SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 9.2

Program	*: B. Tech (CSE)*
Specialization	*:*
Course Title	*: AI Assisted coding*
Course Code	*:*
Semester	*: II*
Academic Session	*: 2025-2026*
Name of Student	*: P Abhinav*
Enrollment No.	*: 2403A51L38*
Batch No.	*: 52*
Date	*:06-06-2026*

Task 1: Use AI to generate concise functional summaries for each Python function in a given script.

Prompt: Generate concise technical summaries for each Python function without describing implementation details.

Code Screenshot:

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

def is_even(n):
    return n % 2 == 0

def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Code Output:

```
def add(a, b):
    """Returns the sum of two numeric values."""
    return a + b

def is_even(n):
    """Determines whether a given number is even."""
    return n % 2 == 0

def factorial(n):
    """Computes the factorial of a non-negative integer."""
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Code Explanation:

- **add()** – Computes the sum of two numeric values.
- **is_even()** – Determines whether a given number is even.
- **factorial()** – Calculates the factorial of a non-negative integer.

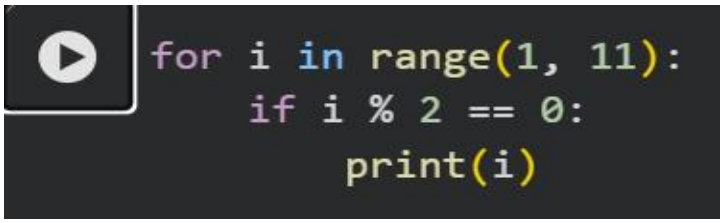
The purpose of this task is to demonstrate how AI can automatically generate concise functional documentation by analyzing Python functions. Instead of describing how the code executes internally, the AI focuses only on the intent of each function. This approach is commonly used in professional software documentation where clarity and brevity are prioritized to help developers quickly understand what a function does.

Each summary provides a high-level description of functionality without exposing logic such as loops or recursion. For example, the add() function is described simply as performing addition, while factorial() is defined as computing factorial values rather than mentioning recursive calls. This ensures the documentation remains readable, scalable, and suitable for large codebases where implementation may change but function purpose remains constant.

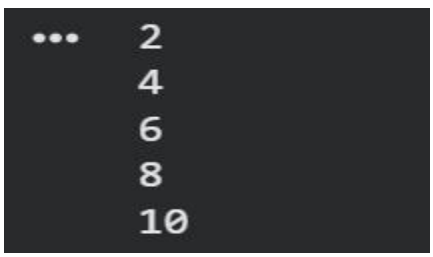
Task 2: Use AI to document the logic behind conditional statements and loops in a Python program.

Prompt: Explain the decision-making logic and loop behavior only. Skip basic syntax explanations.

Code Screenshot:



Output Screenshot:



Explanation:

The loop repeatedly processes numbers from 1 through 10. During each iteration, a condition evaluates whether the current number is divisible by two without leaving a remainder. If the condition is satisfied, the number is considered even and is displayed as output.

This decision structure filters values based on parity, allowing only even numbers to pass through the condition. As a result, the program selectively prints even numbers while ignoring odd ones, demonstrating how conditional checks can control data flow inside iterative processes.

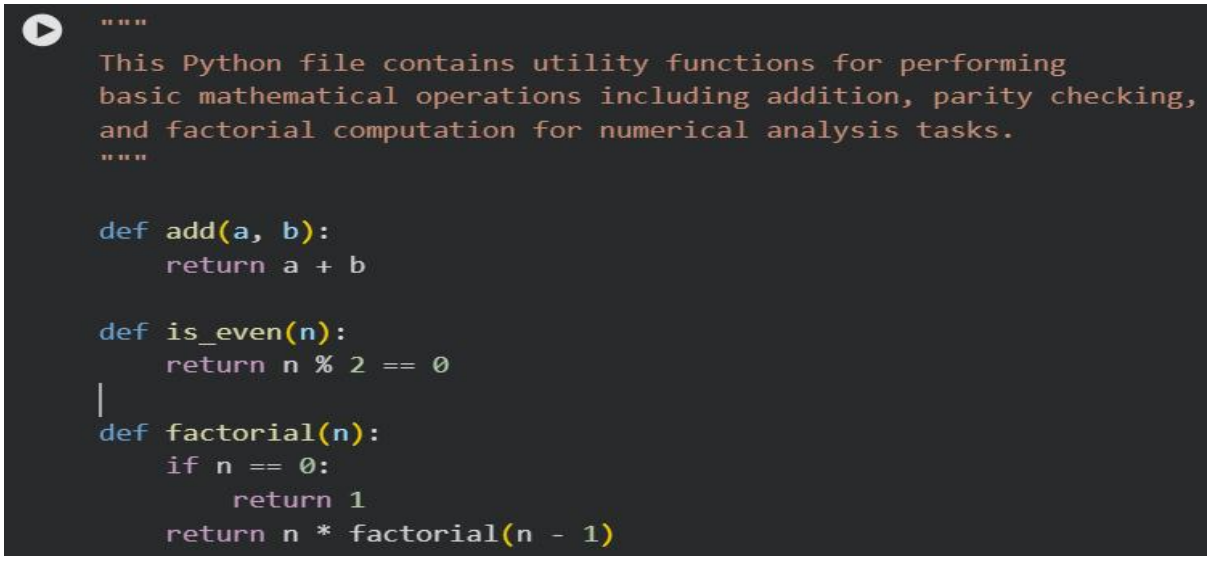
This task highlights how AI can interpret logical behavior in Python programs rather than explaining programming syntax. The loop functions as a repetition mechanism that moves sequentially through a range of values. At each step, the conditional statement acts as a gatekeeper that decides whether a specific value should be processed further.

By applying a mathematical condition to each number, the program effectively filters unwanted data. This logic-driven approach is fundamental in programming for tasks such as validation, searching, and selective processing. AI-generated documentation like this helps developers understand why certain values are chosen without overwhelming them with code-level details.

Task 3: Documentation – File-Level Overview)

Prompt: Generate a high-level overview describing the purpose and functionality of the entire Python file.

Code Screenshot:



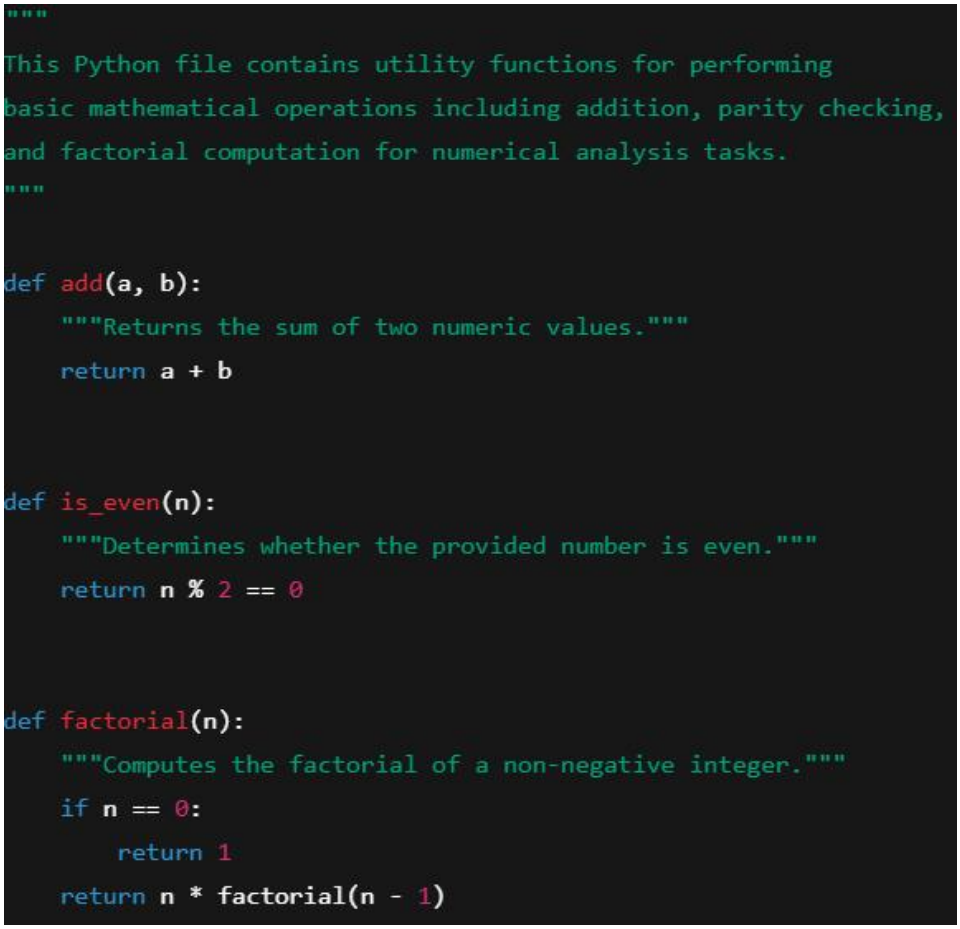
```
"""
This Python file contains utility functions for performing
basic mathematical operations including addition, parity checking,
and factorial computation for numerical analysis tasks.
"""

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

def is_even(n):
    return n % 2 == 0

def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Output Screenshot:



```
"""
This Python file contains utility functions for performing
basic mathematical operations including addition, parity checking,
and factorial computation for numerical analysis tasks.
"""

def add(a, b):
    """Returns the sum of two numeric values."""
    return a + b


def is_even(n):
    """Determines whether the provided number is even."""
    return n % 2 == 0


def factorial(n):
    """Computes the factorial of a non-negative integer."""
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Explanation:

A file-level overview provides readers with an immediate understanding of what the entire script is designed to accomplish. Instead of focusing on individual functions, it explains the broader purpose of the file and the types of operations it supports. This is especially valuable in large projects where developers may interact with hundreds of scripts.

By clearly stating that the file handles mathematical utilities such as addition, even number detection, and factorial calculation, users can quickly determine whether the file is relevant to their needs. This improves maintainability, enhances collaboration among developers, and reduces time spent searching through unfamiliar code.

Task 4: Use AI to improve clarity and consistency of existing documentation in Python code.

Prompt: Improve unclear comments while keeping the technical meaning unchanged.

Code Screenshot:

```
# check number
def is_even(n):
    return n % 2 == 0
```

Code Output:

```
# Determines whether the provided number is even
def is_even(n):
    return n % 2 == 0
#Example 4 is even as its divisible by '0'
```

Explanation:

The original comment is vague and does not clearly describe what kind of checking is being performed. Such unclear documentation can confuse developers, especially in complex systems where many types of validations may exist. AI refinement focuses on making the comment precise and technically meaningful.

The improved version explicitly states that the function determines evenness, which directly aligns with the function's purpose. This enhances readability and ensures consistency with professional coding standards. Clear documentation reduces misunderstandings, improves debugging efficiency, and supports long-term code maintenance.

Task 5: (Documentation – Prompt Detail Impact Study)

Prompt: Write a clear technical summary describing the function's purpose, expected behavior, and typical usage without including implementation details.

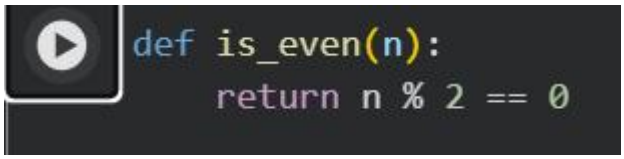
Explanation:

Aspect	Brief Prompt	Detailed Prompt
Completeness	Short basic purpose	Includes purpose and behavior
Clarity	Minimal	Very clear and descriptive
Accuracy	Correct	Correct and informative

This task demonstrates how the quality of AI-generated documentation is strongly influenced by the level of prompt detail. A brief prompt typically results in short and sometimes overly simple descriptions that may lack context. While still technically accurate, such documentation may not fully convey how or when a function should be used.

In contrast, a detailed prompt encourages the AI to produce richer and more structured explanations. It covers functional intent, behavior, and potential usage scenarios, making the documentation more helpful for real-world development. This highlights the importance of prompt engineering when using AI for professional software documentation.

Code Screenshot:



Code Output:

```
# Determines whether the provided number is even
def is_even(n):
    return n % 2 == 0
#Example 4 is even as its divisible by '0'
```

Conclusion:

This study demonstrated how artificial intelligence can effectively generate and enhance software documentation across multiple levels, including function summaries, logical explanations, file-level overviews, and refinement of existing comments. By applying AI to real Python code examples, the tasks showed that concise functional descriptions improve readability, while logical documentation clarifies decision-making processes within loops and conditions. The inclusion of file-level overviews further supports maintainability by offering immediate contextual understanding of program purpose.

Additionally, the prompt detail impact analysis highlighted the critical role of precise instructions in obtaining high-quality AI-generated documentation. Detailed prompts consistently produced clearer, more complete, and technically accurate explanations compared to brief prompts. Overall, the results confirm that AI-assisted documentation can significantly reduce manual effort, enhance code clarity, and support scalable software development when guided by well-structured prompts.