LAB 9 – Documentation Generation

Course: AI Assisted Coding

Week 5

Name: G. Karthikeya

BATCH-14

2303A510D2

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

Scenario

You have joined a development team where several utility functions are already implemented, but the code lacks proper documentation. New team members are struggling to understand how these functions should be used.

Task Description

You are given a Python script containing multiple functions without any docstrings.

Using an AI-assisted coding tool:

• Ask the AI to automatically generate Google-style function docstrings for each function

• Each docstring should include:

o A brief description of the function

o Parameters with data types

o Return values

o At least one example usage (if applicable)

Experiment with different prompting styles (zero-shot or context-based)

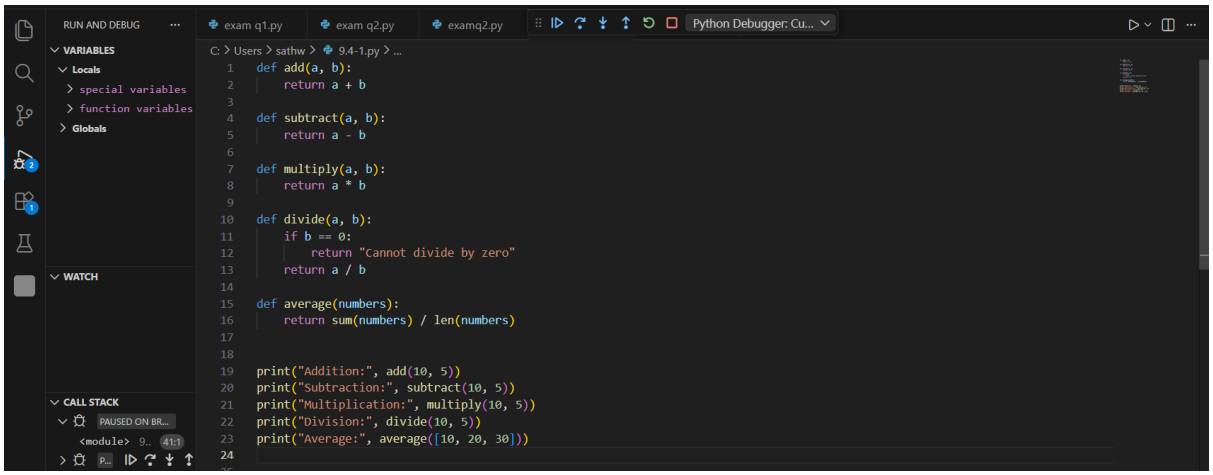
to observe quality differences.

Expected Outcome

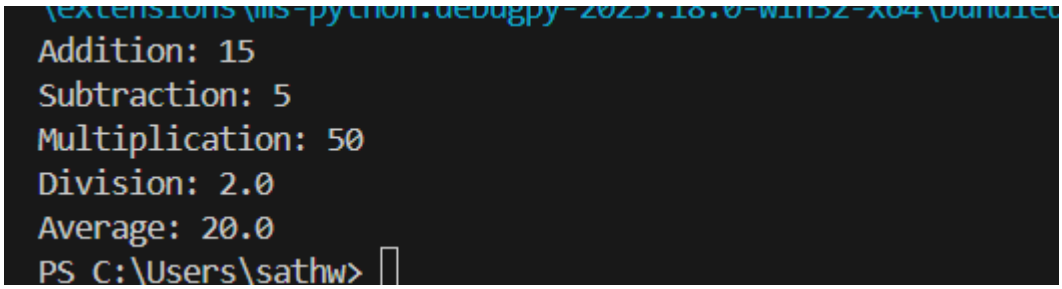
• A Python script with well-structured Google-style docstrings

• Docstrings that clearly explain function behavior and usage

• Improved readability and usability of the codebase

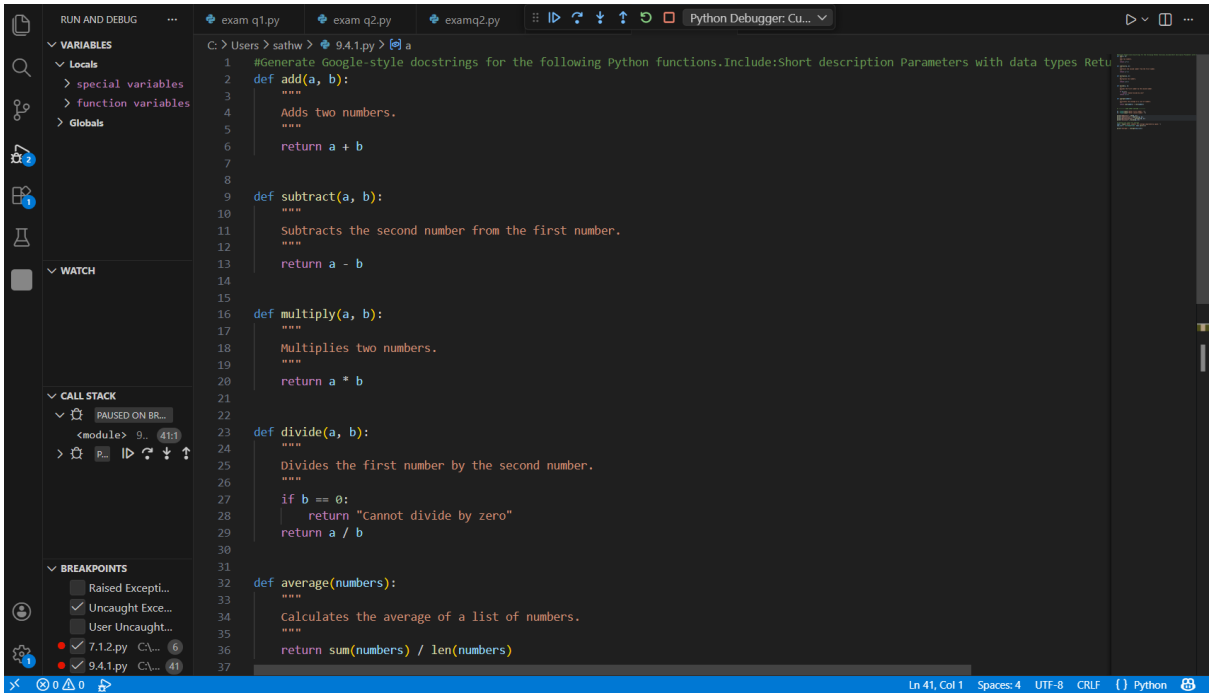


#OUTPUT:



#prompt:

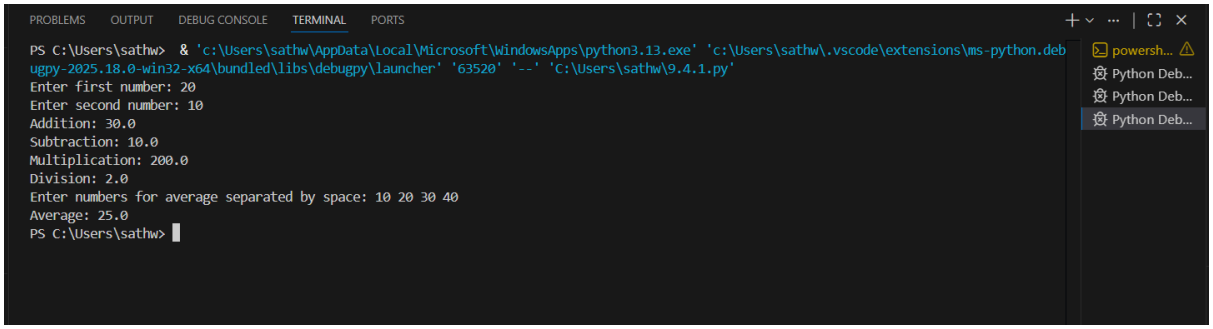
Generate Google-style docstrings for the following Python functions. Include: Short description Parameters with data types Return type Example usage.



```python
#Generate Google-style docstrings for the following Python functions.Include:Short description Parameters with data types Retu
def add(a, b):
    """
    Adds two numbers.
    """
    return a + b


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


def multiply(a, b):
    """
    Multiplies two numbers.
    """
    return a * b


def divide(a, b):
    """
    Divides the first number by the second number.
    """
    if b == 0:
        return "Cannot divide by zero"
    return a / b


def average(numbers):
    """
    Calculates the average of a list of numbers.
    """
    return sum(numbers) / len(numbers)
```

```python
# --------- USER INPUT SECTION ---------

a = float(input("Enter first number: "))
b = float(input("Enter second number: "))

print("Addition:", add(a, b))
print("Subtraction:", subtract(a, b))
print("Multiplication:", multiply(a, b))
print("Division:", divide(a, b))

# Taking list input for average
nums = input("Enter numbers for average separated by space: ")
num_list = list(map(float, nums.split()))

print("Average:", average(num_list))
```

#Output



```
PS C:\Users\sathw> & 'c:\Users\sathw\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\sathw\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63520' '--' 'C:\Users\sathw\9.4.1.py'
Enter first number: 20
Enter second number: 10
Addition: 30.0
Subtraction: 10.0
Multiplication: 200.0
Division: 2.0
Enter numbers for average separated by space: 10 20 30 40
Average: 25.0
PS C:\Users\sathw>
```

EXPLANATION:

In this task, we created mathematical functions like addition, subtraction, multiplication, division, and average.

Initially, the functions had no documentation.
We used an AI tool to automatically generate **Google-style docstrings**.

The docstrings include:

- Function description
- Parameters
- Return values
- Example usage

This improves code readability and makes the program easier for other developers to understand and maintain.

Task 2: Enhancing Readability Through AI-Generated Inline

Comments

Scenario

A Python program contains complex logic that works correctly but is

difficult to understand at first glance. Future maintainers may find it hard

to debug or extend this code.

Task Description

You are provided with a Python script containing:

• Loops

• Conditional logic

• Algorithms (such as Fibonacci sequence, sorting, or searching)

Use AI assistance to:

• Automatically insert inline comments only for complex or non-

obvious logic

• Avoid commenting on trivial or self-explanatory syntax

The goal is to improve clarity without cluttering the code.

Expected Outcome

• A Python script with concise, meaningful inline comments

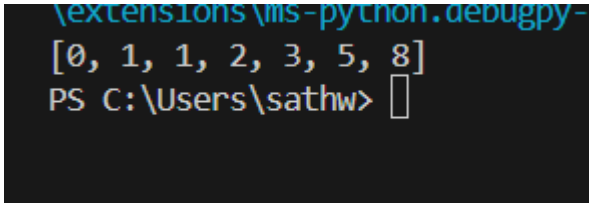
• Comments that explain why the logic exists, not what Python

syntax does

• Noticeable improvement in code readability

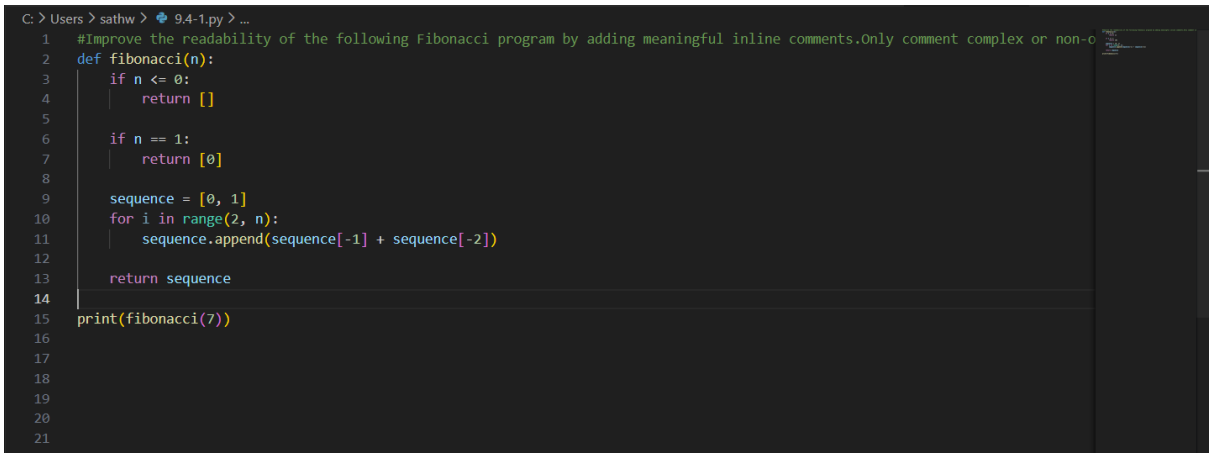
Original Code (Without Comments)

#OUTPUT:

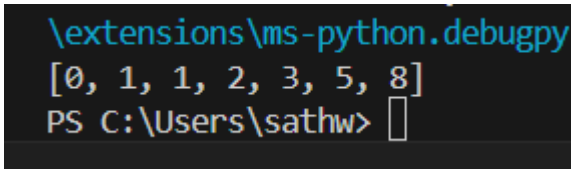


AI-Generated Inline Comments (Improved Version)

#prompt: Improve the readability of the following Fibonacci program by adding meaningful inline comments.Only comment complex or non-obvious logic. Do not explain basic Python syntax.



#OUTPUT:



EXPLANATION

In this task, AI was used to add meaningful inline comments to a Python program containing loops and an algorithm (Fibonacci).

Only complex logic was commented, and basic syntax was not explained.

This improved code readability and made the program easier to understand and maintain.

Task 3: Generating Module-Level Documentation for a Python

Package

Scenario

Your team is preparing a Python module to be shared internally (or

uploaded to a repository). Anyone opening the file should immediately

understand its purpose and structure.

Task Description

Provide a complete Python module to an AI tool and instruct it to

automatically generate a module-level docstring at the top of the file

that includes:

• The purpose of the module

• Required libraries or dependencies

• A brief description of key functions and classes

• A short example of how the module can be used

Focus on clarity and professional tone.

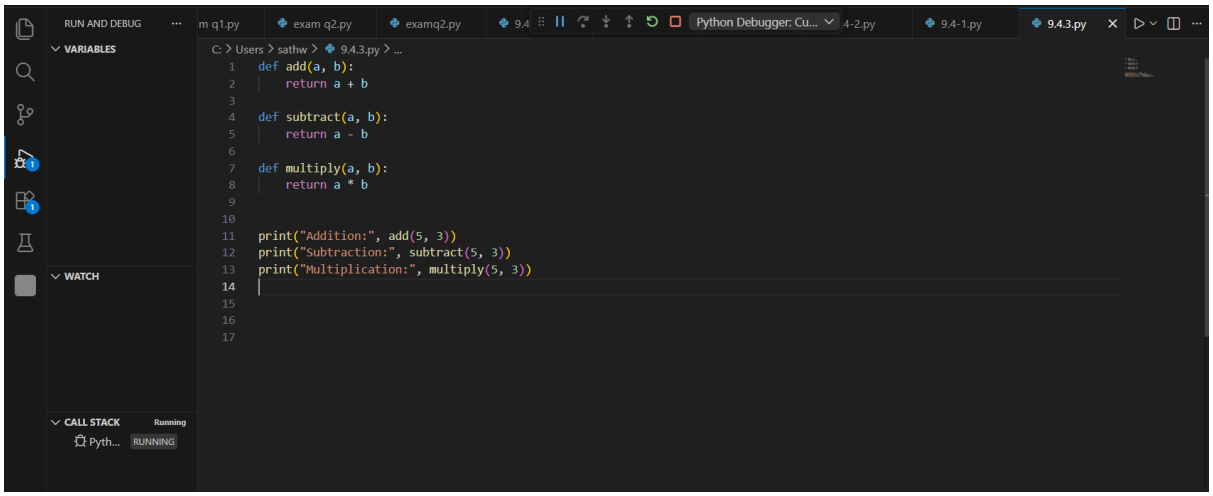
Expected Outcome

• A well-written multi-line module-level docstring

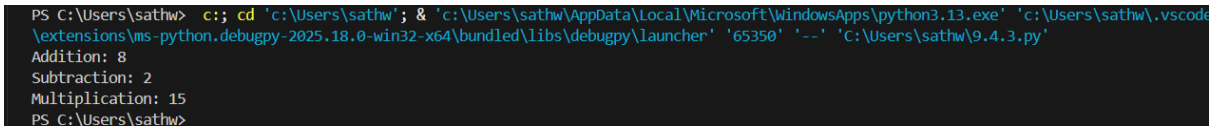
• Clear overview of what the module does and how to use it

• Documentation suitable for real-world projects or repositories

#CODE AND INPUT



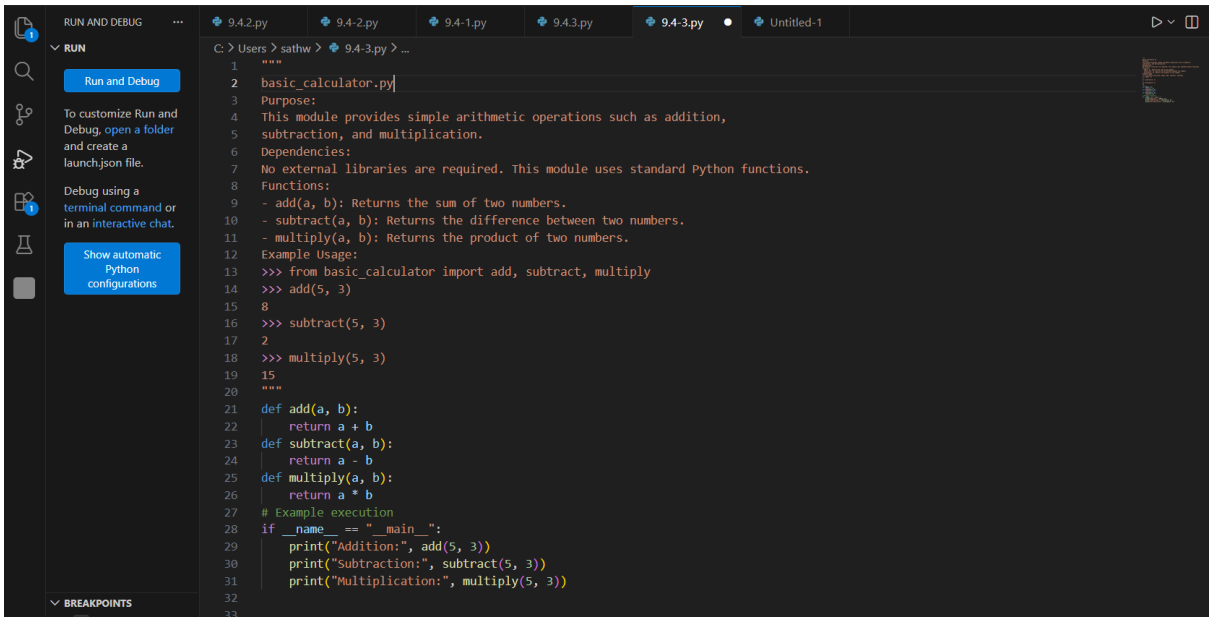
#OUTPUT:



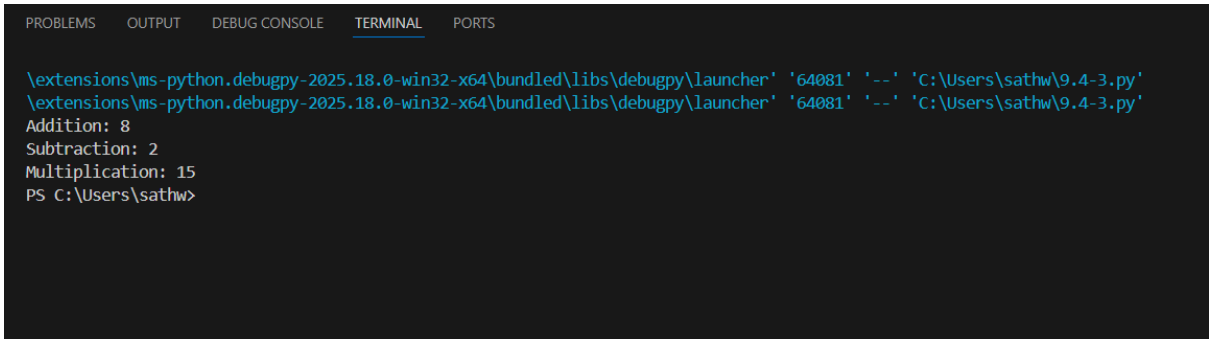
AI Generates Module-Level Docstring:

Generate a professional module-level docstring for this Python file that explains the module purpose, dependencies, key functions, and includes a short example usage, without changing the existing code.

#CODE AND INPUT:



#OUTPUT:



Task 3 is about **adding proper documentation at the top of a Python file**.

The goal is:

- To explain **what the module does**
- To mention **required libraries (if any)**
- To describe **main functions**
- To show **how to use it**

So when someone opens the file, they immediately understand its purpose without reading all the code.

👉 In short:
Task 3 improves professionalism and clarity by adding a clear module-level docstring.

Task 4: Converting Developer Comments into Structured Docstrings

Scenario

In a legacy project, developers have written long explanatory comments inside functions instead of proper docstrings. The team now wants to standardize documentation.

Task Description

You are given a Python script where functions contain detailed inline comments explaining their logic.

Use AI to:

• Automatically convert these comments into structured Google-style or NumPy-style docstrings

• Preserve the original meaning and intent of the comments

• Remove redundant inline comments after conversion

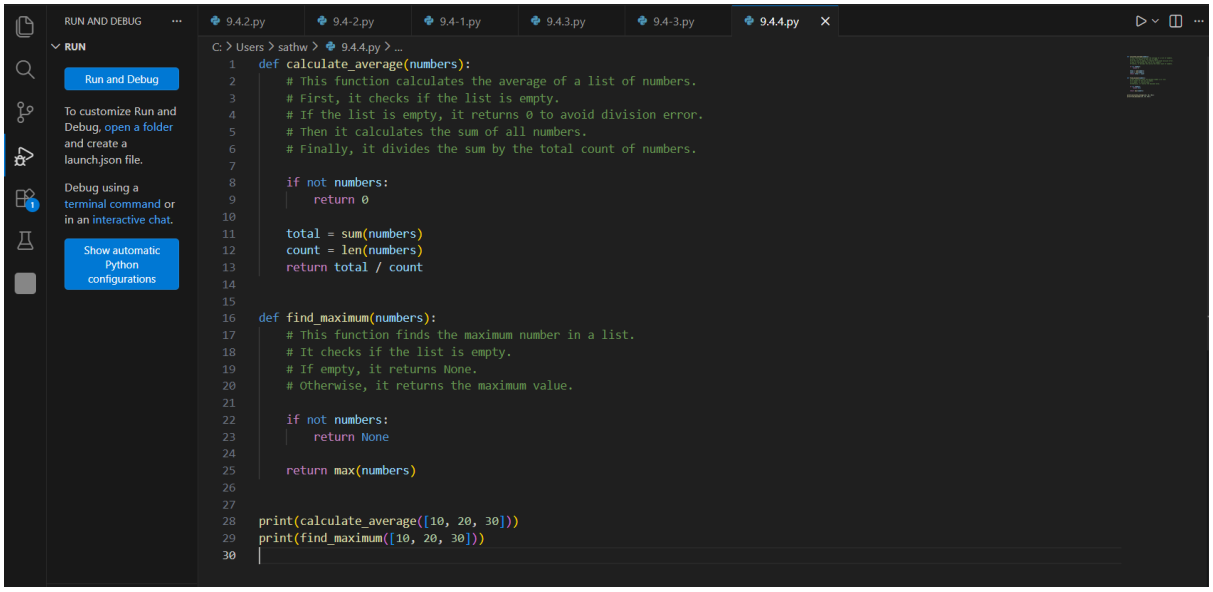
Expected Outcome

• Functions with clean, standardized docstrings

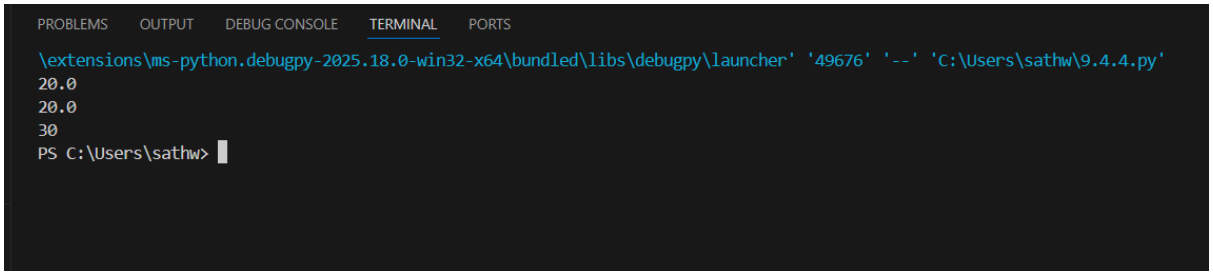
• Reduced clutter inside function bodies

• Improved consistency across the codebase

#CODE AND INPUT:



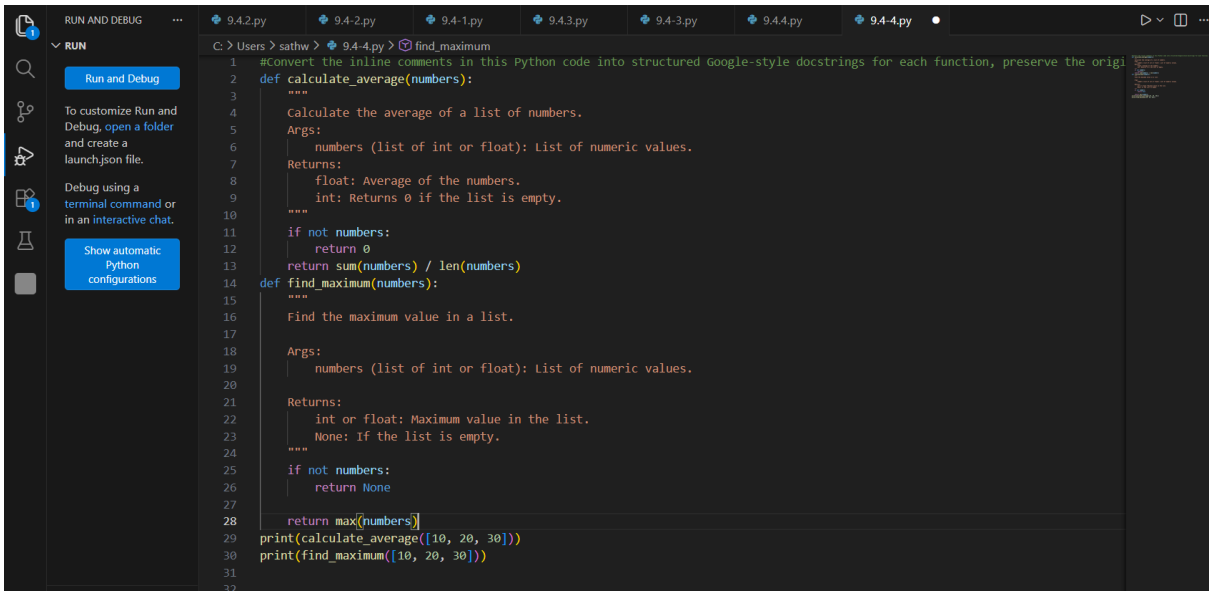
#OUTPUT:



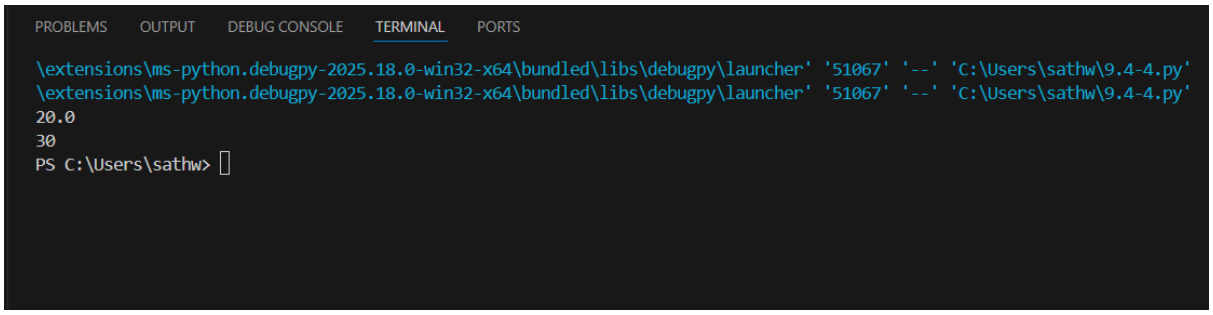
Converted Version:

#PROMPT:

Convert the inline comments in this Python code into structured Google-style docstrings for each function, preserve the original meaning, remove redundant comments inside the function body, and do not change the program logic.



#OUTPUT:



Task 5: Building a Mini Automatic Documentation Generator

Scenario

Your team wants a simple internal tool that helps developers start

documenting new Python files quickly, without writing documentation from scratch.

Task Description

Design a small Python utility that:

• Reads a given .py file

• Automatically detects:

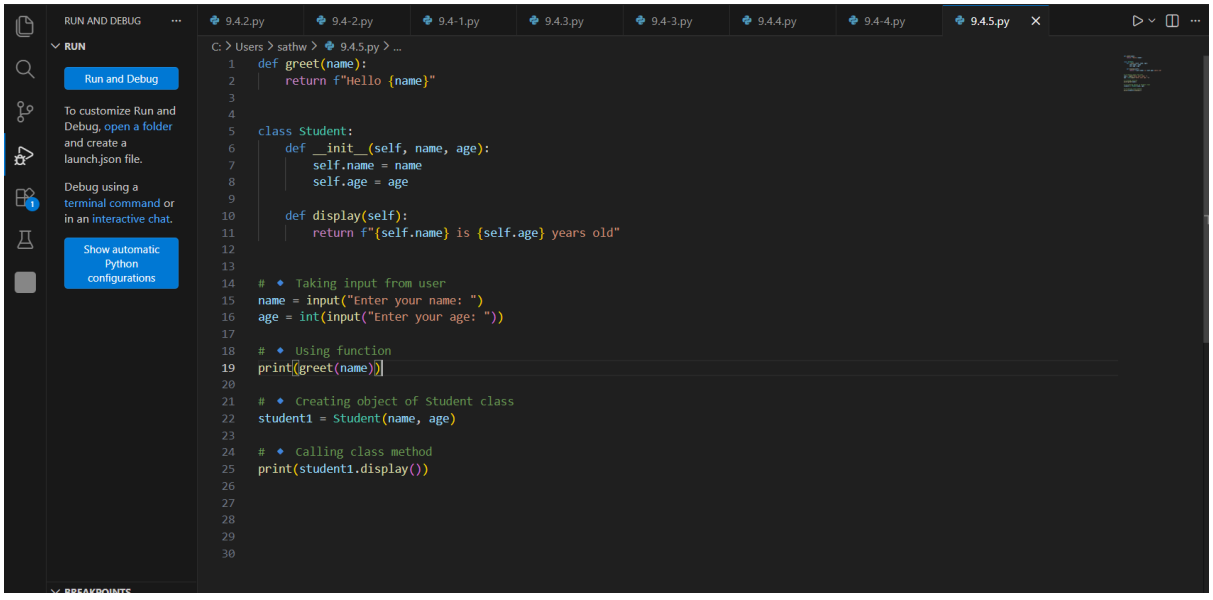
o Functions

o Classes

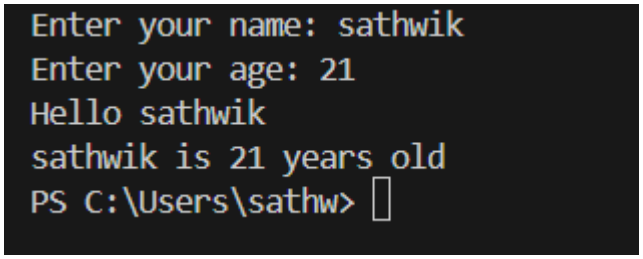
• Inserts placeholder Google-style docstrings for each detected function or class

AI tools may be used to assist in generating or refining this utility.

#CODE AND INPUT:

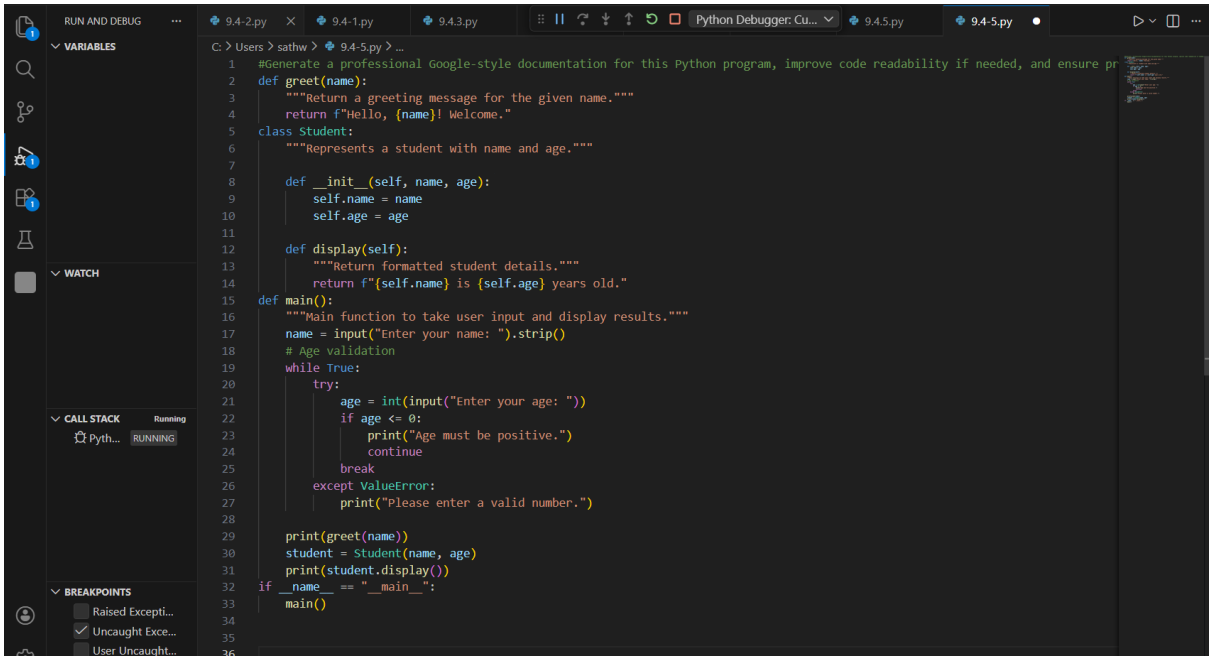


#OUTPUT:

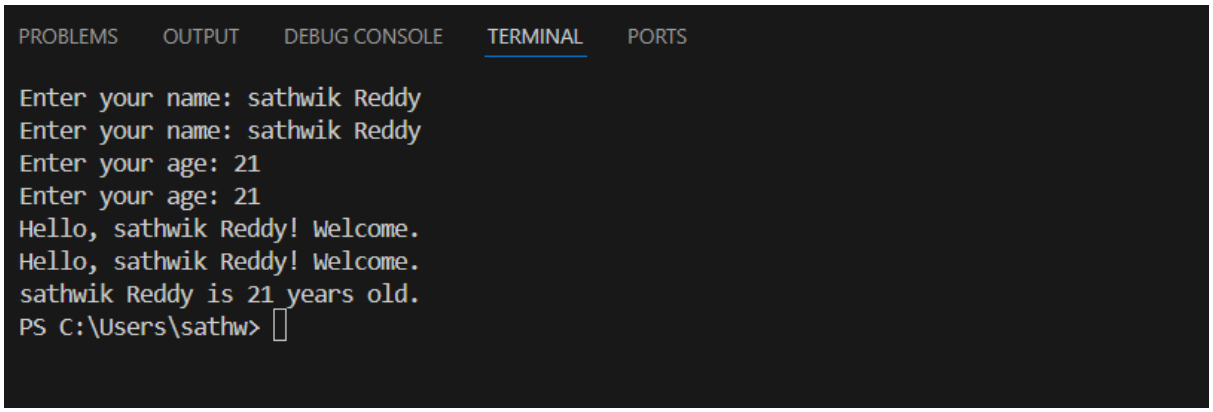


#PROMPT:

Generate a professional Google-style documentation for this Python program, improve code readability if needed, and ensure proper input validation and structured main() usage without changing the core functionality.



#OUTPUT:



#EXPLANATION:

This program:

- **Takes name and age from the user**

- **Validates age to ensure it is a positive number**
- **Uses a greet() function to display a welcome message**
- **Uses a Student class to store and display student details**
- **Uses a main() function to organize the program properly**

👉 In short:
It is a well-structured, validated, and professional version of a simple greeting and student information program.