# AI Assisted Coding

**Assignment-9.4**

**Name:M.Mohan venkatesh**

**BATCH-14**

**HTNO:2303A510F0**

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)

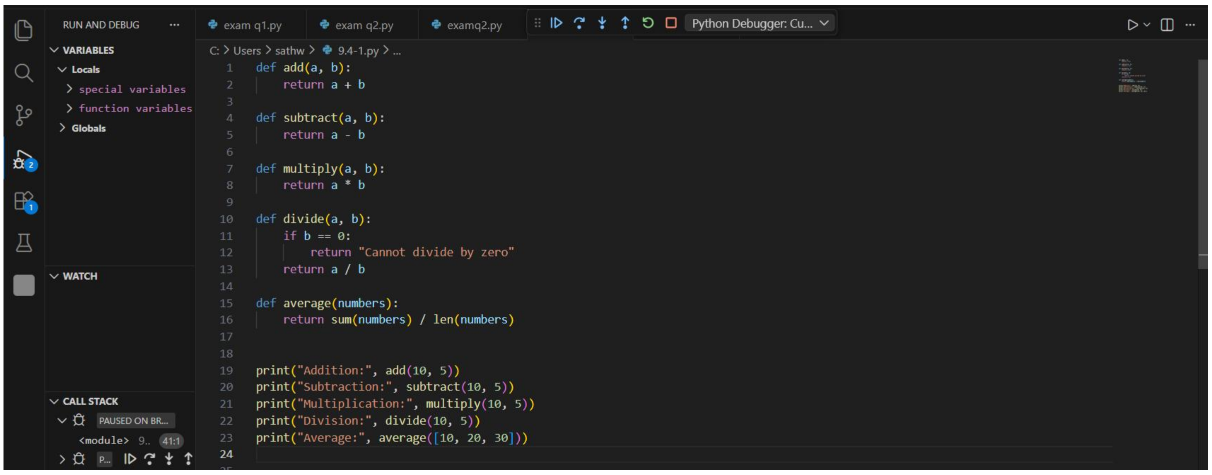
to observe quality differences.

Expected Outcome

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

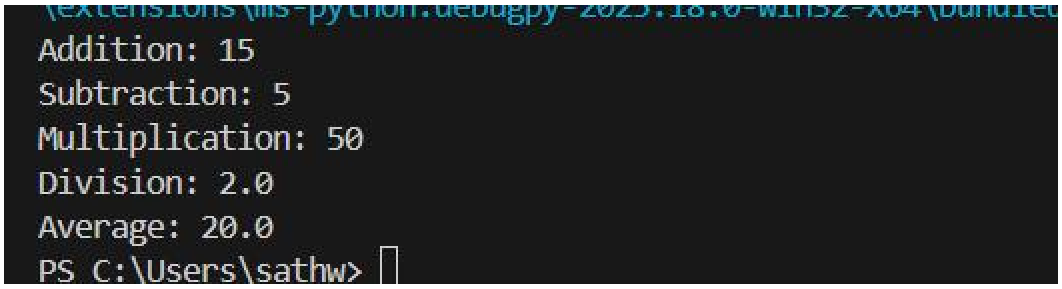
• Docstrings that clearly explain function behavior and usage

• Improved readability and usability of the codebase



```python
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

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

def divide(a, b):
    if b == 0:
        return "Cannot divide by zero"
    return a / b

def average(numbers):
    return sum(numbers) / len(numbers)


print("Addition:", add(10, 5))
print("Subtraction:", subtract(10, 5))
print("Multiplication:", multiply(10, 5))
print("Division:", divide(10, 5))
print("Average:", average([10, 20, 30]))
```

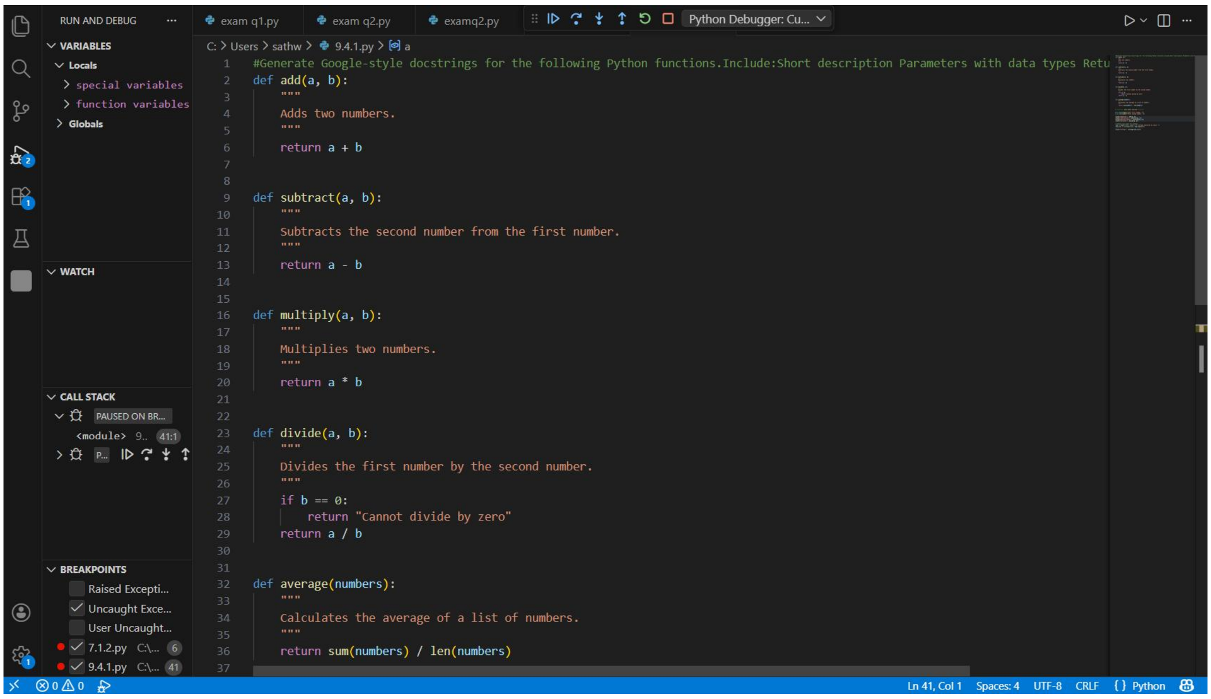
#OUTPUT:



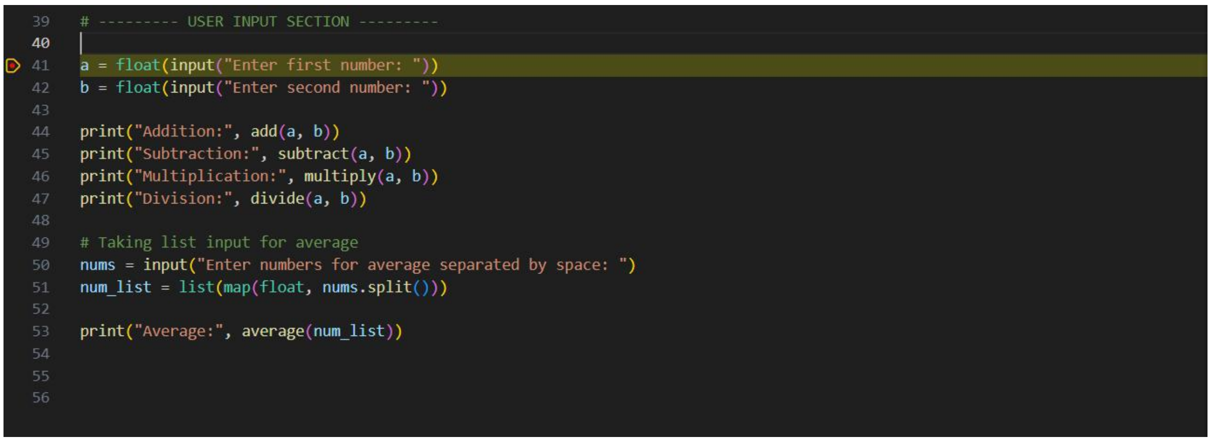
```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Average: 20.0
PS C:\Users\sathw>
```

#prompt:

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

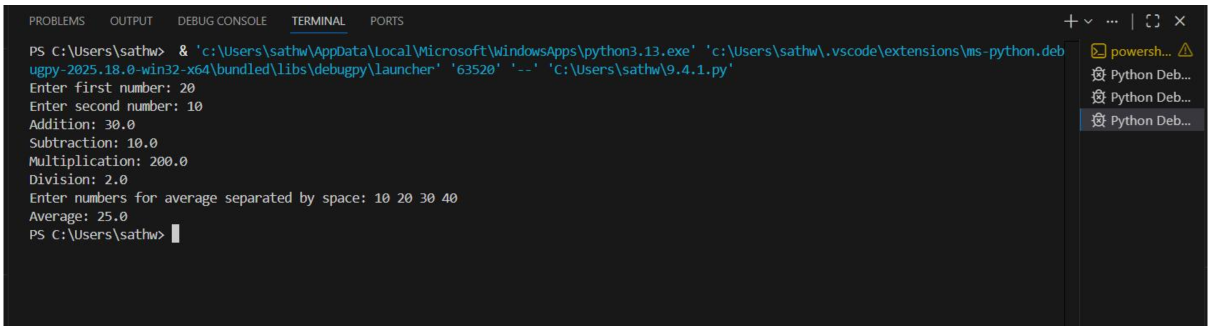


```
#Generate Google-style docstrings for the following Python functions.Include:Short description Parameters with data types Retu
def add(a, b):
    """
    Adds two numbers.
    """
    return a + b


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)
```



```
# --------- 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.

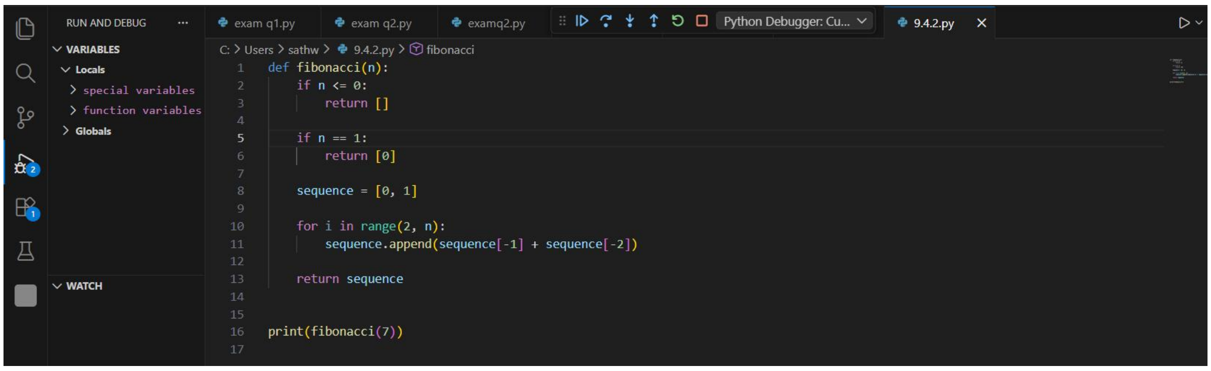
Expected Outcome

• A Python script with concise, meaningful inline comments

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

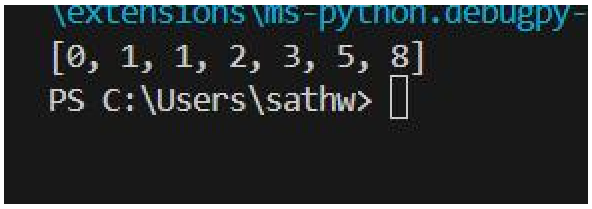
• Noticeable improvement in code readability

Original Code (Without Comments)



```python
def fibonacci(n):
    if n <= 0:
        return []

    if n == 1:
        return [0]

    sequence = [0, 1]

    for i in range(2, n):
        sequence.append(sequence[-1] + sequence[-2])

    return sequence


print(fibonacci(7))
```

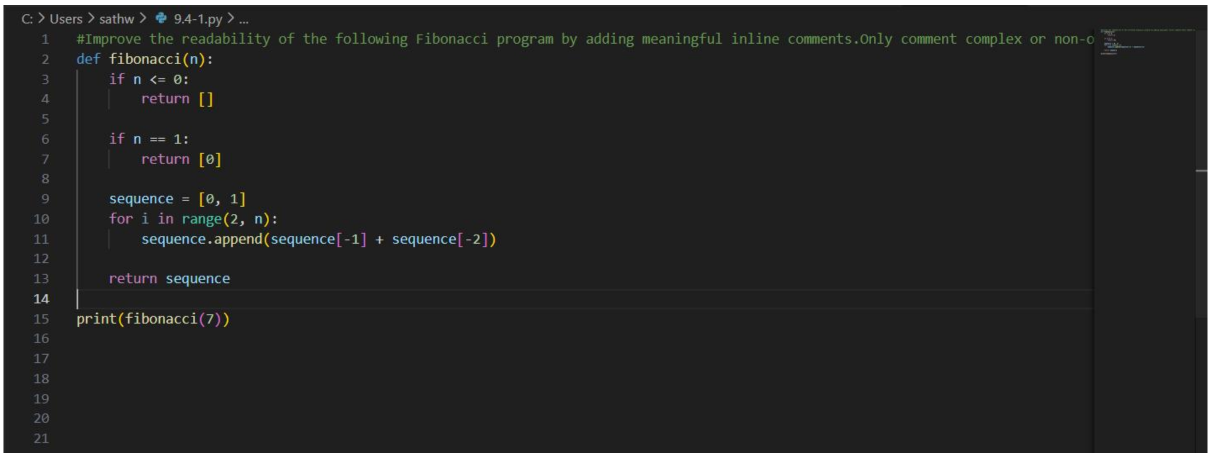
#OUTPUT:



```
\extensions\ms-python.debugpy-
[0, 1, 1, 2, 3, 5, 8]
PS C:\Users\sathw> 
```

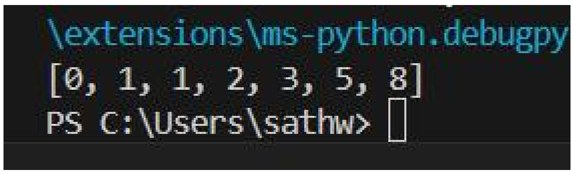
**AI-Generated Inline Comments (Improved Version)**

#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.



```python
#Improve the readability of the following Fibonacci program by adding meaningful inline comments.Only comment complex or non-o
def fibonacci(n):
    if n <= 0:
        return []

    if n == 1:
        return [0]

    sequence = [0, 1]
    for i in range(2, n):
        sequence.append(sequence[-1] + sequence[-2])

    return sequence

print(fibonacci(7))
```

#OUTPUT:



```
\extensions\ms-python.debugpy
[0, 1, 1, 2, 3, 5, 8]
PS C:\Users\sathw> 
```

# 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.

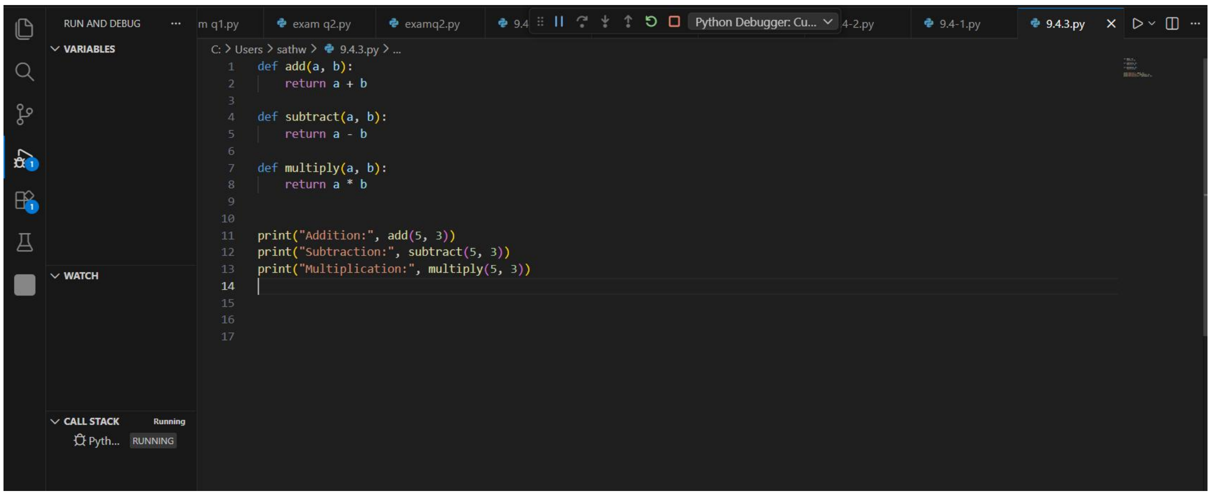
Expected Outcome

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

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

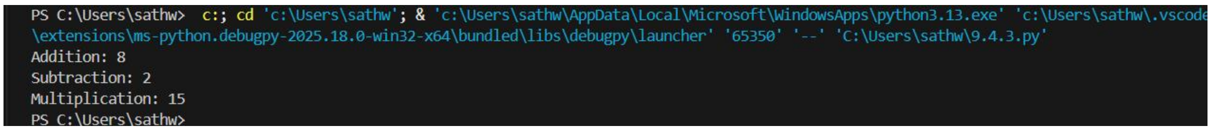
• Documentation suitable for real-world projects or repositories

#CODE AND INPUT



```python
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

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


print("Addition:", add(5, 3))
print("Subtraction:", subtract(5, 3))
print("Multiplication:", multiply(5, 3))
```

#OUTPUT:

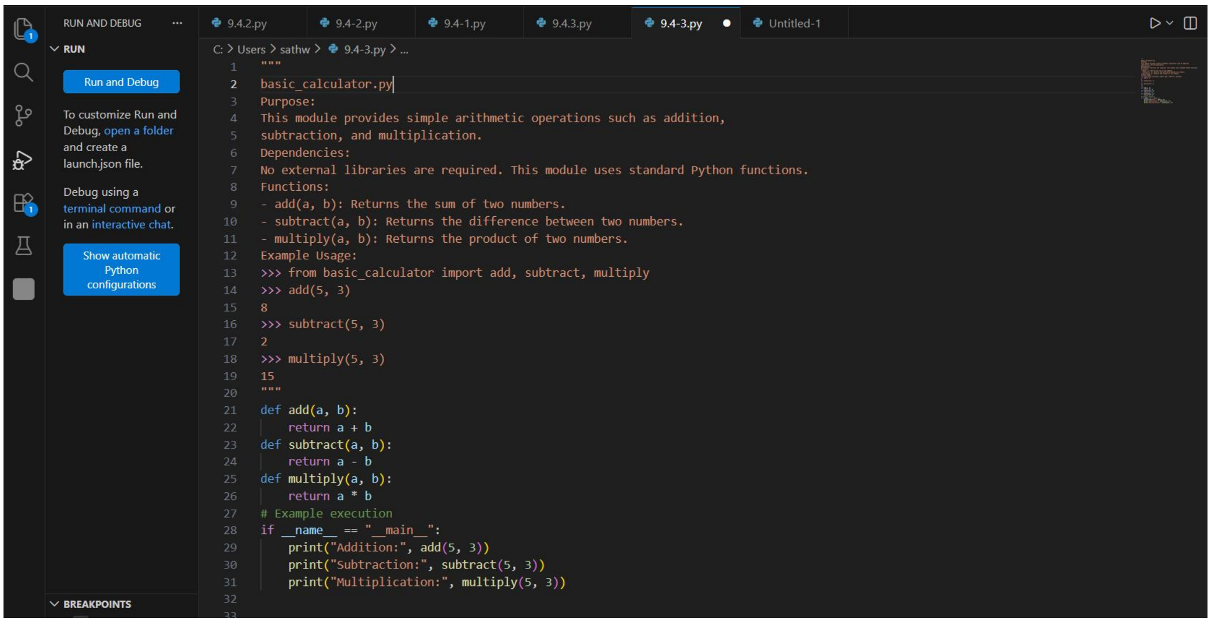


```
PS C:\Users\sathw>  c:; cd 'c:\Users\sathw'; & 'c:\Users\sathw\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\sathw\.vscode
\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65350' '--' 'C:\Users\sathw\9.4.3.py'
Addition: 8
Subtraction: 2
Multiplication: 15
PS C:\Users\sathw>
```

AI Generates Module-Level Docstring:

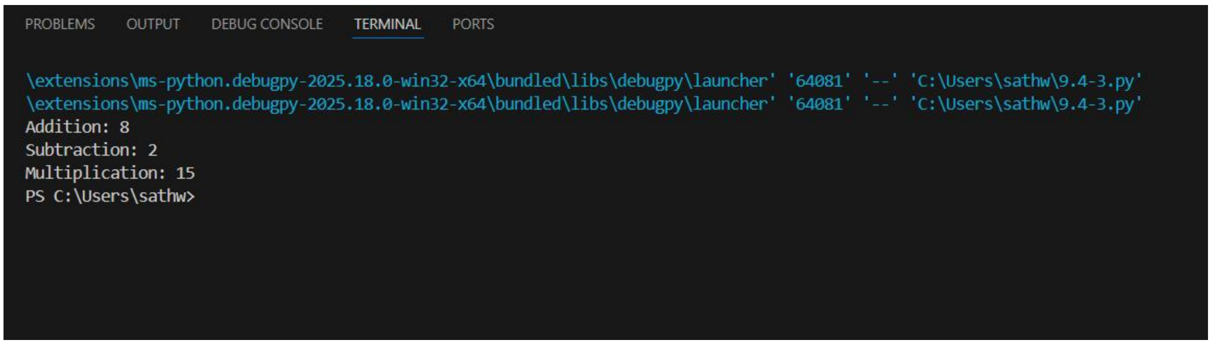
# Generate a professional module-level docstring for this Python file that explains the module purpose, dependencies, key functions, and includes a short example usage, without changing the existing code.

#CODE AND INPUT:


RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Debug using a terminal command or in an interactive chat.
Show automatic Python configurations
BREAKPOINTS
9.4-3.py

```python
"""
basic_calculator.py
Purpose:
This module provides simple arithmetic operations such as addition,
subtraction, and multiplication.
Dependencies:
No external libraries are required. This module uses standard Python functions.
Functions:
- add(a, b): Returns the sum of two numbers.
- subtract(a, b): Returns the difference between two numbers.
- multiply(a, b): Returns the product of two numbers.
Example Usage:
>>> from basic_calculator import add, subtract, multiply
>>> add(5, 3)
8
>>> subtract(5, 3)
2
>>> multiply(5, 3)
15
"""
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
# Example execution
if __name__ == "__main__":
    print("Addition:", add(5, 3))
    print("Subtraction:", subtract(5, 3))
    print("Multiplication:", multiply(5, 3))
```

#OUTPUT:



```
\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64081' '--' 'C:\Users\sathw\9.4-3.py'
\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64081' '--' 'C:\Users\sathw\9.4-3.py'
Addition: 8
Subtraction: 2
Multiplication: 15
PS C:\Users\sathw>
```

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

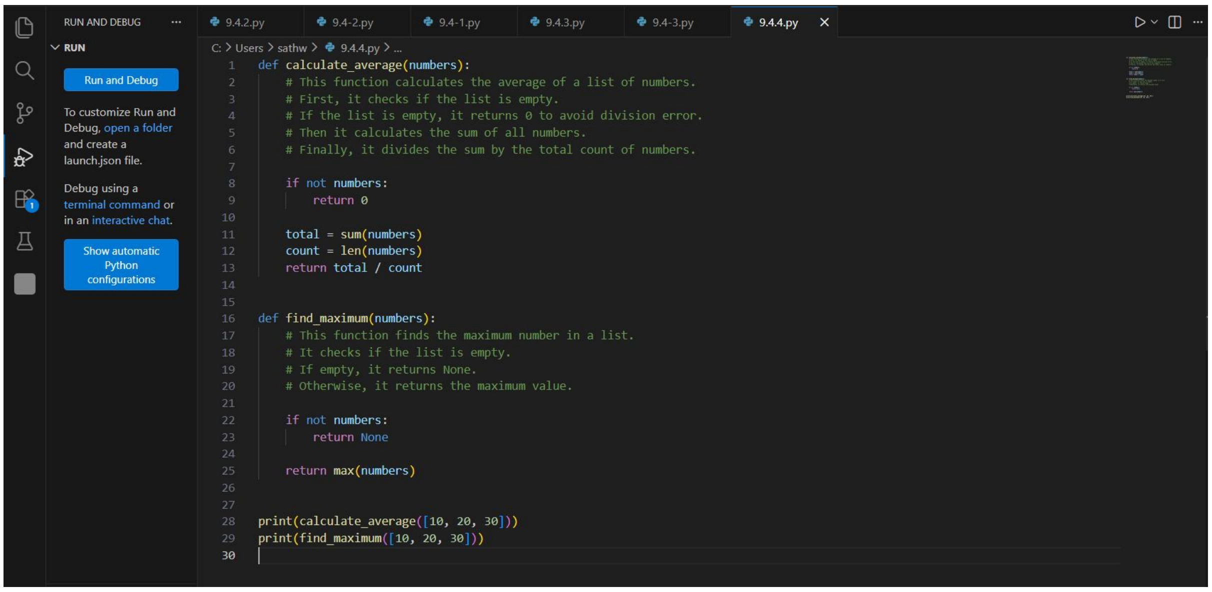
Expected Outcome

• Functions with clean, standardized docstrings

• Reduced clutter inside function bodies

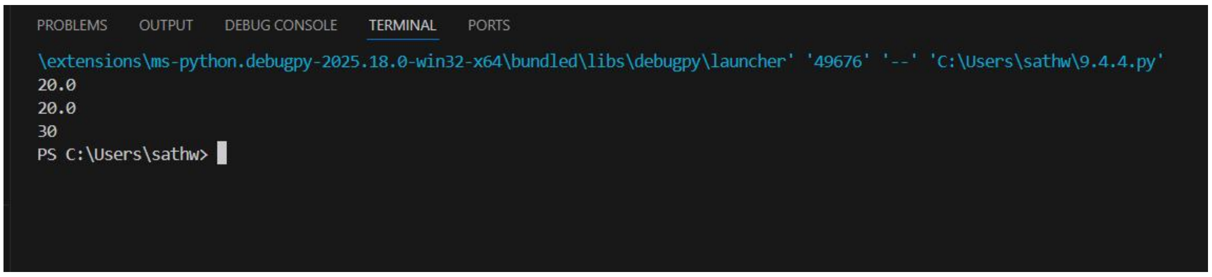
• Improved consistency across the codebase

#CODE AND INPUT:



```python
def calculate_average(numbers):
    # This function calculates the average of a list of numbers.
    # First, it checks if the list is empty.
    # If the list is empty, it returns 0 to avoid division error.
    # Then it calculates the sum of all numbers.
    # Finally, it divides the sum by the total count of numbers.

    if not numbers:
        return 0

    total = sum(numbers)
    count = len(numbers)
    return total / count


def find_maximum(numbers):
    # This function finds the maximum number in a list.
    # It checks if the list is empty.
    # If empty, it returns None.
    # Otherwise, it returns the maximum value.

    if not numbers:
        return None

    return max(numbers)


print(calculate_average([10, 20, 30]))
print(find_maximum([10, 20, 30]))
```

#OUTPUT:

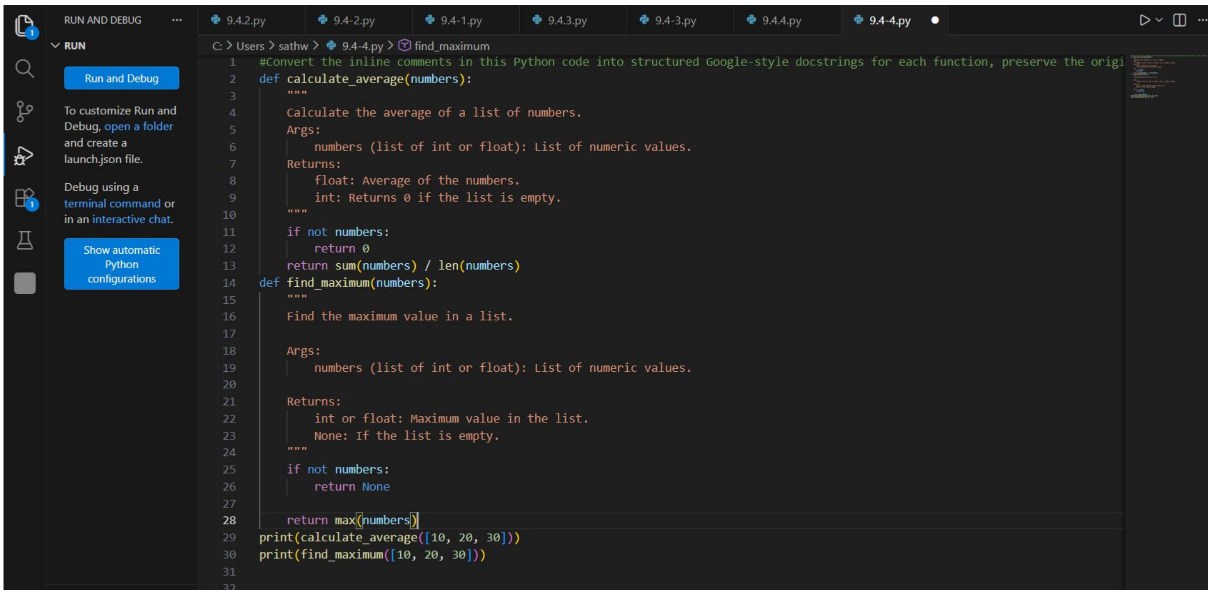


```
\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '49676' '--' 'C:\Users\sathw\9.4.4.py'
20.0
20.0
30
PS C:\Users\sathw>
```

**Converted Version:**

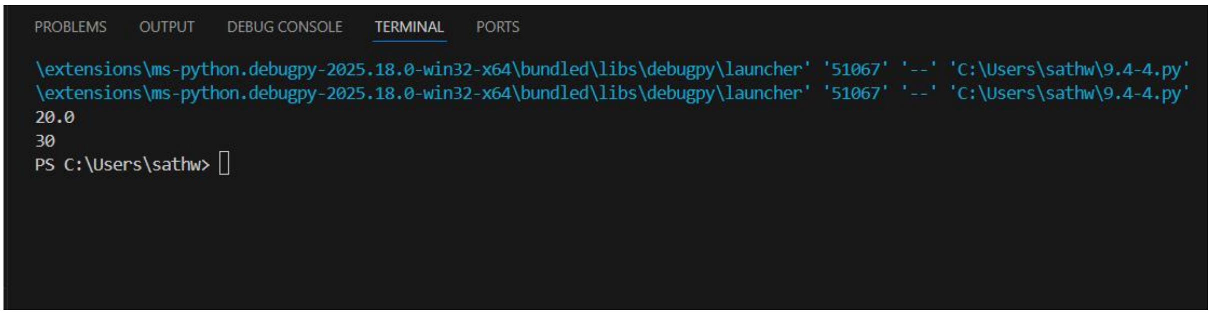
**#PROMPT:**

**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.**


RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Debug using a terminal command or in an interactive chat.
Show automatic Python configurations
9.4.2.py
9.4-2.py
9.4-1.py
9.4.3.py
9.4-3.py
9.4.4.py
9.4-4.py
C: > Users > sathw > 9.4-4.py > find_maximum

```python
#Convert the inline comments in this Python code into structured Google-style docstrings for each function, preserve the origi
def calculate_average(numbers):
    """
    Calculate the average of a list of numbers.
    Args:
        numbers (list of int or float): List of numeric values.
    Returns:
        float: Average of the numbers.
        int: Returns 0 if the list is empty.
    """
    if not numbers:
        return 0
    return sum(numbers) / len(numbers)
def find_maximum(numbers):
    """
    Find the maximum value in a list.

    Args:
        numbers (list of int or float): List of numeric values.

    Returns:
        int or float: Maximum value in the list.
        None: If the list is empty.
    """
    if not numbers:
        return None

    return max(numbers)
print(calculate_average([10, 20, 30]))
print(find_maximum([10, 20, 30]))
```

**#OUTPUT:**



```
\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51067' '--' 'C:\Users\sathw\9.4-4.py'
\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51067' '--' 'C:\Users\sathw\9.4-4.py'
20.0
30
PS C:\Users\sathw>
```

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:

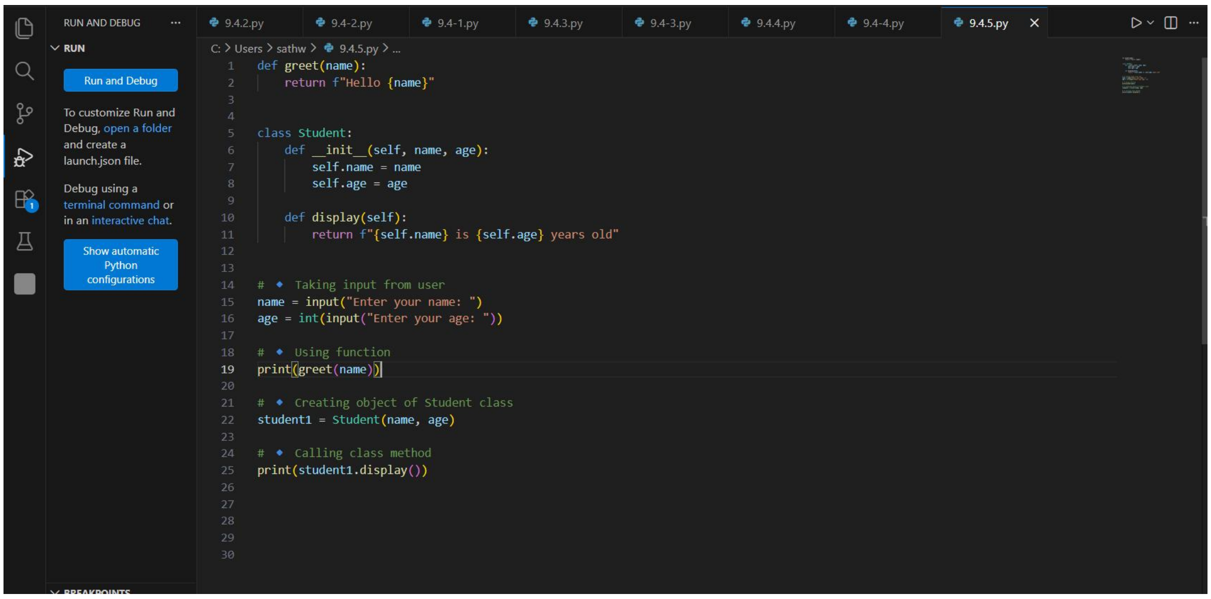
o Functions

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:

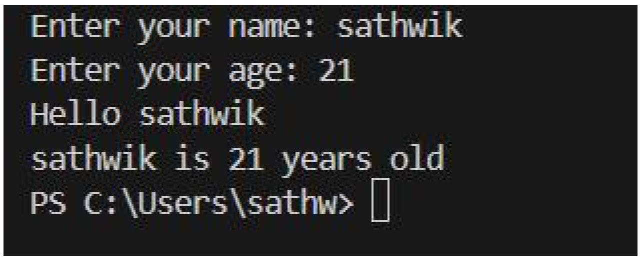


To customize Run and Debug, open a folder and create a launch.json file.

Debug using a terminal command or in an interactive chat.

```python
def greet(name):
    return f"Hello {name}"


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

    def display(self):
        return f"{self.name} is {self.age} years old"


# 🔹 Taking input from user
name = input("Enter your name: ")
age = int(input("Enter your age: "))

# 🔹 Using function
print(greet(name))

# 🔹 Creating object of Student class
student1 = Student(name, age)

# 🔹 Calling class method
print(student1.display())
```

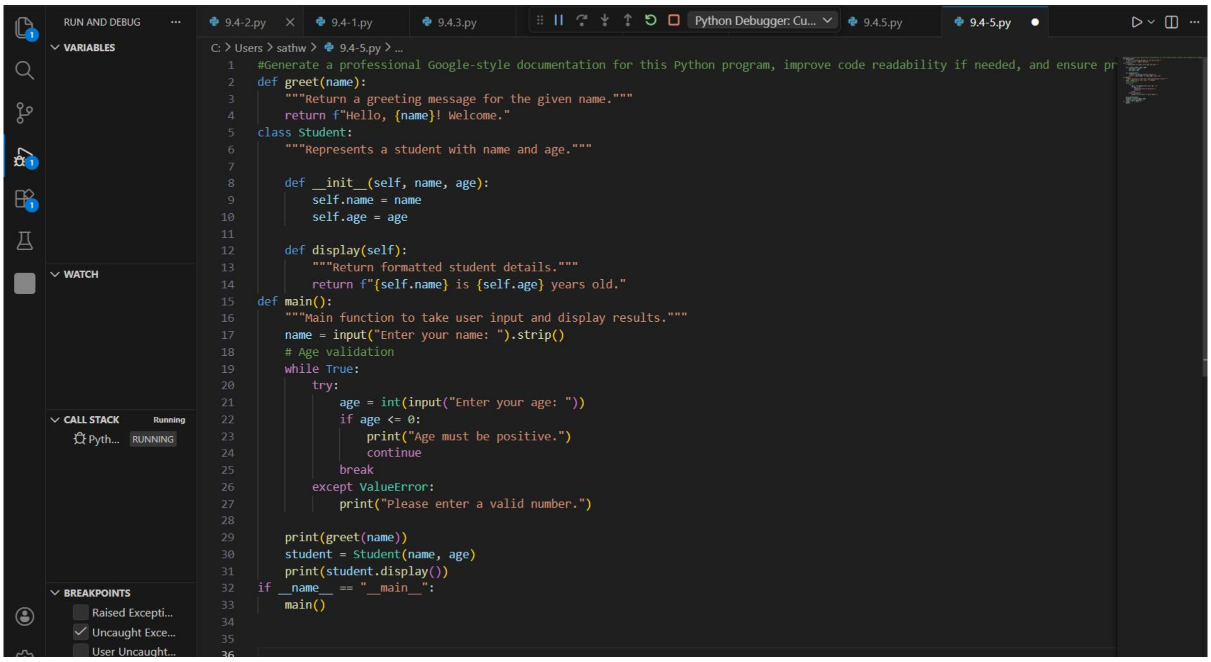
#OUTPUT:



```
Enter your name: sathwik
Enter your age: 21
Hello sathwik
sathwik is 21 years old
PS C:\Users\sathw> 
```

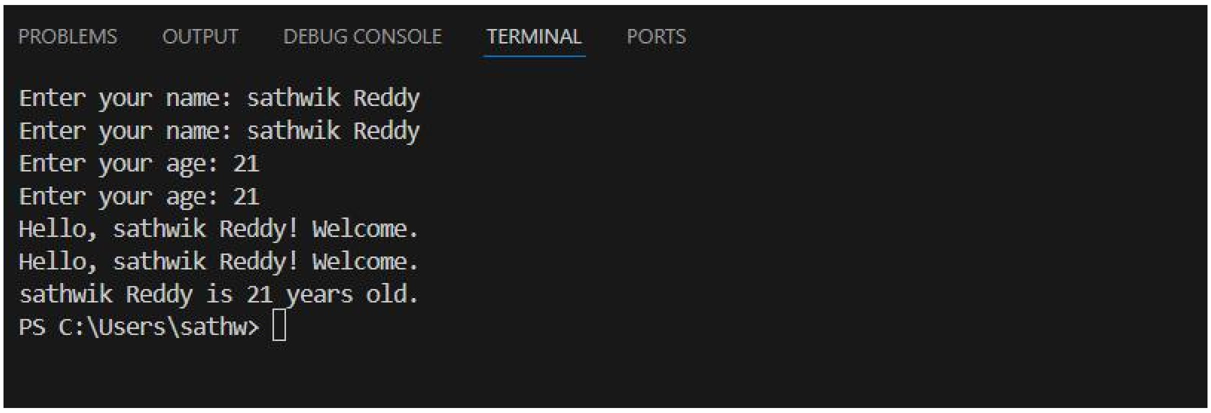
## #PROMPT:

**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.**



```python
#Generate a professional Google-style documentation for this Python program, improve code readability if needed, and ensure pr
def greet(name):
    """Return a greeting message for the given name."""
    return f"Hello, {name}! Welcome."
class Student:
    """Represents a student with name and age."""

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        """Return formatted student details."""
        return f"{self.name} is {self.age} years old."
def main():
    """Main function to take user input and display results."""
    name = input("Enter your name: ").strip()
    # Age validation
    while True:
        try:
            age = int(input("Enter your age: "))
            if age <= 0:
                print("Age must be positive.")
                continue
            break
        except ValueError:
            print("Please enter a valid number.")

    print(greet(name))
    student = Student(name, age)
    print(student.display())
if __name__ == "__main__":
    main()
```

## #OUTPUT:



```
Enter your name: sathwik Reddy
Enter your name: sathwik Reddy
Enter your age: 21
Enter your age: 21
Hello, sathwik Reddy! Welcome.
Hello, sathwik Reddy! Welcome.
sathwik Reddy is 21 years old.
PS C:\Users\sathw>
```

## #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.**