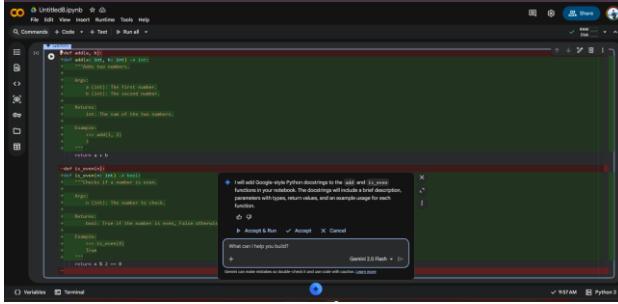


SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING																			
Program Name: B. Tech		Assignment Type: Lab																			
Course Coordinator Name		Dr. Rishabh Mittal																			
Instructor(s)Name		<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>Mr. S Naresh Kumar</td></tr> <tr><td>Ms. B. Swathi</td></tr> <tr><td>Dr. Sasanko Shekhar Gantayat</td></tr> <tr><td>Mr. Md Sallauddin</td></tr> <tr><td>Dr. Mathivanan</td></tr> <tr><td>Mr. Y Srikanth</td></tr> <tr><td>Ms. N Shilpa</td></tr> <tr><td>Dr. Rishabh Mittal (Coordinator)</td></tr> <tr><td>Dr. R. Prashant Kumar</td></tr> <tr><td>Mr. Ankushavali MD</td></tr> <tr><td>Mr. B Viswanath</td></tr> <tr><td>Ms. Sujitha Reddy</td></tr> <tr><td>Ms. A. Anitha</td></tr> <tr><td>Ms. M.Madhuri</td></tr> <tr><td>Ms. Katherashala Swetha</td></tr> <tr><td>Ms. Velpula sumalatha</td></tr> <tr><td>Mr. Bingi Raju</td></tr> <tr><td>Mr. G. Kranthi</td></tr> </table>		Mr. S Naresh Kumar	Ms. B. Swathi	Dr. Sasanko Shekhar Gantayat	Mr. Md Sallauddin	Dr. Mathivanan	Mr. Y Srikanth	Ms. N Shilpa	Dr. Rishabh Mittal (Coordinator)	Dr. R. Prashant Kumar	Mr. Ankushavali MD	Mr. B Viswanath	Ms. Sujitha Reddy	Ms. A. Anitha	Ms. M.Madhuri	Ms. Katherashala Swetha	Ms. Velpula sumalatha	Mr. Bingi Raju	Mr. G. Kranthi
Mr. S Naresh Kumar																					
Ms. B. Swathi																					
Dr. Sasanko Shekhar Gantayat																					
Mr. Md Sallauddin																					
Dr. Mathivanan																					
Mr. Y Srikanth																					
Ms. N Shilpa																					
Dr. Rishabh Mittal (Coordinator)																					
Dr. R. Prashant Kumar																					
Mr. Ankushavali MD																					
Mr. B Viswanath																					
Ms. Sujitha Reddy																					
Ms. A. Anitha																					
Ms. M.Madhuri																					
Ms. Katherashala Swetha																					
Ms. Velpula sumalatha																					
Mr. Bingi Raju																					
Mr. G. Kranthi																					
Course Code	23CS002PC304	Course Title	AI Assisted Coding																		
Year/Sem	III/I	Regulation	R23																		
Date and Day of Assignment	Week 5 - Thursday	Time(s)	23CSBTB01 To 23CSBTB52																		
Duration	2 Hours	Applicable to Batches	All Batches																		
AssignmentNumber: 9.4 (Present assignment number)/24(Total number of assignments)																					

Q . N o	Question	<i>Expe cted Time to</i>
------------------	-----------------	--------------------------------------

.		<i>complete</i>
1	<p>Lab 9 – Documentation Generation: Automatic Documentation and Code Comments</p> <p>Lab Objectives</p> <ul style="list-style-type: none"> • 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 comments, and module-level documentation. • To compare outputs from different prompting styles for quality analysis. <p>Lab Outcomes</p> <ul style="list-style-type: none"> • Generate structured code documentation using AI tools • Apply appropriate documentation styles to different code contexts • Improve code readability through selective commenting • Convert informal developer comments into professional documentation • Analyze and refine AI-generated documentation 	Week 5
	<p>Task 1: Auto-Generating Function Documentation in a Shared Codebase</p> <p>Scenario</p> <p>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.</p> <p>Task Description</p> <p>You are given a Python script containing multiple functions without any docstrings.</p> <p>Using an AI-assisted coding tool:</p> <ul style="list-style-type: none"> • Ask the AI to automatically generate Google-style function docstrings for each function • Each docstring should include: <ul style="list-style-type: none"> ◦ A brief description of the function ◦ Parameters with data types ◦ Return values ◦ At least one example usage (if applicable) <p>Experiment with different prompting styles (zero-shot or context-based) to observe quality differences.</p> <p>Expected Outcome</p>	

	<ul style="list-style-type: none"> • A Python script with well-structured Google-style docstrings • Docstrings that clearly explain function behavior and usage • Improved readability and usability of the codebase 	
	<p>Prompt</p> <p>Generate Google-style Python docstrings for each function.</p> <p>Include:</p> <ul style="list-style-type: none"> - Brief description - Parameters with types - Return values - One example usage 	
	 <p>The AI automatically generated Google-style docstrings for the following functions:</p> <pre> # Function to add two numbers def add(a, b): """Add two numbers. Args: a (int): The first number. b (int): The second number. Returns: int: The sum of the two numbers. Examples: >>> add(3, 5) 8 """ return a + b def is_even(n): """Check if a number is even. Args: n (int): The number to check. Returns: bool: True if the number is even, False otherwise. Examples: >>> is_even(4) True """ return n % 2 == 0 </pre>	<ul style="list-style-type: none"> • The AI automatically generated Google-style docstrings • Parameters, return types, and example usage are included • Improves readability and onboarding for new developers • Zero-shot prompt gave good quality documentation

- 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

Prompt

Add inline comments only for complex or non-obvious logic.

Do not comment on simple Python syntax.

Explain why the logic is used.

Sample

```
def fibonacci(n):
    a, b = 0, 1
    result = []
    for _ in range(n):
        result.append(a)
        a, b = b, a + b
    return result
```

The screenshot shows the Gemini AI interface with a code editor and a sidebar. The code editor contains a Python function definition for `fibonacci`. The sidebar has a section titled "I've added inline comments to the `fibonacci` function in cell 1.2. These comments explain the initialization of the sequence and the core logic for generating the next numbers." Below this, there is a "Working..." status message and a "What can I help you build?" input field.

```
def fibonacci(n):
    # Initialize the first two Fibonacci numbers
    # 'a' holds the current Fibonacci number, 'b' holds the next one
    a, b = 0, 1
    result = []
    for _ in range(n):
        result.append(a)
        # Update 'a' to be the previous 'b' and 'b' to be the sum of the previous 'a' and 'b'
        a, b = b, a + b
    return result
```

- Comments explain **why tuple assignment is used**
- No unnecessary comments on simple syntax
- Improves maintainability and understanding

	<ul style="list-style-type: none"> • Code remains clean and readable 	
Task 3: Generating Module-Level Documentation for a Python Package		
	<p>Scenario</p> <p>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.</p> <p>Task Description</p> <p>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:</p> <ul style="list-style-type: none"> • 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 <p>Focus on clarity and professional tone.</p> <p>Expected Outcome</p> <ul style="list-style-type: none"> • 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 	

The screenshot shows a Jupyter Notebook interface with a Python code cell containing the following content:

```

t1
'''This module provides a collection of utility functions and a basic Calculator class.

Purpose:
    To offer simple mathematical operations and sequence generation functionalities.

Dependencies:
    - math (standard library, for square root calculation)

Key Functions/Classes:
    - add(a: int, b: int) -> int: Adds two numbers.
    - is_even(n: int) -> bool: Checks if a number is even.
    - fibonacci(n: int) -> list: Generates the first 'n' Fibonacci numbers.
    - square_root(n: float) -> float: Calculates the square root of a number.
    - Calculator: A class for performing common arithmetic methods.
        - multiply(a: Union[int, float], b: Union[int, float]) -> Union[int, float]: Multiplies two numbers.

Example Usage:
>>> # Using standard functions
>>> print(add(5, 3))
8
>>> print(is_even(4))
True
>>> print(fibonacci(5))
[0, 1, 1, 2, 3]
>>> print(square_root(9))
3.0
...
>>> # Using the Calculator class
>>> calc = Calculator()
>>> print(calc.multiply(6, 7))
42
...

```

A tooltip window titled "module - Dependencies - Key functions/classes - Example usage" is open, providing a detailed description of the module's purpose, dependencies, key functions, and example usage.

- Module purpose and usage is clear at first glance
- Dependencies are documented
- Helps developers understand the package instantly
- Suitable for GitHub or internal repositories

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.

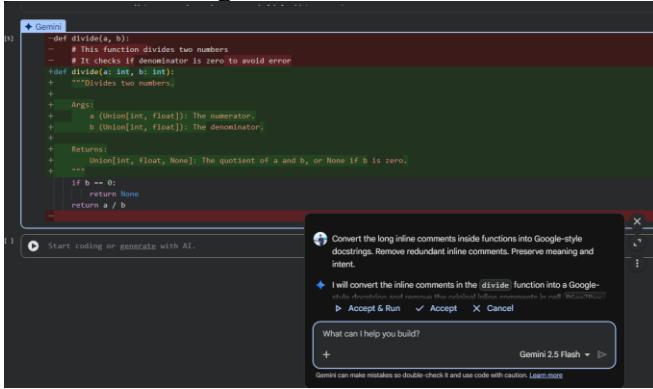
Task Description

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

	<ul style="list-style-type: none"> • Functions with clean, standardized docstrings • Reduced clutter inside function bodies • Improved consistency across the codebase 	
	<p>Prompt</p> <p>Convert the long inline comments inside functions into Google-style docstrings.</p> <p>Remove redundant inline comments.</p> <p>Preserve meaning and intent.</p> 	
	<ul style="list-style-type: none"> • Inline comments converted into clean docstring • Code body is now clutter-free • Documentation is standardized and professional • Improves consistency across codebase 	

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:
 - Functions
 - 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.

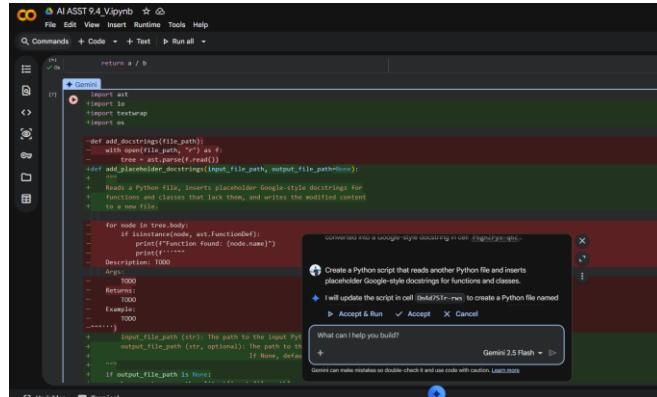
Note: The goal is **documentation scaffolding**, not perfect documentation.

Expected Outcome

- A working Python script that processes another .py file
 - Automatically inserted placeholder docstrings
 - Clear demonstration of how AI can assist in documentation automation

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Create a Python script that reads another Python file and inserts placeholder Google-style docstrings for functions and classes.



A screenshot of a Jupyter Notebook cell demonstrating the use of AI to generate placeholder Google-style docstrings. The cell contains Python code with placeholder docstrings (e.g., `Args:`, `Returns:`, `Todo`). A tooltip from an AI tool (Gemini 2.5) provides instructions for generating docstrings for functions and classes, and offers to update the script in the cell.

```
prune(1, "my_file.py")
...
... Content of sample_modified.py ...
import os

def my_function(a, b):
    """Description: TODO
    Args:
        TODO
    Returns:
        TODO
    Example:
        TODO
    """
    pass

class MyClass:
    """Description: TODO
    Methods:
    ...
    def __init__(self, *args):
        """Description: TODO
        Args:
            TODO
        Returns:
            TODO
    """
    pass
```

Create a Python script that reads another Python file and inserts placeholder Google-style docstrings for functions and classes.
I will update the script in cell (and it'll run) to create a Python file named
Show me the content of 'sample_modified.py' Explain the 'add_placeholder_docstrings' function
What can I help you build? Gemini 2.5 Flash Gemini 2.5 Flash