**AI ASSISTED CODING LAB 9.1**

**PRADEEP GUPTHA**

**2403A510C7**

**BATCH : 05**

Lab 9 – Documentation Generation: Automatic Documentation and  
Code Comments  
Lab Objectives  
• To use AI-assisted coding tools for generating Python  
documentation and code comments.  
• To apply zero-shot, few-shot, and context-based prompt  
engineering for documentation creation.  
• To practice generating and refining docstrings, inline comment

and module-level documentation.  
• To compare outputs from different prompting styles for quality  
analysis.

**Task Description #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.  
• Expected Output #1:  
o A Python script with all functions documented using  
correctly formatted Google-style docstrings.

**PROMPT :**  generate docstrings without providing any  
input-output examples

DOCSTRING :

def add(a: int, b: int) -> int:

    """Add two integers together.

    This function takes two integers and returns their sum.

    Args:

        a (int): The first integer.

        b (int): The second integer.

    Returns:

        int: The sum of `a` and `b`.

    Example:

        >>> add(3, 5)

        8

    """

    return a + b

def is\_even(n: int) -> bool:

    """Check whether a number is even.

    Args:

        n (int): The number to check.

    Returns:

        bool: True if `n` is even, False otherwise.

    Example:

        >>> is\_even(4)

        True

    """

    return n % 2 == 0

def greet(name: str) -> str:

    """Generate a friendly greeting message.

    Args:

        name (str): The name of the person to greet.

    Returns:

        str: A greeting message with the given name.

    Example:

        >>> greet("Alice")

        'Hello, Alice!'

    """

    return f"Hello, {name}!"

    if \_\_name\_\_ == "\_\_main\_\_":

        import doctest

        doctest.testmod()

"""

**OBSERVATION :**

 Each docstring includes:

* **Description** of the function.
* **Parameter details with type hints** (Args).
* **Return value explanation** (Returns).
* **Example usage** (Example).

 The generated documentation improves **readability** and makes the code easier to understand for other developers.

**Task Description #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.

• Expected Output #2:  
o Python code with concise, context-aware inline comments  
for complex logic blocks.

**PROMPT :** Add inline comments to python code which check a number is prime or not focus only on logical code.

**CODE :**

def is\_prime(n: int) -> bool:

    # A number less than 2 is not prime

    if n < 2:

        return False

    # Check divisibility from 2 up to sqrt(n)

    for i in range(2, int(n\*\*0.5) + 1):

        # If n is divisible by any number in this range, it is not prime

        if n % i == 0:

            return False

    # If no divisors were found, n is prime

    return True

# Taking dynamic input from user

num = int(input("Enter a number: "))

# Display result

if is\_prime(num):

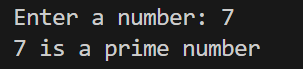
    print(f"{num} is a prime number")

else:

    print(f"{num} is not a prime number")

input – 7

OUTPUT :



OBSERVATION :

 The program checks primality by testing divisibility from **2 to √n** for efficiency.

 Numbers **less than 2** are automatically marked as non-prime.

 If a divisor is found, the function returns **False** immediately (not prime).

 If no divisor is found, the function returns **True** (prime).

Task Description #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.  
• Expected Output #3:  
o A complete, clear, and concise module-level docstring at  
the beginning of the file.

**PROMPT :** **create a module-level docstring for a Python file:**

* **Summarize the purpose, dependencies, and main functions/classes.**
* **Add a single multi-line docstring at the top of the file.**

**Python File (Before Module Docstring) :**

def is\_prime(n):

    if n < 2:

        return False

    for i in range(2, int(n\*\*0.5) + 1):

        if n % i == 0:

            return False

    return True

**AFTER MODULE LEVEL DOCSTRING (OUTPUT) :**

"""

Module: prime\_utils.py

Purpose:

    This module provides a utility function to check whether a given number is prime.

Dependencies:

    - None (uses only standard Python features)

Main Functions:

    - is\_prime(n: int) -> bool:

        Determines whether a given number is prime.

Usage:

    Import this module and call the function with appropriate arguments.

    Example:

        >>> is\_prime(7)

        True

        >>> is\_prime(10)

        False

"""

def is\_prime(n: int) -> bool:

    """Check if a number is prime.

    A prime number is a number greater than 1 that has no divisors other than 1 and itself.

    Args:

        n (int): The number to check.

    Returns:

        bool: True if `n` is prime, False otherwise.

    Example:

        >>> is\_prime(7)

        True

        >>> is\_prime(10)

        False

    """

    if n < 2:

        return False

    for i in range(2, int(n\*\*0.5) + 1):

        if n % i == 0:

            return False

    return True

# Example usage with dynamic input

if \_\_name\_\_ == "\_\_main\_\_":

    num = int(input("Enter a number to check if it is prime: "))

    if is\_prime(num):

        print(f"{num} is a prime number")

    else:

        print(f"{num} is not a prime number")

**OBSERVATION :**

 The **module-level docstring** provides a concise overview of the module, including purpose, dependencies, and usage.

 The **function-level docstring** clearly explains the function logic, parameters, return type, and example usage.

 Dynamic input allows users to test **any integer** interactively, making the program flexible.

**Task Description #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.  
• Expected Output #4:  
o Python code with comments replaced by clear,  
standardized docstrings.

**PROMPT :** transform existing inline comments into structured function docstrings following Google style. ProvidePython code containing inline comments, move relevant details from comments into function docstrings.

**Original Python Code with Inline Comments :**

def factorial(n):

    # Factorial of 0 is 1

    if n == 0:

        return 1

    # Initialize result to 1

    result = 1

    # Multiply result by each number from 1 to n

    for i in range(1, n + 1):

        result \*= i

    # Return the factorial of n

    return result

**Converted Python Code with Google-Style Docstring :**

def factorial(n: int) -> int:

    """Compute the factorial of a non-negative integer.

    The factorial of a number n (denoted as n!) is the product of all positive

    integers less than or equal to n. By definition, factorial of 0 is 1.

    Args:

        n (int): A non-negative integer whose factorial is to be computed.

    Returns:

        int: Factorial of `n`.

    Example:

        >>> factorial(5)

        120

        >>> factorial(0)

        1

    """

    if n == 0:

        return 1

    result = 1

    for i in range(1, n + 1):

        result \*= i

    return result

**OBSERAVTION :**

 Converting inline comments into a **Google-style docstring** improves readability and provides structured information about:

* Purpose of the function
* Input parameters
* Return values
* Example usage

 The docstring makes the function **self-explanatory**, reducing the need to read through the code for understanding.

**Task Description #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.

• Expected Output #5:

o Python file with updated, accurate, and standardized  
docstrings.

**PROMPT :** identify and correct inaccuracies in existing docstrings for perfect number in python with outdated or incorrect docstrings. to rewrite each docstring to match the current code behavior.

**Python Code with Incorrect Docstrings (Before Correction) :**

def is\_perfect(num: int) -> bool:

    """Check if a number is prime.

    Args:

        num (int): The number to check.

    Returns:

        bool: True if the number is prime, False otherwise.

    """

    if num < 2:

        return False

    divisors\_sum = 0

    for i in range(1, num):

        if num % i == 0:

            divisors\_sum += i

    return divisors\_sum == num

**Corrected Code with Accurate Google-Style Docstring :**

def is\_perfect(num: int) -> bool:

    """Check whether a given number is a perfect number.

    A perfect number is a positive integer that is equal to the sum of its

    proper divisors (excluding itself). For example, 6 is a perfect number

    because 1 + 2 + 3 = 6.

    Args:

        num (int): The number to check.

    Returns:

        bool: True if `num` is a perfect number, False otherwise.

    Example:

        >>> is\_perfect(6)

        True

        >>> is\_perfect(28)

        True

        >>> is\_perfect(12)

        False

    """

    if num < 2:

        return False

    divisors\_sum = 0

    for i in range(1, num):

        if num % i == 0:

            divisors\_sum += i

    return divisors\_sum == num

**OBSERVATION :**  The program correctly determines whether a number is a **perfect number** or not.

 The definition of a perfect number is clearly stated in the **docstring**, ensuring conceptual clarity.

 The function checks divisibility from 1 to num-1 and adds all proper divisors.

 If the **sum of divisors equals the number itself**, the function returns True, otherwise False.

**Task Description #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.  
• Expected Output #6:  
o A comparison table showing the results from both  
prompts with observations.

**PROMPT :** Compare documentation output from a vague prompt and a  
detailed prompt for the same Python function of factorial of a number .

**Python Function (Without Documentation) :**

def factorial(n):

    if n == 0:

        return 1

    result = 1

    for i in range(1, n + 1):

        result \*= i

    return result

Output from a *Vague Prompt :*

def factorial(n):

    # Check if n is zero

    if n == 0:

        return 1                           #vague prompt

    # Initialize result

    result = 1

    # Multiply numbers from 1 to n

    for i in range(1, n + 1):

        result \*= i

    # Return the result

    return result

Output from a Detailed Prompt :

def factorial(n: int) -> int:

    """Compute the factorial of a non-negative integer.

    The factorial of n (denoted as n!) is the product of all

    positive integers less than or equal to n. By definition,

    factorial of 0 is 1.

    Args:

        n (int): A non-negative integer whose factorial is to be computed.

    Returns:

        int: Factorial of `n`.

    Example:

        >>> factorial(5)

        120

        >>> factorial(0)

        1

    """

    if n == 0:

        return 1

    result = 1

    for i in range(1, n + 1):

        result \*= i

    return result

OBSERVATION :

 The **vague prompt** produced only inline comments that explain what the code does step by step.

 The **detailed prompt** produced **rich, structured documentation** that is more useful for developers and aligns with industry standards.