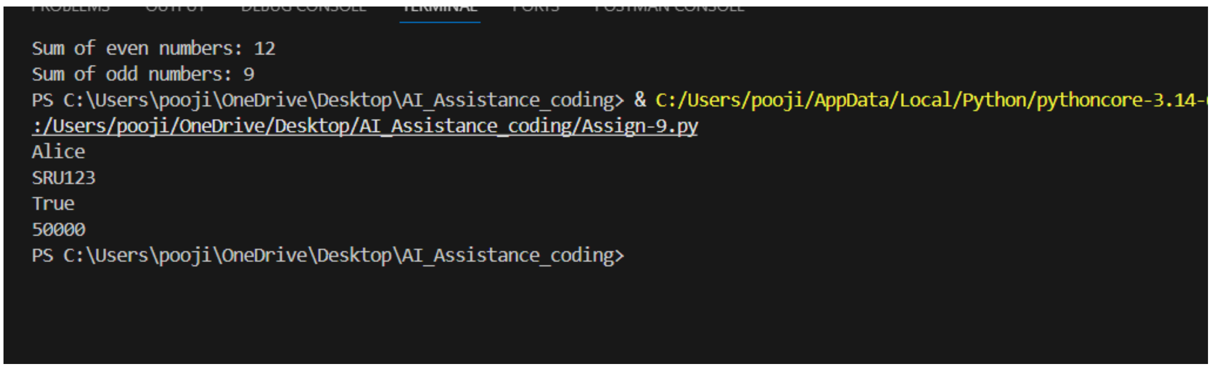
AI-Generated Inline Comments



Assign-9.py > ...

```python
class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name            # Assign name to object
        self.roll_no = roll_no      # Assign roll number
        self.hostel_status = hostel_status  # Assign hostel status

    def fee_update(self):
        if self.hostel_status:
            self.fee = 50000        # Set fee if hostel student
        else:
            self.fee = 30000        # Set fee if not hostel student

    def display_details(self):
        print(self.name)            # Print student name
        print(self.roll_no)         # Print roll number
        print(self.hostel_status) # Print hostel status
        print(self.fee)             # Print fee
# Example usage:
student1 = sru_student("Alice", "SRU123", True)
student1.fee_update()
student1.display_details()
student2 = sru_student("Bob", "SRU456", False)
```

Output:



```
Sum of even numbers: 12
Sum of odd numbers: 9
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-
:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/Assign-9.py
Alice
SRU123
True
50000
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

AI-generated comments are quick and syntactically correct.
However, they sometimes explain the obvious instead of intent.
Manual comments better describe why logic exists.
AI comments may miss high-level explanations.
AI works best as a support tool, not a replacement.

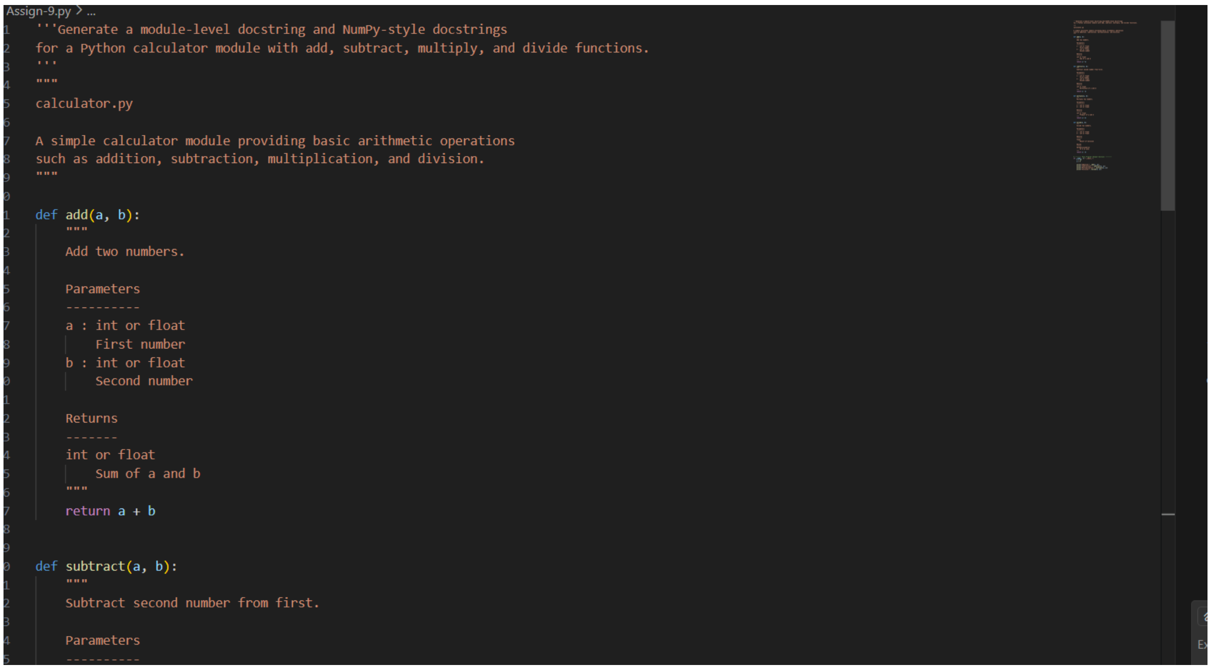
Task-03:

Prompt:

Generate a module-level docstring and NumPy-style docstrings

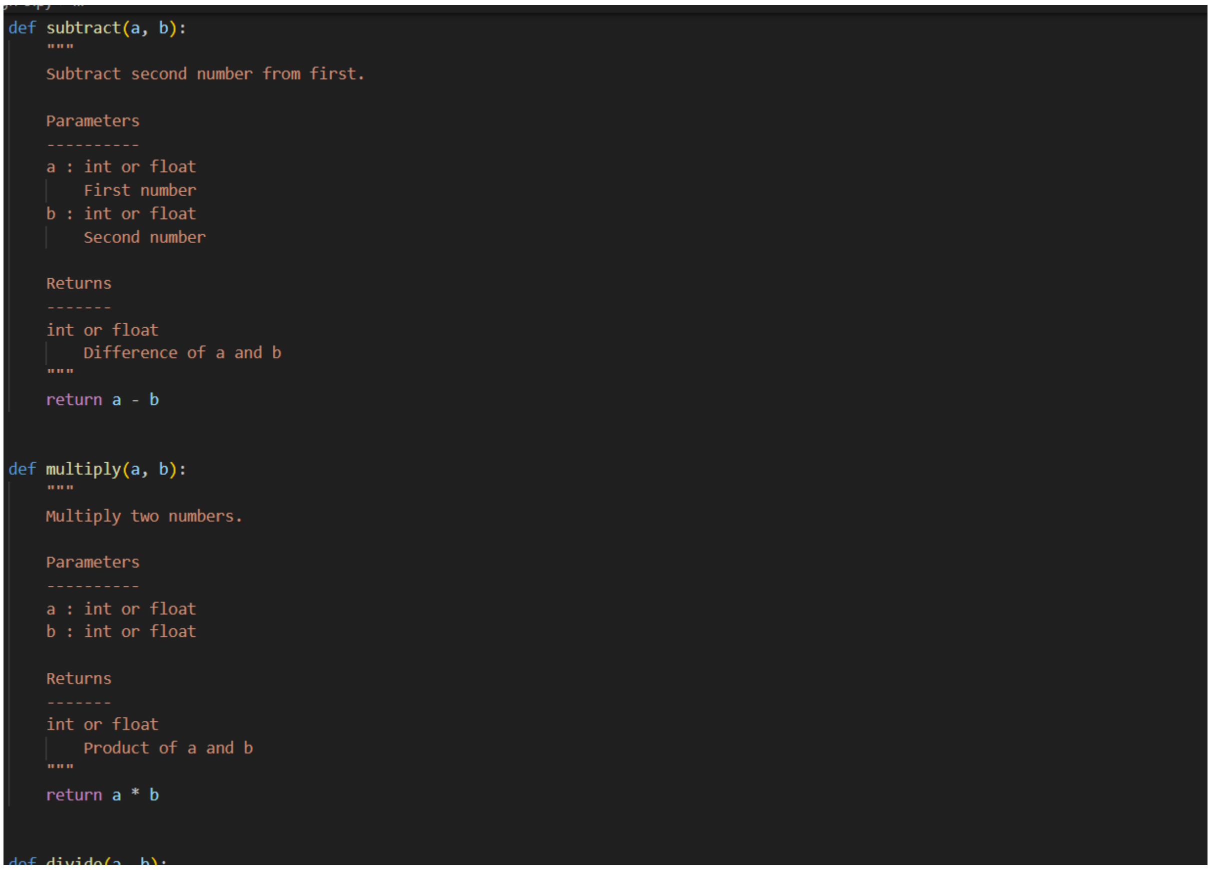
for a Python calculator module with add, subtract, multiply, and divide functions.

Code:

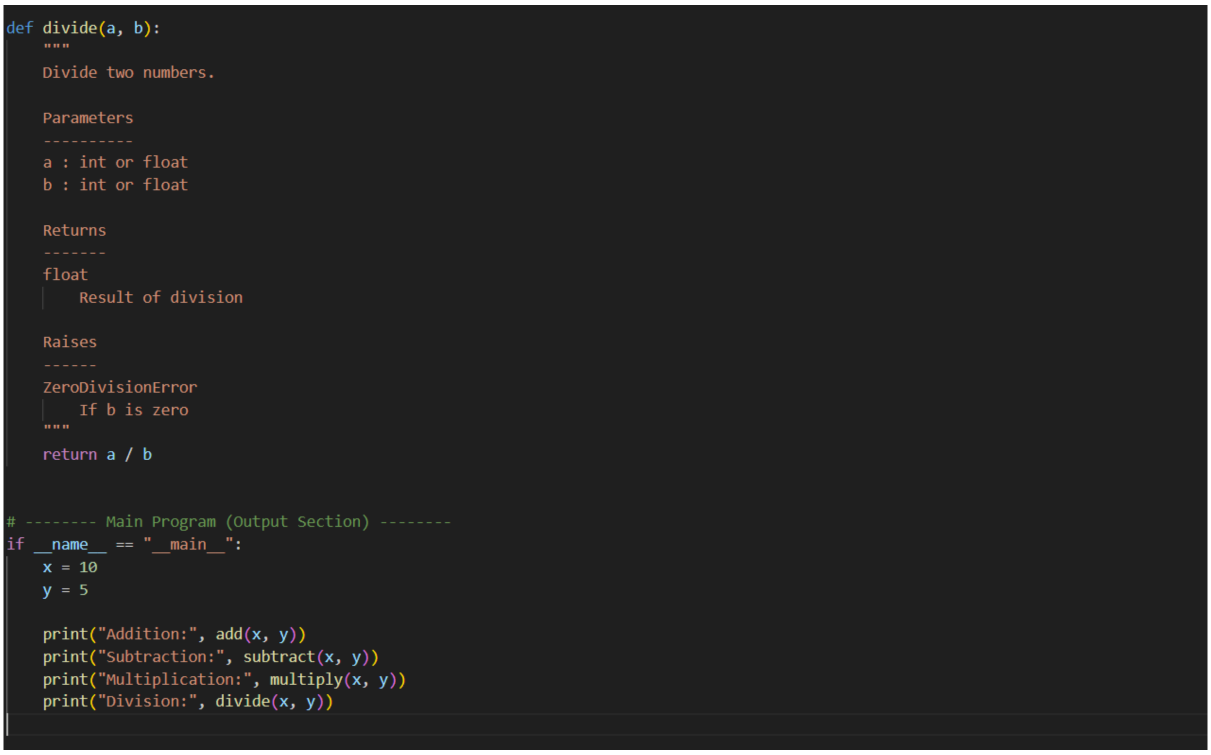


Assign-9.py > ...

```python
'''Generate a module-level docstring and NumPy-style docstrings
for a Python calculator module with add, subtract, multiply, and divide functions.
'''
"""
calculator.py

A simple calculator module providing basic arithmetic operations
such as addition, subtraction, multiplication, and division.
"""

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

    Parameters
    ----------
    a : int or float
        First number
    b : int or float
        Second number

    Returns
    -------
    int or float
        Sum of a and b
    """
    return a + b


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

    Parameters
    ----------
```

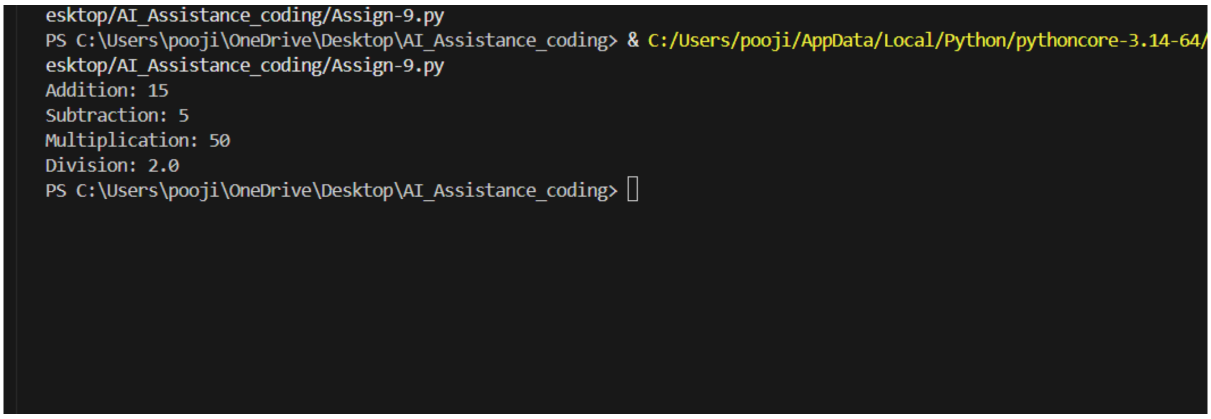


```python
def subtract(a, b):
    """
    Subtract second number from first.

    Parameters
    ----------
    a : int or float
        First number
    b : int or float
        Second number

    Returns
    -------
    int or float
        Difference of a and b
    """
    return a - b


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

    Parameters
    ----------
    a : int or float
    b : int or float

    Returns
    -------
    int or float
        Product of a and b
    """
    return a * b


def divide(a, b):
```



```
def divide(a, b):
    """
    Divide two numbers.

    Parameters
    ----------
    a : int or float
    b : int or float

    Returns
    -------
    float
        Result of division

    Raises
    ------
    ZeroDivisionError
        If b is zero
    """
    return a / b


# -------- Main Program (Output Section) --------
if __name__ == "__main__":
    x = 10
    y = 5

    print("Addition:", add(x, y))
    print("Subtraction:", subtract(x, y))
    print("Multiplication:", multiply(x, y))
    print("Division:", divide(x, y))
```

Output:



```
esktop/AI_Assistance_coding/Assign-9.py
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/
esktop/AI_Assistance_coding/Assign-9.py
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> 
```

Explanation:

Manual documentation is created by developers themselves and often includes rich, context-specific explanations that capture the reasoning behind design choices. While this type of documentation is usually more accurate and insightful, it demands significant time and continuous effort to keep it updated as the code evolves. In contrast, AI-generated documentation is produced automatically, offering speed and uniformity, which is especially useful for large or rapidly changing projects. However, such documentation can sometimes be too general and may fail to convey the deeper intent or logic behind the code.

Structured documentation for scripts containing multiple functions presents information in a well-organized manner by clearly outlining each function’s role, parameters, return values, and interactions. This organization makes it easier for readers to understand how different components operate together. As a result, code readability is improved, maintenance becomes simpler, and team collaboration is more effective.