AI ASSISTED CODING

ASSIGNMENT-9.2

NAME : G.OMKAR

HT.NO : 2403A52039

BATCH : 03

Task 1 :

(Documentation – Google-Style Docstrings for Python Functions)

• Task: Use AI to add Google-style docstrings to all functions in a given Python script.

• Instructions:

o Prompt AI to generate docstrings without providing any input-output examples.

o Ensure each docstring includes:

Function description

Parameters with type hints

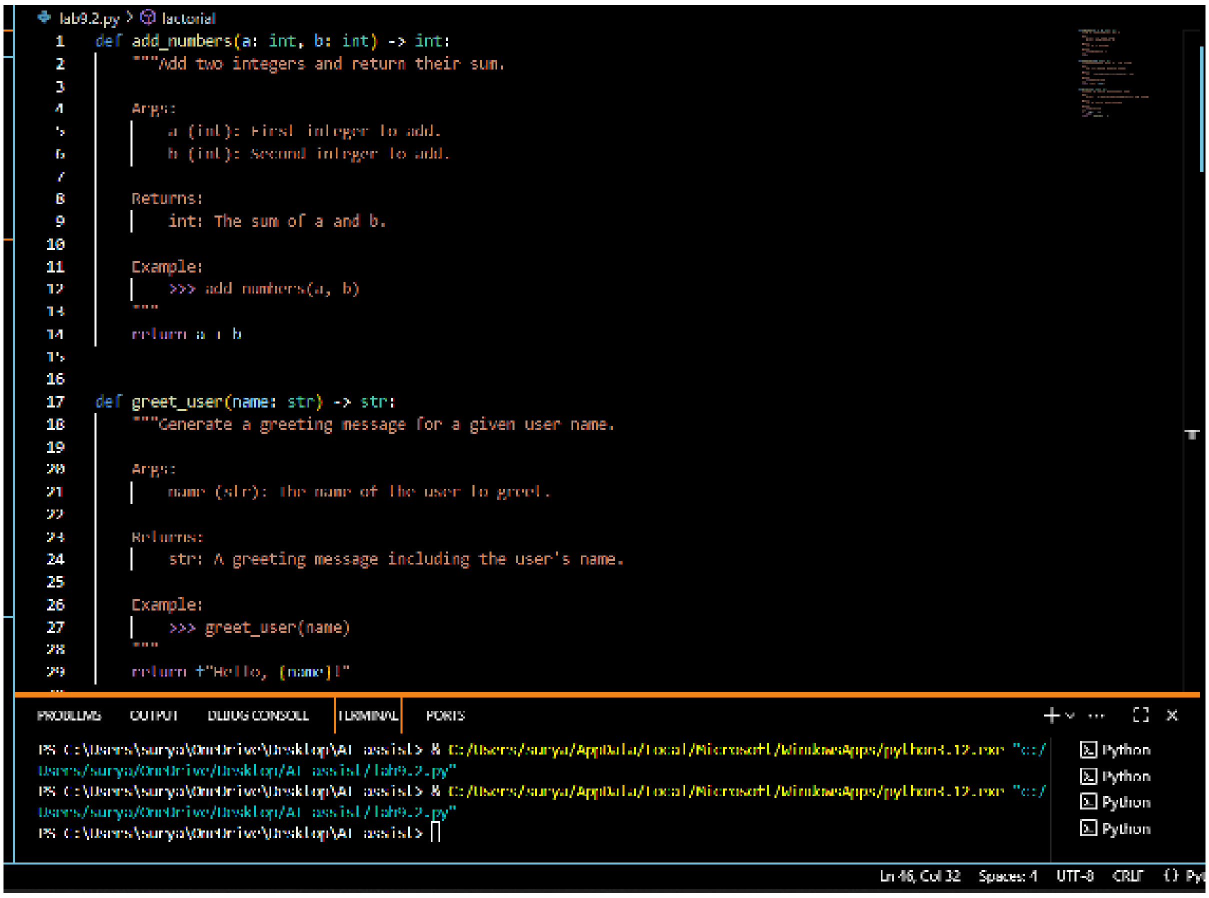
Return values with type hints

Example usage

o Review the generated docstrings for accuracy and formatting

Prompt : Add Google-style docstrings to all functions in the Python script. Each docstring should include a short description, parameters with type hints, return type, and an example usage. Do not change the code logic or include input-output examples

CODE :



```
lab9.2.py > factorial
def add_numbers(a: int, b: int) -> int:
    """Add two integers and return their sum.

    Args:
        a (int): First integer to add.
        b (int): Second integer to add.

    Returns:
        int: The sum of a and b.

    Example:
        >>> add_numbers(a, b)
    """
    return a + b


def greet_user(name: str) -> str:
    """Generate a greeting message for a given user name.

    Args:
        name (str): The name of the user to greet.

    Returns:
        str: A greeting message including the user's name.

    Example:
        >>> greet_user(name)
    """
    return f"Hello, {name}!"
```

```
PS C:\Users\surya\OneDrive\Desktop\AI assist> & C:/Users/surya/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/surya/OneDrive/Desktop/AI assist/lab9.2.py"
PS C:\Users\surya\OneDrive\Desktop\AI assist> & C:/Users/surya/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/surya/OneDrive/Desktop/AI assist/lab9.2.py"
PS C:\Users\surya\OneDrive\Desktop\AI assist>
```

Observation : AI successfully added Google-style docstrings to all functions. Each docstring includes a clear description, parameters with type hints, return type, and example usage. The formatting is consistent and improves code readability.

Task 2 :

(Documentation – Inline Comments for Complex Logic)

• Task: Use AI to add meaningful inline comments to a Python program explaining only complex logic parts.

• Instructions:

o Provide a Python script without comments to the AI.

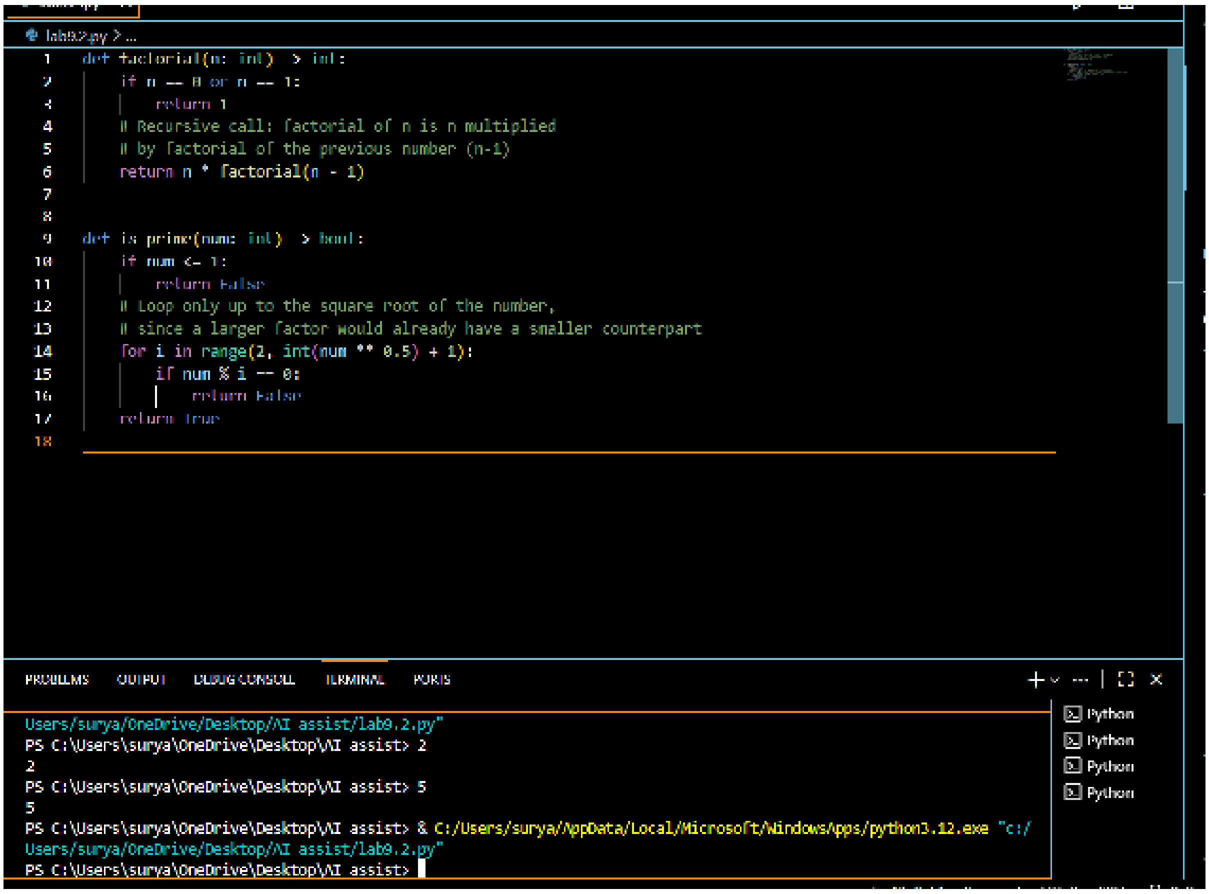
o Instruct AI to skip obvious syntax explanations and focus

only on tricky or non-intuitive code sections.

o Verify that comments improve code readability and maintainability.

Prompt : Add meaningful inline comments to the Python script. Only explain complex or non-intuitive logic. Do not add comments for simple syntax or obvious operations. Ensure comments improve readability and maintainability

CODE :



```python
def factorial(n: int) -> int:
    if n == 0 or n == 1:
        return 1
    # Recursive call: factorial of n is n multiplied
    # by factorial of the previous number (n-1)
    return n * factorial(n - 1)


def is_prime(num: int) -> bool:
    if num <= 1:
        return False
    # Loop only up to the square root of the number,
    # since a larger factor would already have a smaller counterpart
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True
```

```
Users/surya/OneDrive/Desktop/AI assist/lab9.2.py"
PS C:\Users\surya\OneDrive\Desktop\AI assist> 2
2
PS C:\Users\surya\OneDrive\Desktop\AI assist> 5
5
PS C:\Users\surya\OneDrive\Desktop\AI assist> & C:/Users/surya/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/
Users/surya/OneDrive/Desktop/AI assist/lab9.2.py"
PS C:\Users\surya\OneDrive\Desktop\AI assist>
```

Observation : AI added concise inline comments only to complex logic parts, such as recursion and loop conditions. Obvious syntax was skipped, making the code more readable and easier to maintain without unnecessary clutter.

Task 3 :

(Documentation – Module-Level Documentation)

• Task: Use AI to create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file.

• Instructions:

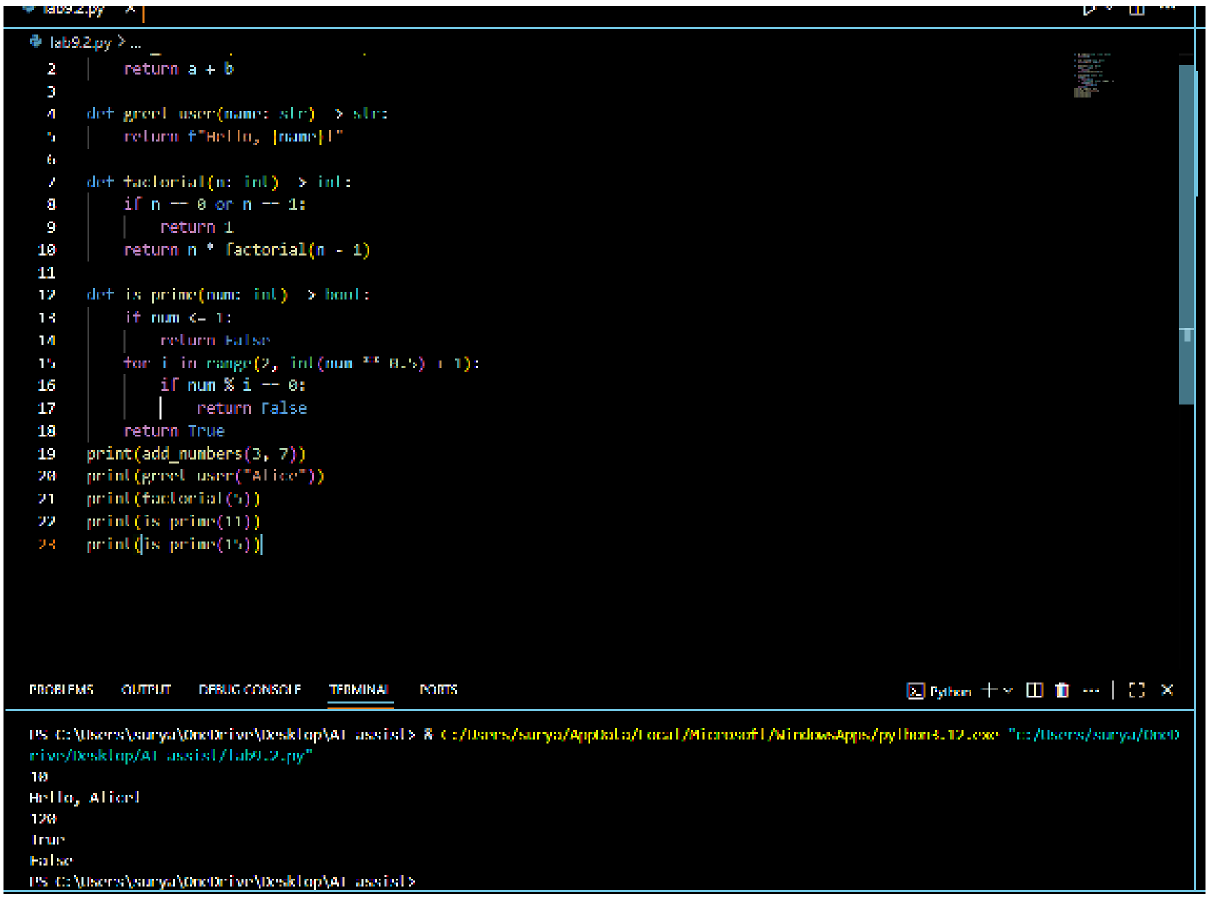
o Supply the entire Python file to AI.

o Instruct AI to write a single multi-line docstring at the top of the file.

o Ensure the docstring clearly describes functionality and usage without rewriting the entire code

Prompt : Write a module-level docstring for the given Python file. The docstring should summarize the purpose of the module, list any dependencies, and describe the main functions or classes. Place the docstring at the very top of the file. Do not rewrite the code, only add the documentation. Keep it clear, concise, and formatted in Google-style.

CODE :



```python
    return a + b

def greet_user(name: str) -> str:
    return f"Hello, {name}!"

def factorial(n: int) -> int:
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

def is_prime(num: int) -> bool:
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True
print(add_numbers(3, 7))
print(greet_user("Alice"))
print(factorial(5))
print(is_prime(11))
print(is_prime(15))
```

```
PS C:\Users\surya\OneDrive\Desktop\AI assist> & C:/Users/surya/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/surya/OneD
rive/Desktop/AI assist/lab9.2.py"
10
Hello, Alice!
120
True
False
PS C:\Users\surya\OneDrive\Desktop\AI assist>
```

Observation :

AI generated a clear module-level docstring placed at the top of the file. The docstring summarizes the purpose, dependencies, and key functions of the module. It improves readability, provides context for users, and makes the file easier to maintain.

Task 4 :

(Documentation – Convert Comments to Structured Docstrings)

• Task: Use AI to transform existing inline comments into structured function docstrings following Google style.

• Instructions:

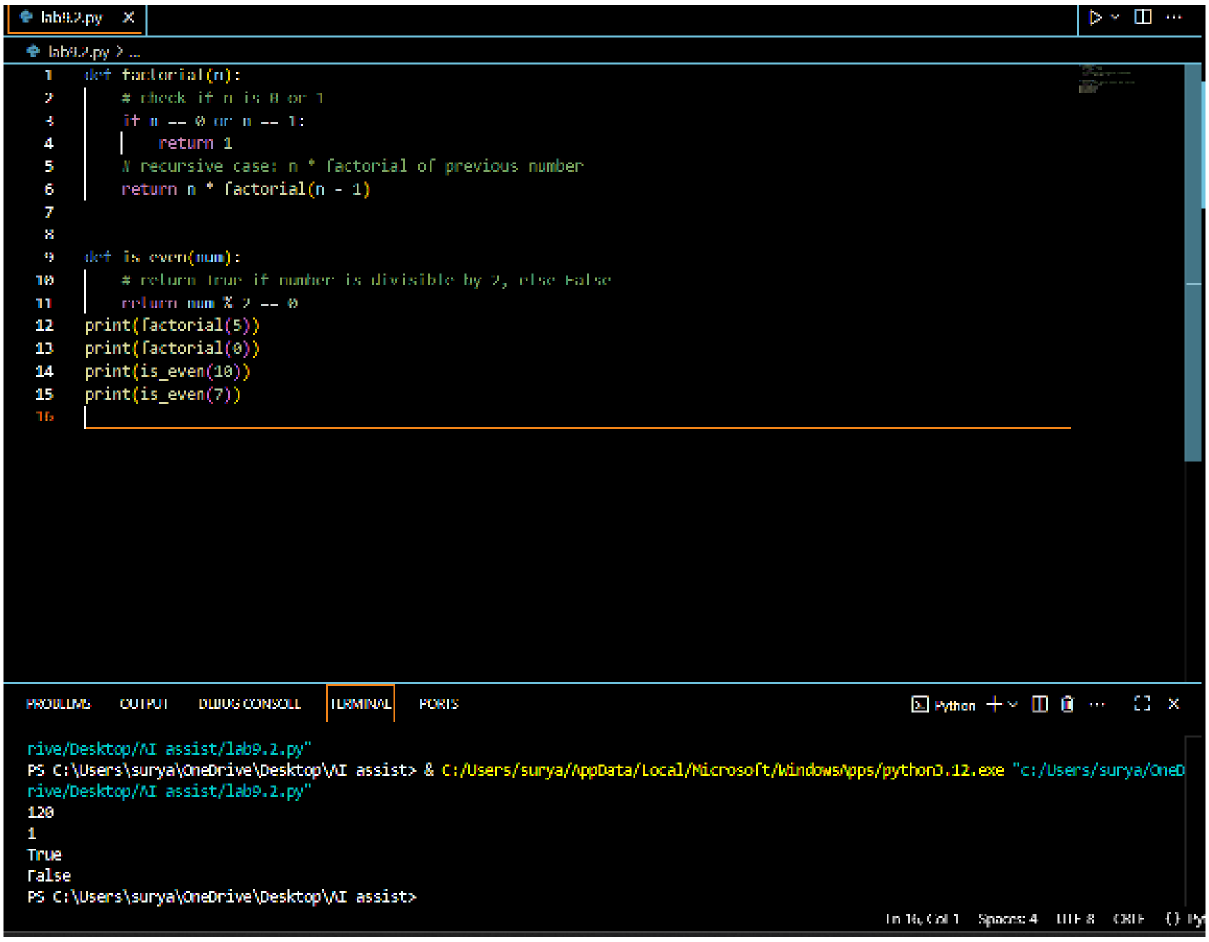
o Provide AI with Python code containing inline comments.

o Ask AI to move relevant details from comments into function docstrings.

o Verify that the new docstrings keep the meaning intact while improving structure

Prompt : Transform all inline comments in the given Python script into structured Google-style docstrings. Each docstring should include a brief description, parameters with type hints, return values, and an example usage. Remove the original inline comments, but preserve their meaning in the new docstrings. Do not change the code logic

CODE :



lab9.2.py

```python
def factorial(n):
    # check if n is 0 or 1
    if n == 0 or n == 1:
        return 1
    # recursive case: n * factorial of previous number
    return n * factorial(n - 1)


def is_even(num):
    # return True if number is divisible by 2, else False
    return num % 2 == 0
print(factorial(5))
print(factorial(0))
print(is_even(10))
print(is_even(7))
```

```
rive/Desktop/AI assist/lab9.2.py"
PS C:\Users\surya\OneDrive\Desktop\AI assist> & C:/Users/surya/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/surya/OneD
rive/Desktop/AI assist/lab9.2.py"
120
1
True
False
PS C:\Users\surya\OneDrive\Desktop\AI assist>
```

Observation : AI successfully converted inline comments into structured Google-style docstrings. The meaning of the original comments was preserved, and the functions now have standardized documentation, improving readability and maintainability.

Task 5 :

(Documentation – Review and Correct Docstrings)

• Task: Use AI to identify and correct inaccuracies in existing docstrings.

• Instructions:

o Provide Python code with outdated or incorrect docstrings.

o Instruct AI to rewrite each docstring to match the current code behavior.

o Ensure corrections follow Google-style formatting.

Prompt : You are given Python code with inaccurate or outdated docstrings.

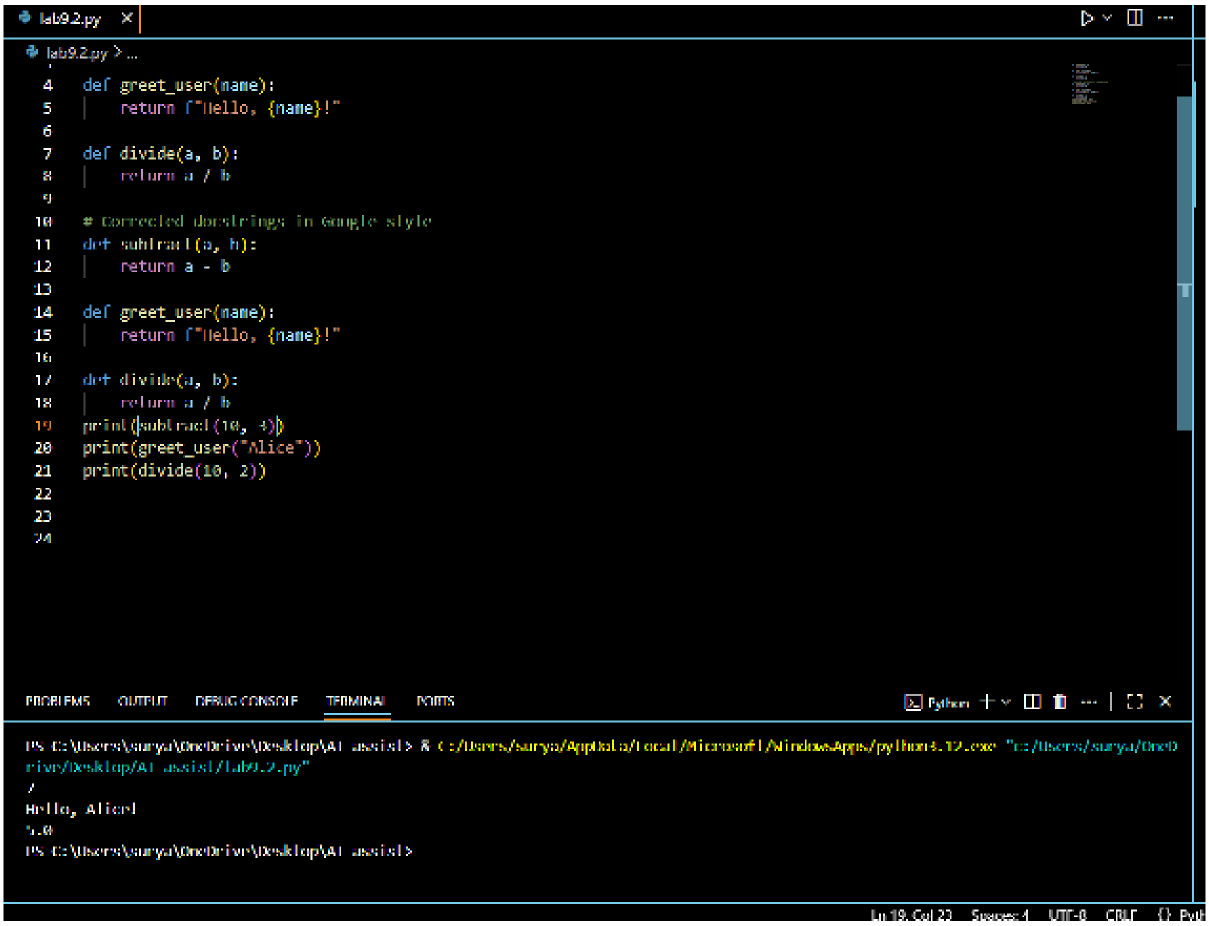
Your task is to:

1. Review each function's docstring.
2. Correct it so it accurately describes the current behavior of the function.
3. Follow Google-style docstring formatting.

Example code:

```
def add(a, b):
    """Subtracts b from a and returns the result."""
    return a + b
```

CODE :



```python
def greet_user(name):
    return f"Hello, {name}!"

def divide(a, b):
    return a / b

# Corrected docstrings in Google style
def subtract(a, b):
    return a - b

def greet_user(name):
    return f"Hello, {name}!"

def divide(a, b):
    return a / b
print(subtract(10, 4))
print(greet_user("Alice"))
print(divide(10, 2))
```

```
PS C:\Users\surya\OneDrive\Desktop\AI assist> & c:/Users/surya/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/surya/OneDrive/Desktop/AI assist/lab9.2.py"
6
Hello, Alice!
5.0
PS C:\Users\surya\OneDrive\Desktop\AI assist>
```

Observation : The original docstrings were inaccurate and misleading, describing behaviors that the functions did not perform. After correction, the docstrings now accurately reflect what each function does, including parameters, return values, and exceptions, using clear Google-style formatting. This improves readability, maintainability, and ensures developers understand the actual behavior of the code.

Task 6 :

(Documentation – Prompt Comparison Experiment)

• Task: Compare documentation output from a vague prompt and a detailed prompt for the same Python function.

• Instructions:

o Create two prompts: one simple (“Add comments to this function”) and one detailed (“Add Google-style docstrings with parameters, return types, and examples”).

o Use AI to process the same Python function with both prompts.

o Analyze and record differences in quality, accuracy, and completeness.

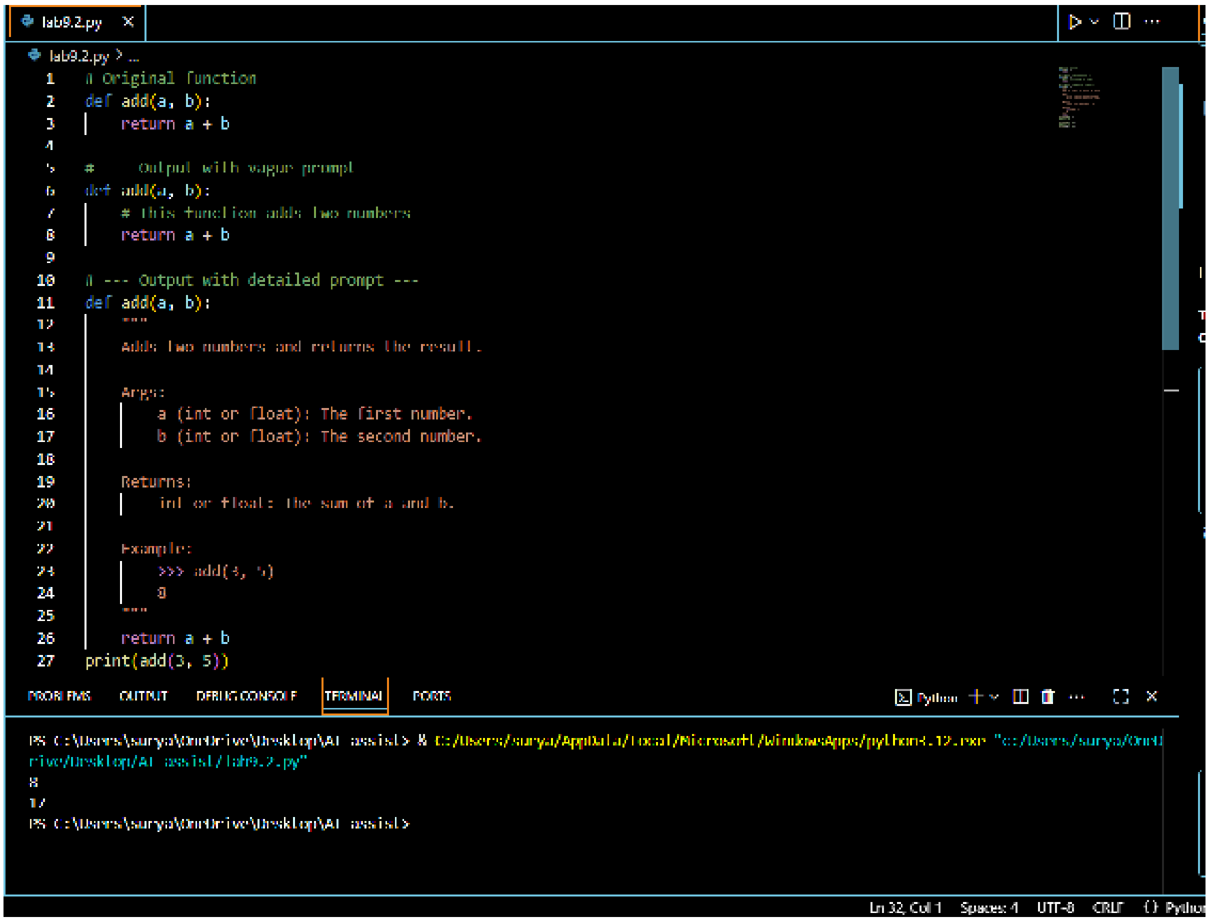
Prompt : Compare documentation output for a Python function using two approaches:

1. A vague prompt that asks for general comments.

2. A detailed prompt that asks for Google-style docstrings with parameters, return types, and examples.

Analyze the differences in clarity, accuracy, completeness, and usability of the documentation produced.

CODE :



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

#     Output with vague prompt
def add(a, b):
    # This function adds two numbers
    return a + b

# --- Output with detailed prompt ---
def add(a, b):
    """
    Adds two numbers and returns the result.

    Args:
        a (int or float): The first number.
        b (int or float): The second number.

    Returns:
        int or float: The sum of a and b.

    Example:
        >>> add(3, 5)
        8
    """
    return a + b
print(add(3, 5))
```

```
PS C:\Users\surya\OneDrive\Desktop\AI assist> & C:/Users/surya/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/surya/OneD
rive/Desktop/AI assist/lab9.2.py"
8
17
PS C:\Users\surya\OneDrive\Desktop\AI assist>
```

Observation : Using a vague prompt results in minimal, often incomplete comments that give little context, while a detailed prompt produces clear, structured, and thorough documentation. Detailed prompts improve readability, accuracy, and usefulness for other developers.