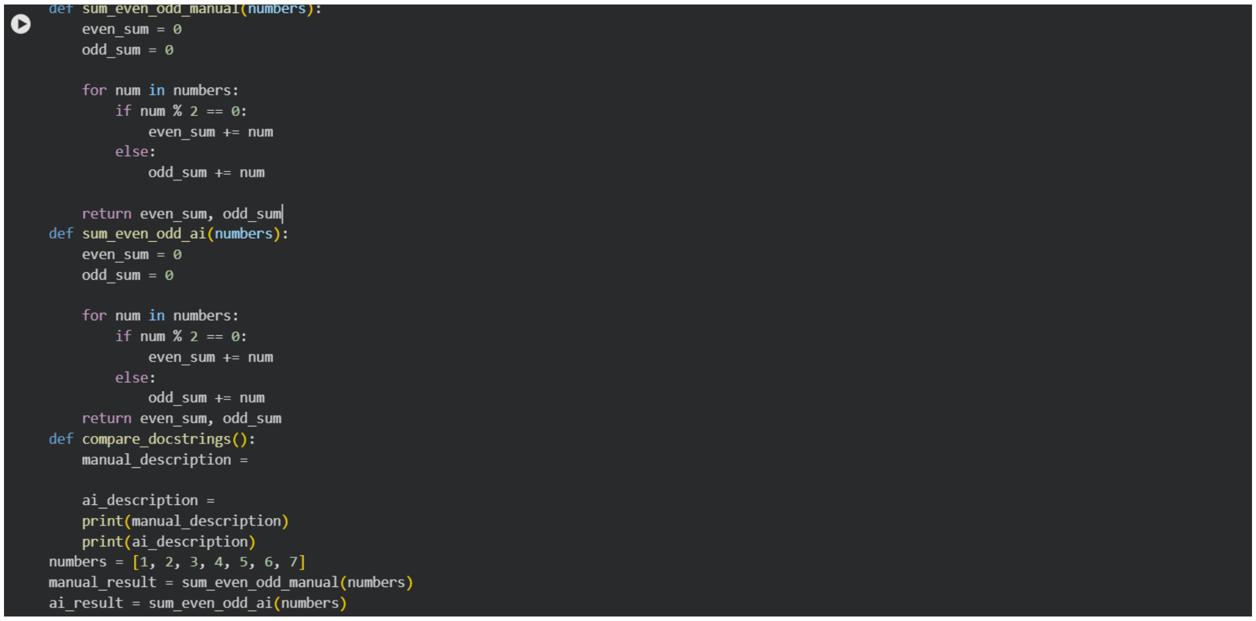
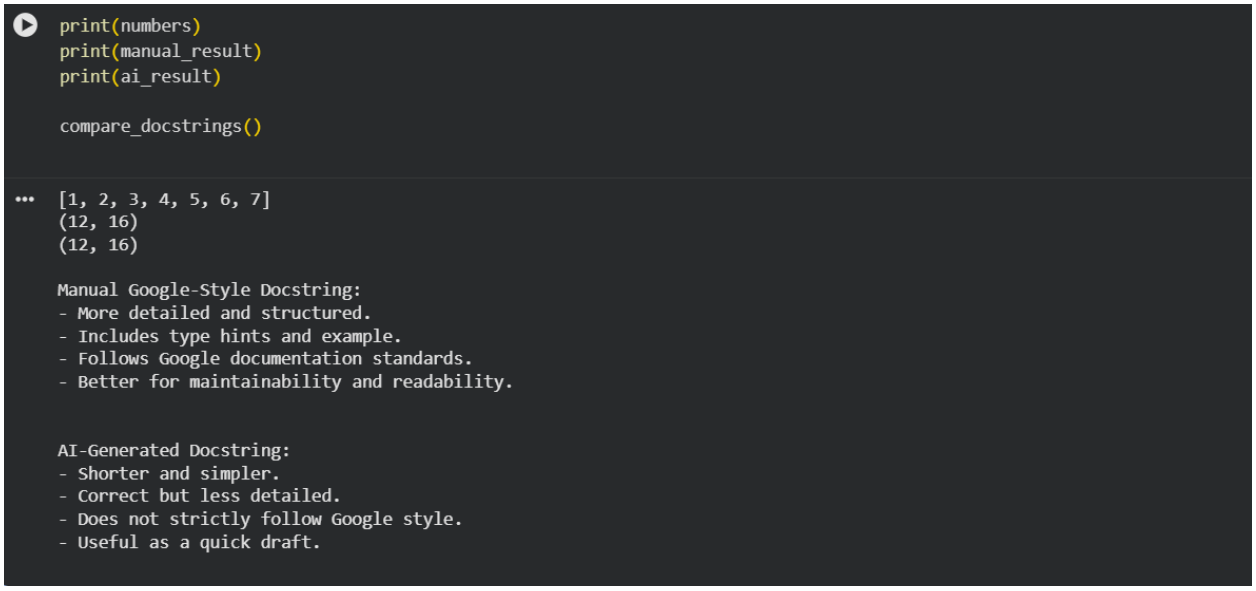
ASSIGNMENT 9.3

Task 1: Basic Docstring Generation
Scenario
You are developing a utility function that processes numerical lists and must be properly documented for future maintenance.



```python
def sum_even_odd_manual(numbers):
    even_sum = 0
    odd_sum = 0

    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num

    return even_sum, odd_sum
def sum_even_odd_ai(numbers):
    even_sum = 0
    odd_sum = 0

    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    return even_sum, odd_sum
def compare_docstrings():
    manual_description =

    ai_description =
    print(manual_description)
    print(ai_description)
numbers = [1, 2, 3, 4, 5, 6, 7]
manual_result = sum_even_odd_manual(numbers)
ai_result = sum_even_odd_ai(numbers)
```

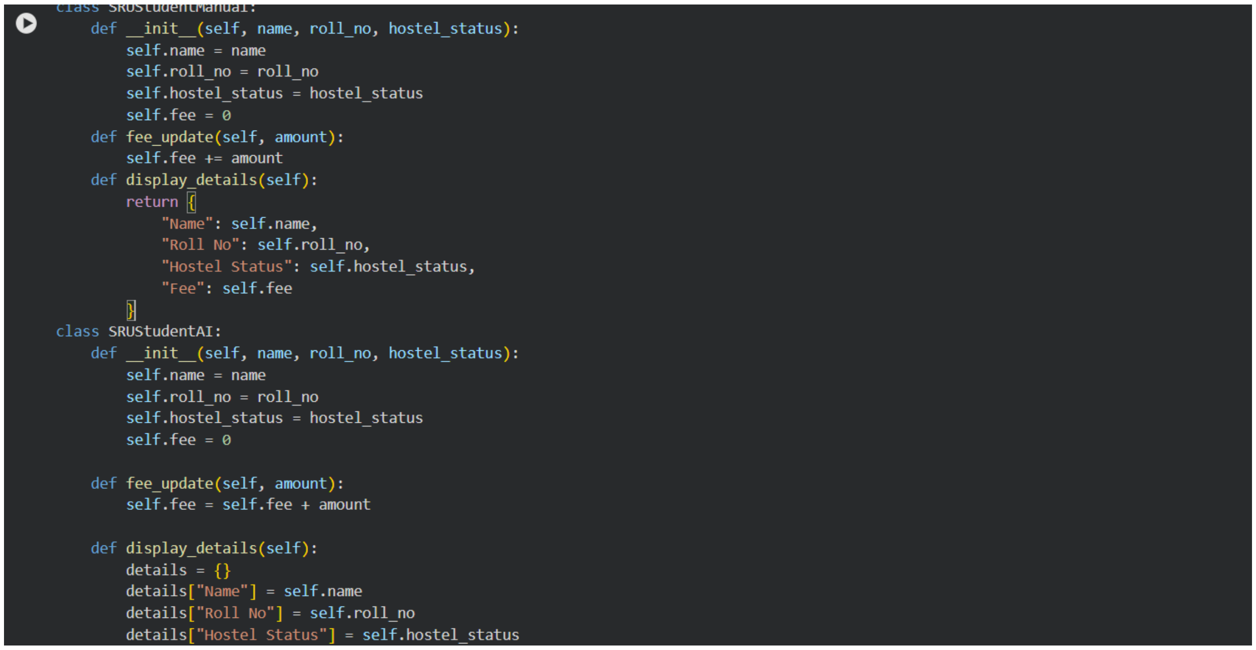


```python
print(numbers)
print(manual_result)
print(ai_result)

compare_docstrings()
```

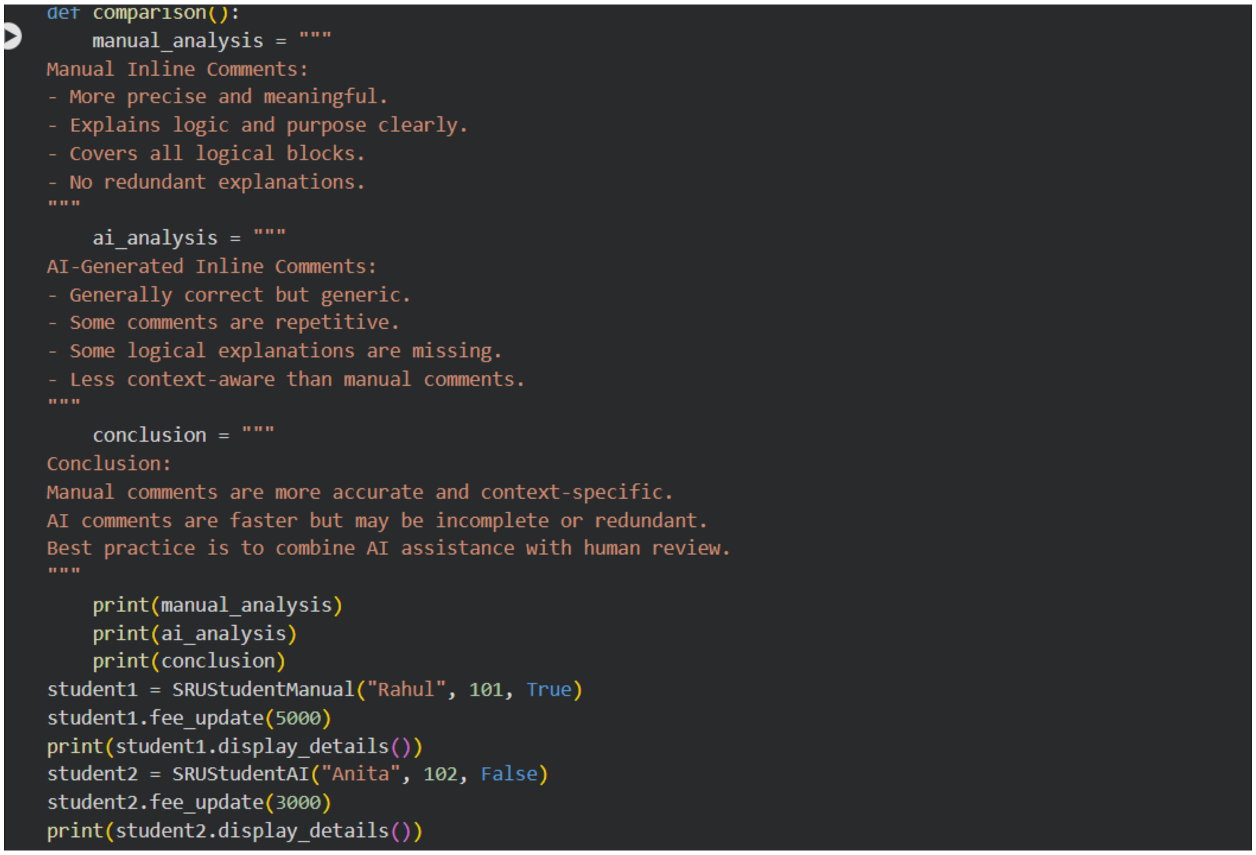
```
[1, 2, 3, 4, 5, 6, 7]
(12, 16)
(12, 16)

Manual Google-Style Docstring:
- More detailed and structured.
- Includes type hints and example.
- Follows Google documentation standards.
- Better for maintainability and readability.


AI-Generated Docstring:
- Shorter and simpler.
- Correct but less detailed.
- Does not strictly follow Google style.
- Useful as a quick draft.
```

Task 2: Automatic Inline Comments

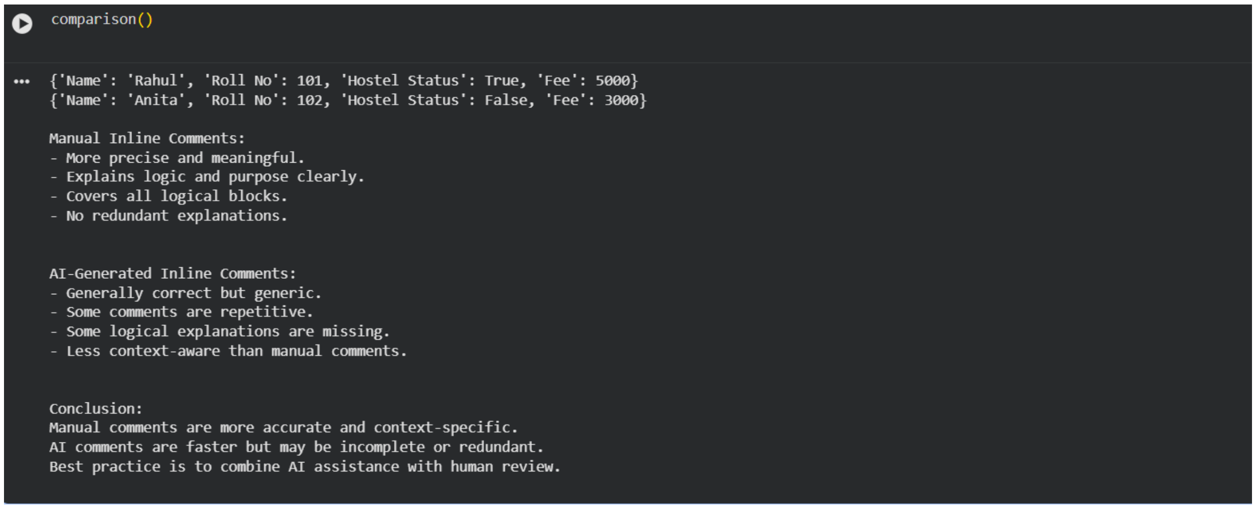
Scenario
You are developing a student management module that must be easy to understand for new
developers.



```python
class SRUStudentManual:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name
        self.roll_no = roll_no
        self.hostel_status = hostel_status
        self.fee = 0
    def fee_update(self, amount):
        self.fee += amount
    def display_details(self):
        return {
            "Name": self.name,
            "Roll No": self.roll_no,
            "Hostel Status": self.hostel_status,
            "Fee": self.fee
        }
class SRUStudentAI:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name
        self.roll_no = roll_no
        self.hostel_status = hostel_status
        self.fee = 0

    def fee_update(self, amount):
        self.fee = self.fee + amount

    def display_details(self):
        details = {}
        details["Name"] = self.name
        details["Roll No"] = self.roll_no
        details["Hostel Status"] = self.hostel_status
```



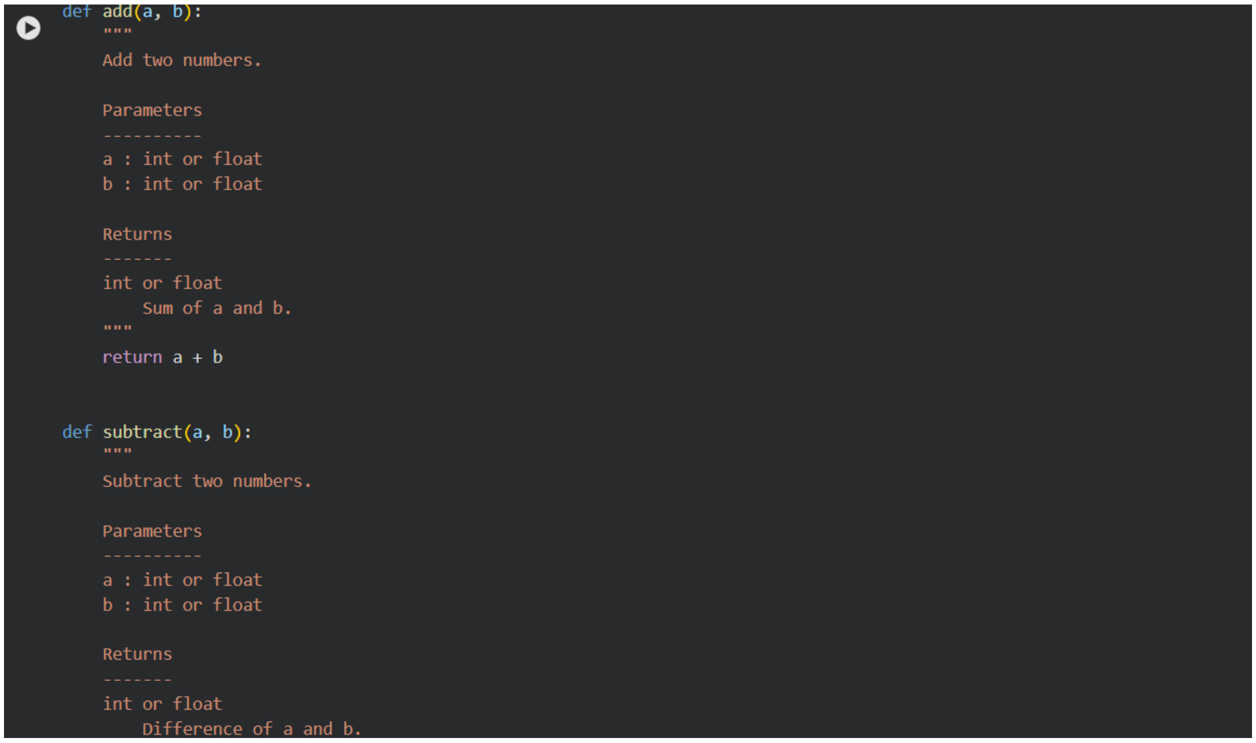
```python
def comparison():
    manual_analysis = """
Manual Inline Comments:
- More precise and meaningful.
- Explains logic and purpose clearly.
- Covers all logical blocks.
- No redundant explanations.
"""
    ai_analysis = """
AI-Generated Inline Comments:
- Generally correct but generic.
- Some comments are repetitive.
- Some logical explanations are missing.
- Less context-aware than manual comments.
"""
    conclusion = """
Conclusion:
Manual comments are more accurate and context-specific.
AI comments are faster but may be incomplete or redundant.
Best practice is to combine AI assistance with human review.
"""
    print(manual_analysis)
    print(ai_analysis)
    print(conclusion)
student1 = SRUStudentManual("Rahul", 101, True)
student1.fee_update(5000)
print(student1.display_details())
student2 = SRUStudentAI("Anita", 102, False)
student2.fee_update(3000)
print(student2.display_details())
```



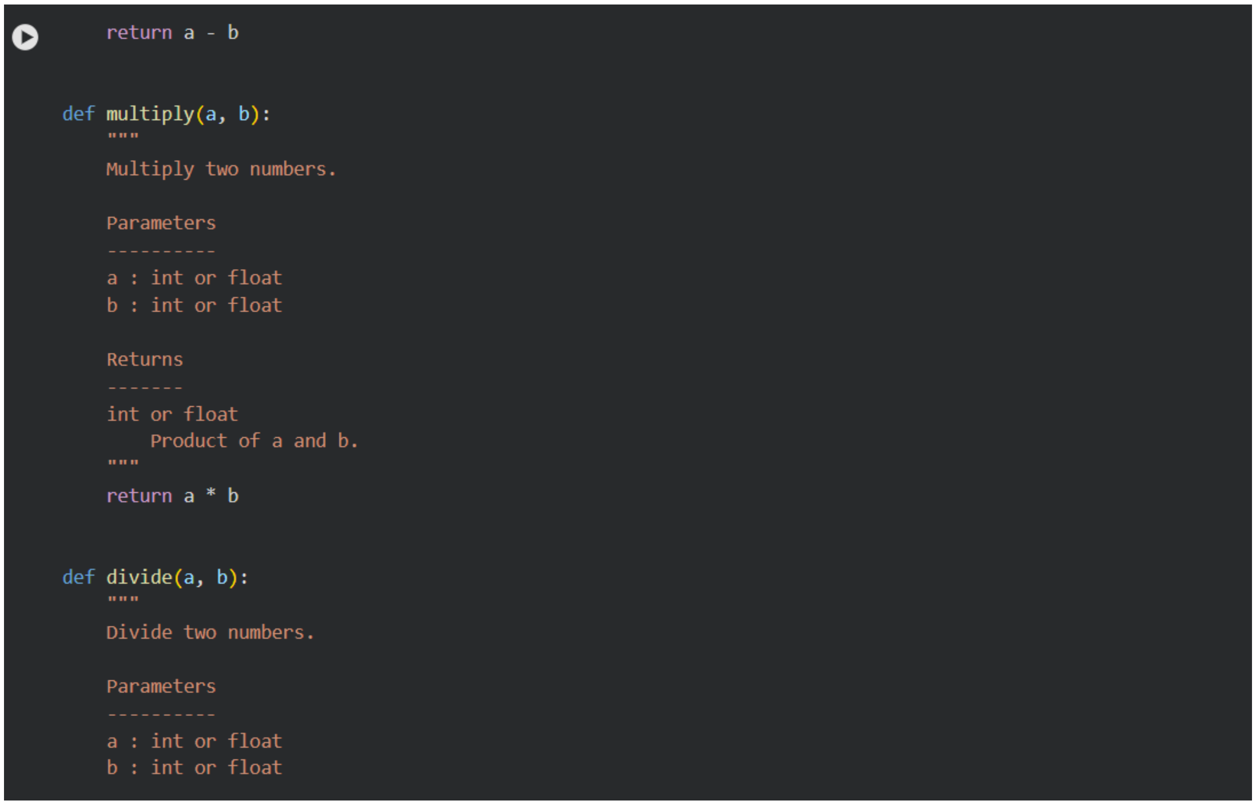
```
comparison()
```

```
{'Name': 'Rahul', 'Roll No': 101, 'Hostel Status': True, 'Fee': 5000}
{'Name': 'Anita', 'Roll No': 102, 'Hostel Status': False, 'Fee': 3000}

Manual Inline Comments:
- More precise and meaningful.
- Explains logic and purpose clearly.
- Covers all logical blocks.
- No redundant explanations.


AI-Generated Inline Comments:
- Generally correct but generic.
- Some comments are repetitive.
- Some logical explanations are missing.
- Less context-aware than manual comments.


Conclusion:
Manual comments are more accurate and context-specific.
AI comments are faster but may be incomplete or redundant.
Best practice is to combine AI assistance with human review.
```

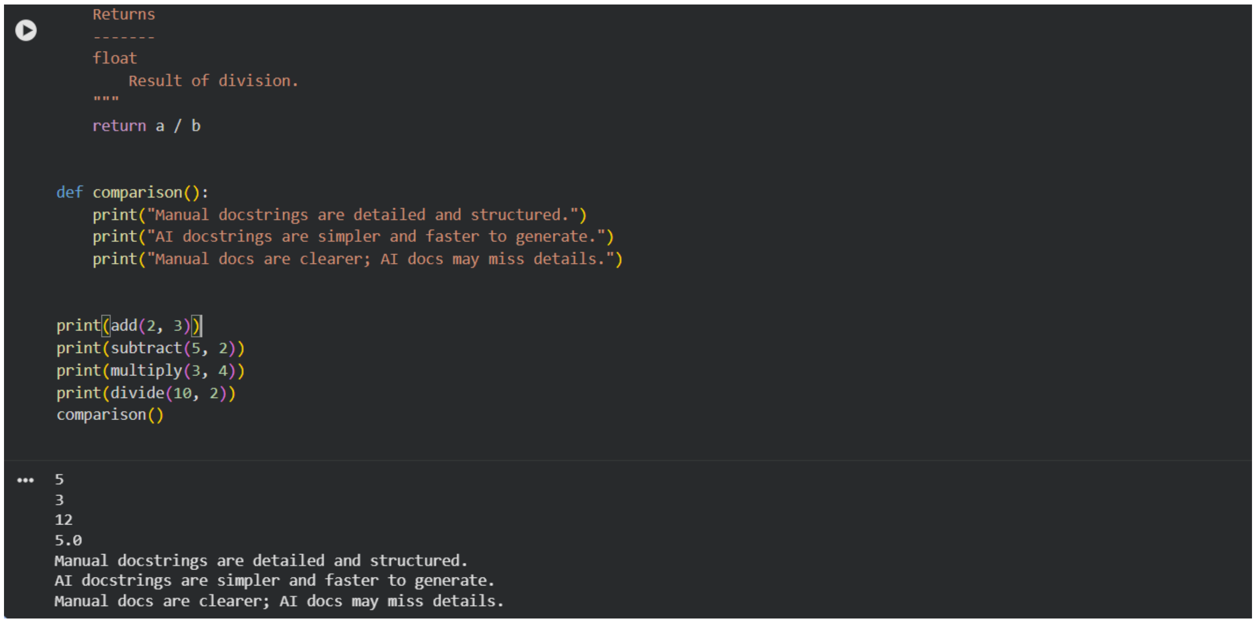
Task 3: Module-Level and Function-Level Documentation
Scenario
You are building a small calculator module that will be shared across multiple projects and
requires structured documentation.



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

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

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


def subtract(a, b):
    """
    Subtract two numbers.

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

    Returns
    -------
    int or float
        Difference of a and b.
```



```python
    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):
    """
    Divide two numbers.

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



```python
    Returns
    -------
    float
        Result of division.
    """
    return a / b


def comparison():
    print("Manual docstrings are detailed and structured.")
    print("AI docstrings are simpler and faster to generate.")
    print("Manual docs are clearer; AI docs may miss details.")


print(add(2, 3))
print(subtract(5, 2))
print(multiply(3, 4))
print(divide(10, 2))
comparison()
```

```
5
3
12
5.0
Manual docstrings are detailed and structured.
AI docstrings are simpler and faster to generate.
Manual docs are clearer; AI docs may miss details.
```