

## Assignment 9.3(Ai Assisted Coding)

Htno:2303a51288

Btno:05

### Task 1:

#### Basic Docstring Generation

#### Scenario

You are developing a utility function that processes numerical lists and must be properly documented for future maintenance.

#### Requirements

- Write a Python function to return the sum of even numbers and sum of odd numbers in a given list
- Manually add a Google Style docstring to the function
- Use an AI-assisted tool (Copilot / Cursor AI) to generate a function-level docstring
- Compare the AI-generated docstring with the manually written docstring
- Analyze clarity, correctness, and completeness

#### Expected Output

- Python function with manual Google-style docstring
- AI-generated docstring for the same function
- Comparison explaining differences between manual and AI-generated documentation
- Improved understanding of AI-generated function-level documentation.

Code:

The image displays two screenshots of the Visual Studio Code (VS Code) editor interface, showing the development of a Python function named `sum_even_odd` in a file named `docstring.py`.

**Top Screenshot:** The editor shows the initial code with a docstring and example usage. The docstring describes the function's purpose and arguments. The example usage defines a list of numbers and calls the function.

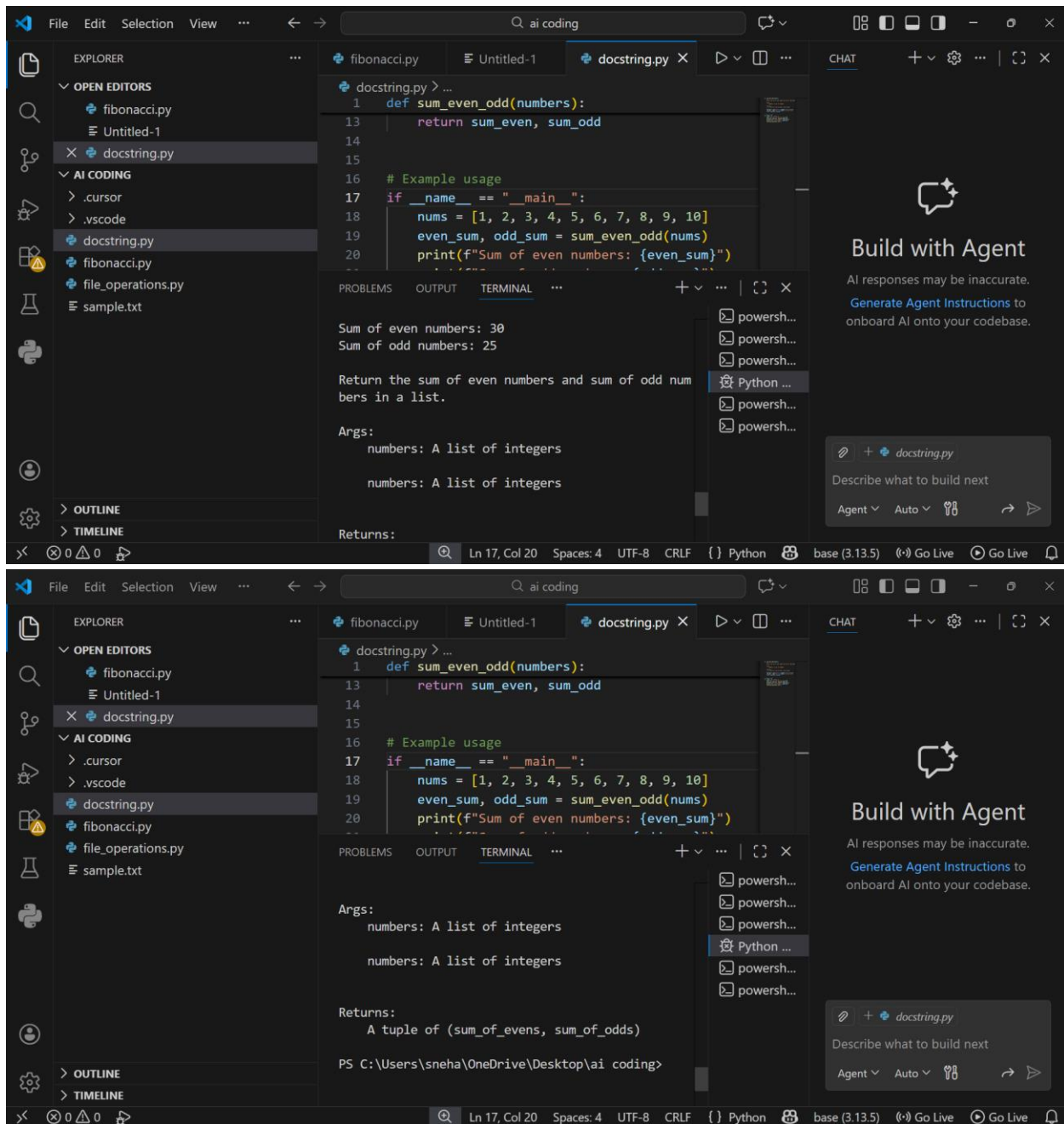
```
1 def sum_even_odd(numbers):
2     """
3     Return the sum of even numbers and sum of odd numbers.
4
5     Args:
6         numbers: A list of integers
7
8     Returns:
9         A tuple of (sum_of_evens, sum_of_odds)
10    """
11    sum_even = sum(n for n in numbers if n % 2 == 0)
12    sum_odd = sum(n for n in numbers if n % 2 != 0)
13    return sum_even, sum_odd
14
15 # Example usage
16 if __name__ == "__main__":
17     nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

**Bottom Screenshot:** The editor shows the code after adding print statements to verify the function's output. The example usage section is updated to call the function and print the results.

```
13     return sum_even, sum_odd
14
15 # Example usage
16 if __name__ == "__main__":
17     nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
18     even_sum, odd_sum = sum_even_odd(nums)
19     print(f"Sum of even numbers: {even_sum}")
20     print(f"Sum of odd numbers: {odd_sum}")
21     print(sum_even_odd.__doc__)
```

The terminal output shows the result of the function call: `A tuple of (sum_of_evens, sum_of_odds)`.

Output:



## Explanation:

-->The manual docstring is more detailed and follows proper Google style. It clearly explains the input type, return values, and includes an example, making it easier to understand.

-->The AI-generated docstring is correct but shorter. It provides basic information but lacks detailed explanation and an example.

**Conclusion:** AI saves time, but manual improvement makes documentation clearer and more complete.

## Task 2: Automatic Inline Comments

### Scenario

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

developers.

### Requirements

- Write a Python program for an `sru_student` class with the following:
  - Attributes: `name`, `roll_no`, `hostel_status`
  - Methods: `fee_update()` and `display_details()`
- Manually write inline comments for each line or logical block
- Use an AI-assisted tool to automatically add inline comments
- Compare manual comments with AI-generated comments
- Identify missing, redundant, or incorrect AI comments

### Expected Output

- Python class with manually written inline comments
- AI-generated inline comments added to the same code
- Comparative analysis of manual vs AI comments
- Critical discussion on strengths and limitations of AI-generated comments

Code:

The image displays two screenshots of a Google Colab notebook interface. The top screenshot shows a Python class named `sru_student` with methods `__init__`, `fee_update`, and `display_details`. The bottom screenshot shows the same code with AI-generated inline comments added to each line.

**Top Screenshot: Original Code**

```
[10] class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name          # Store student's name
        self.roll_no = roll_no    # Store student's roll number
        self.hostel_status = hostel_status # Store hostel status (True/False)
        self.fee = 50000          # Base academic fee

    def fee_update(self):
        if self.hostel_status:    # If student stays in hostel
            self.fee += 20000     # Add hostel fee
        else:                     # If day scholar
            self.fee += 0         # No extra hostel fee
        return self.fee          # Return total fee

    def display_details(self):
        print("Name:", self.name) # Print student name
        print("Roll No:", self.roll_no) # Print roll number
        print("Hostel Status:", self.hostel_status) # Print hostel info
        print("Total Fee:", self.fee) # Print final fee
```

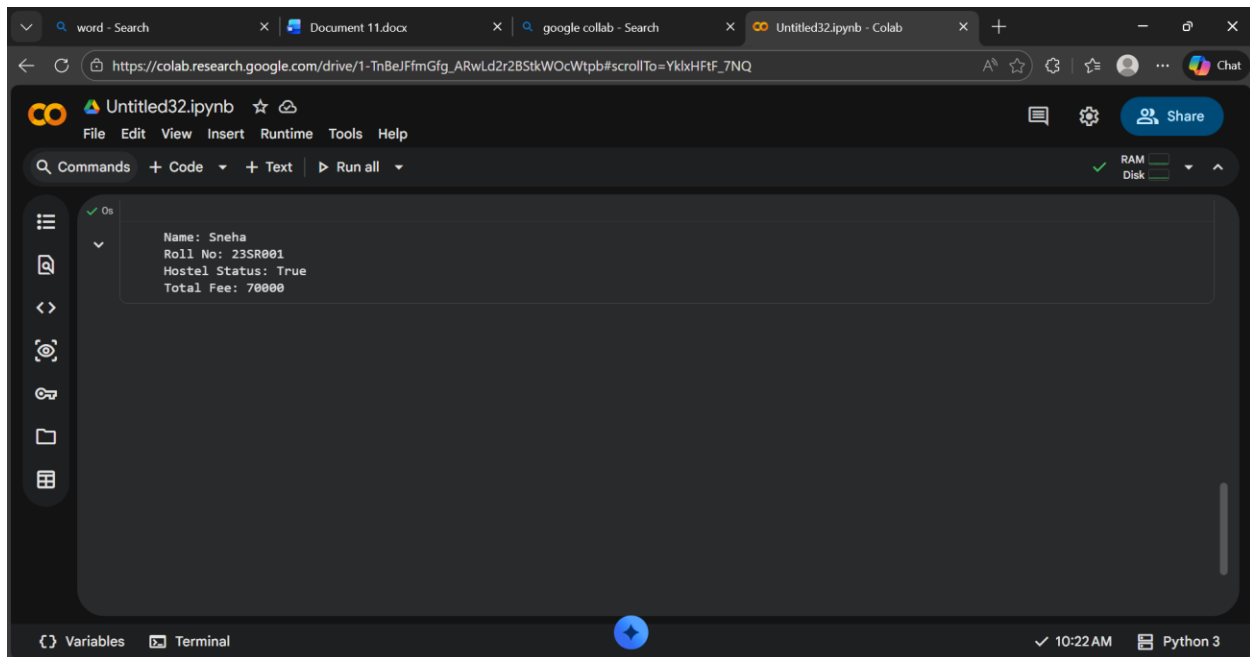
**Bottom Screenshot: AI-Generated Comments**

```
[12] class sru_student:
    def __init__(self, name, roll_no, hostel_status):
        self.name = name # Student name
        self.roll_no = roll_no # Roll number
        self.hostel_status = hostel_status # Hostel status
        self.fee = 50000 # Initial fee

    def fee_update(self):
        if self.hostel_status: # Check hostel
            self.fee += 20000 # Increase fee
        else: # Otherwise
            self.fee += 0 # No change
        return self.fee # Return fee

    def display_details(self):
        print("Name:", self.name) # Print name
        print("Roll No:", self.roll_no) # Print roll
        print("Hostel Status:", self.hostel_status) # Print hostel
```

Output:



Explanation:

## Comparison: Manual vs AI Comments

Feature	Manual Comments	AI Comments	Analysis
Detail Level	More descriptive	Short/basic	AI lacks detailed explanation
Clarity	Explains purpose clearly	Basic meaning only	Manual helps beginners more
Redundancy	Minimal	Some redundant (e.g., "Print name")	AI often states obvious
Logic Explanation	Explains fee logic clearly	Just "increase fee"	AI misses context
Completeness	Covers all important logic	Covers lines but shallow	AI comments are surface-level

## 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.

#### Requirements

- Write a Python script containing 3–4 functions (e.g., add, subtract, multiply, divide)
- Manually write NumPy Style docstrings for each function
- Use AI assistance to generate:
  - A module-level docstring
  - Individual function-level docstrings
- Compare AI-generated docstrings with manually written ones
- Evaluate documentation structure, accuracy, and readability

#### Expected Output

- Python script with manual NumPy-style docstrings
- AI-generated module-level and function-level documentation
- Comparison between AI-generated and manual documentation
- Clear understanding of structured documentation for multi-function scripts.

#### Code:

word - Search x 2303a51305-Assignment 9.3(a) x google colab - Search x Untitled32.ipynb - Colab x + -

https://colab.research.google.com/drive/1-TnBeJFmGfg\_ARwLd2r2BStkWOcWtpb#scrollTo=le\_NM0FFD3lt

Untitled32.ipynb ☆ ☁

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

RAM Disk

[16] ✓ 0s

```
"""
Calculator module providing basic arithmetic operations.

This module includes functions for addition, subtraction,
multiplication, and division. It can be reused in different
projects requiring simple mathematical operations.
"""

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

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

    Returns
    -----
    """
```

Variables Terminal

✓ 10:40 AM Python 3

word - Search x 2303a51305-Assignment 9.3(a) x google colab - Search x Untitled32.ipynb - Colab x + -

https://colab.research.google.com/drive/1-TnBeJFmGfg\_ARwLd2r2BStkWOcWtpb#scrollTo=le\_NM0FFD3lt

Untitled32.ipynb ☆ ☁

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

RAM Disk

[16] ✓ 0s

```
Sum of a and b.
"""
return a + b

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

    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
```

Variables Terminal

✓ 10:40 AM Python 3

Snipping Tool

Screenshot copied to clipboard  
Automatically saved to screenshots folder.

Markup and share

word - Search x 2303a51305-Assignment 9.3(a) x google colab - Search x Untitled32.ipynb - Colab x + -

https://colab.research.google.com/drive/1-TnBeJFmGfg\_ARwLd2r2BStkWOcWtpb#scrollTo=le\_NM0FFD3lt

Untitled32.ipynb ☆ Share

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

RAM Disk

```
[16] ✓ 0s def multiply(a, b):  
    """  
    Multiply two numbers.  
  
    Parameters  
    -----  
    a : int or float  
        First number.  
    b : int or float  
        Second number.  
  
    Returns  
    -----  
    int or float  
        Product of a and b.  
    """  
    return a * b  
  
def divide(a, b):
```

Variables Terminal

✓ 10:40 AM

Snipping Tool  
Screenshot copied to  
Automatically saved

word - Search x 2303a51305-Assignment 9.3(a) x google colab - Search x Untitled32.ipynb - Colab x + -

https://colab.research.google.com/drive/1-TnBeJFmGfg\_ARwLd2r2BStkWOcWtpb#scrollTo=le\_NM0FFD3lt

Untitled32.ipynb ☆ Share

File Edit View Insert Runtime Tools Help

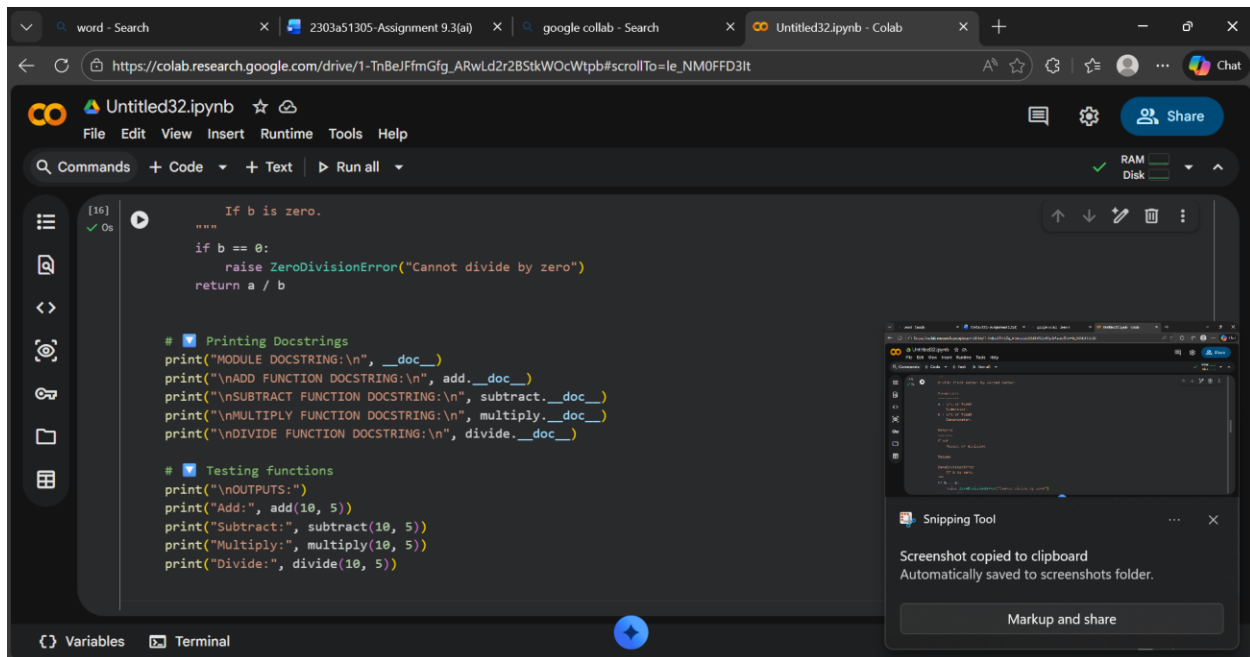
Commands + Code + Text ▶ Run all

RAM Disk

```
[16] ✓ 0s """  
Divide first number by second number.  
  
Parameters  
-----  
a : int or float  
    Numerator.  
b : int or float  
    Denominator.  
  
Returns  
-----  
float  
    Result of division.  
  
Raises  
-----  
ZeroDivisionError  
    If b is zero.  
"""  
if b == 0:  
    raise ZeroDivisionError("Cannot divide by zero")
```

Variables Terminal

✓ 10:40 AM Python 3



Output:

The screenshot shows a Google Colab notebook titled 'Untitled32.ipynb'. The code cell contains the following Python code:

```
[16] print("\nOUTPUTS:")
      print("Add:", add(10, 5))
      print("Subtract:", subtract(10, 5))
      print("Multiply:", multiply(10, 5))
      print("Divide:", divide(10, 5))
```

The output of the code is displayed below the cell:

```
... MODULE DOCSTRING:

Calculator module providing basic arithmetic operations.

This module includes functions for addition, subtraction,
multiplication, and division. It can be reused in different
projects requiring simple mathematical operations.

ADD FUNCTION DOCSTRING:

Add two numbers.

Parameters
-----
a : int or float
    First number.
```

The interface includes a left sidebar with navigation icons, a top menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help', and a bottom status bar showing '10:40 AM' and 'Python 3'.

The screenshot shows the same Google Colab notebook, but the code cell now contains docstrings for the 'Add' and 'Subtract' functions:

```
... Add two numbers.

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

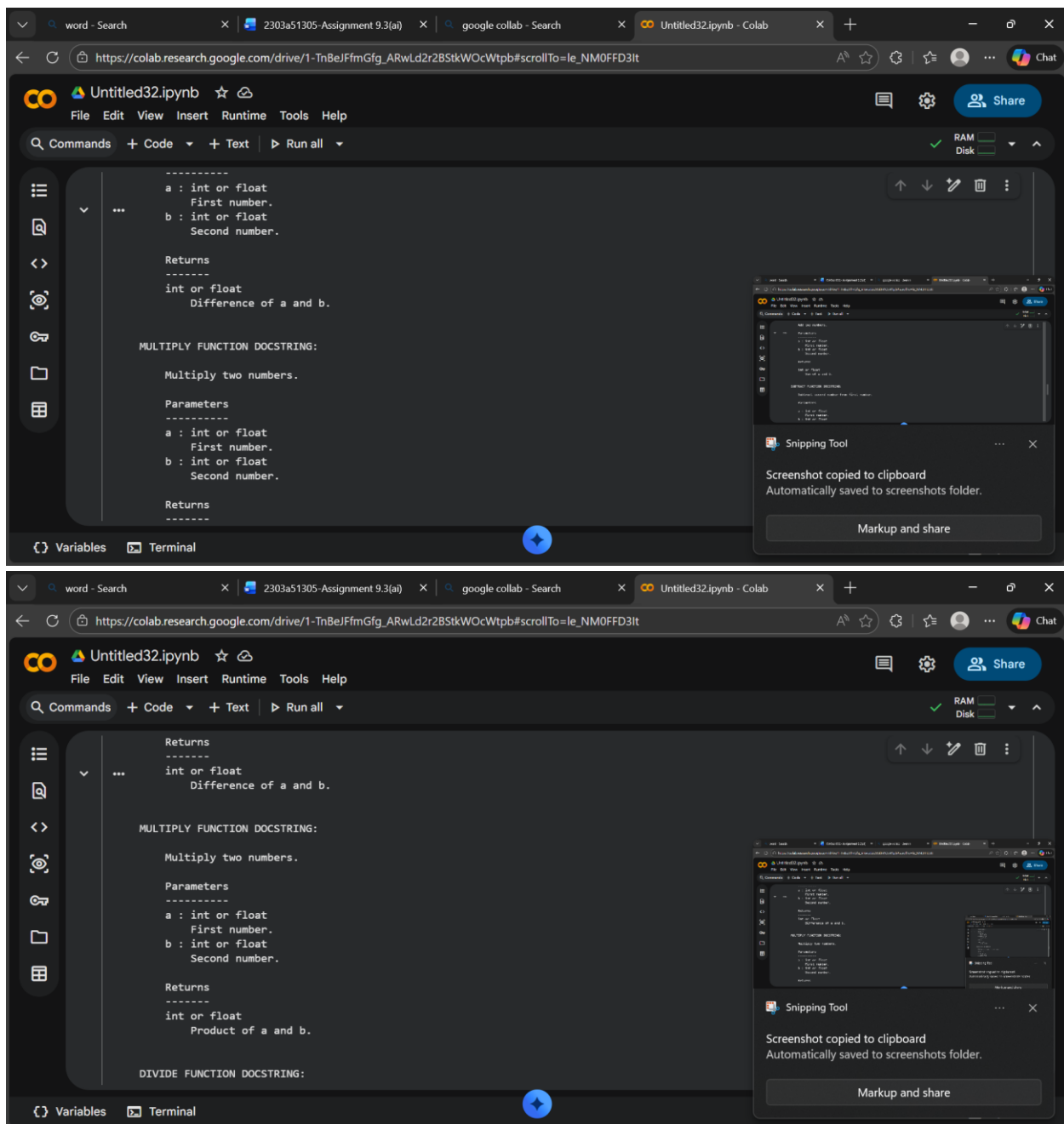
Returns
-----
int or float
    Sum of a and b.

SUBTRACT FUNCTION DOCSTRING:

Subtract second number from first number.

Parameters
-----
a : int or float
    First number.
b : int or float
```

The interface elements are consistent with the previous screenshot, including the sidebar, menu bar, and status bar.



Explanation:

Aspect	Manual NumPy Docstrings	AI-Generated Docstrings	Analysis
Format	Proper NumPy style	Generic style	Manual follows structured standard
Detail	Clear sections	Short descriptions	AI is brief
Error Handling	Mentions ZeroDivisionError	Often missing	Manual more complete

Readability

Highly structured

Simple and readable

AI easier but less formal

Conclusion:

AI is helpful for generating initial documentation, but manual refinement is required for professional-quality module and function documentation.

