

# **ASSIGNMENT-9.4**

2303A51739

Gangula Navyasri

BATCH-25

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:
  - A brief description of the function
  - Parameters with data types
  - Return values
  - 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

The screenshot shows a Python code editor interface with the following details:

- File Explorer:** Shows files like 'add.py', 'age\_classifier.py', 'Ai ass 7.3.docx', 'Ai ass 7.3.pdf', 'Ai assis Ass 1.pdf', 'app.py', 'ass 3.4 ai.pdf', 'ass 4.4.pdf', 'ass 4.4.py', 'ass 6.3.docx', 'ass 6.3.pdf', 'ass 6.3.py', 'ass 7.3.py', 'ass 8.4 Ai.docx', 'ass 8.4 Ai.pdf', 'Ass 9.4.py' (selected), 'lab 4.3 word.pdf', 'lab 4.3.py', 'lab ass 3.4.py', 'lab assignment 3.3.pdf', 'lab assignment 1-4.pdf', 'lab assignment 2-3.pdf', 'lab assignment54 ai.pdf', 'leap\_year.py', 'logger.py', 'multiples.py', 'student.py', 'sum\_to\_n.py'.
- Code Editor:** Displays the content of 'Ass 9.4.py'. The code includes:

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

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

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

def factorial(n):
    if n < 0:
        raise ValueError("Negative numbers not allowed")
    if n == 0:
        return 1
    return n * factorial(n - 1)

print(add(2, 3))
print(divide(10, 2))
print(is_even(4))
print(factorial(5))
```
- Terminal:** Shows the command line output of running the script:

```
PS C:\Users\parva\OneDrive\Desktop\AI Asstd> & C:/Users/parva/AppData/Local/Programs/Python/Python311/python.exe "C:/Users/parva/OneDrive/Desktop/AI Asstd/Ass 9.4.py"
5
5.0
True
120
PS C:\Users\parva\OneDrive\Desktop\AI Asstd>
```
- Status Bar:** Shows file paths, line numbers (Ln 4, Col 1), and other system information.

## 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

```
File Edit Selection View Go Run ... < → 🔍 AI Assisted
EXPLORER ... Ass 9.4.py ✘
AI ASSISTED
add.py
age_classifier.py
AI ass 7.3.docx
AI ass 7.3.pdf
AI ass Ass 1.pdf
apps.py
ass 3.4.pdf
ass 4.4.pdf
ass 4.4.py
ass 6.3.docx
ass 6.3.pdf
ass 6.3.py
ass 7.3.py
ass 8.4.pdf
ass 8.4.pdf
Ass 9.4.py
lab 4.3.word.pdf
lab 4.3.py
lab ass 3.4.py
lab assignment 3.3.pdf
lab assignment 1.4.pdf
lab assignment 2.3.pdf
lab assignment 4.pdf
leap_year.py
logger.py
multiples.py
student.py
sum_to_n.py

Ass 9.4.py > 🐍 bubble_sort
def fibonaccii():
    1
    if n <= 0:
        2
            return []
    if n == 1:
        3
            return [0]
    4
    sequence = [0, 1]
    5
    for i in range(2, n):
        6
        next_value = sequence[i - 1] + sequence[i - 2]
        7
        sequence.append(next_value)
    8
    return sequence
    9
def binary_search(arr, target):
    10
    left = 0
    11
    right = len(arr) - 1
    12
    while left <= right:
        13
        mid = (left + right) // 2
        14
        if arr[mid] == target:
            15
                return mid
        16
        elif arr[mid] < target:
            17
                left = mid + 1
        18
        else:
            19
                right = mid - 1
    20
    return -1
    21
def bubble_sort(arr):
    22
    n = len(arr)
    23
    for i in range(n):
        24
            swapped = False
        25
            for j in range(0, n - i - 1):
                26
                    if arr[j] > arr[j + 1]:
                        27
                            arr[j], arr[j + 1] = arr[j + 1], arr[j]
                        28
                            swapped = True
        29
    30
    if swapped:
        31
    32
    33
    34
    35
    36
    37
    38
    39
    40
    41
    42

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

9.4.py"
5
5.0
5.0E
129
PS C:\Users\parva\OneDrive\Desktop\AI Assisted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assisted/Ass 9.4.py"
Fibonacci: [0, 1, 1, 2, 3, 5, 8]
Sorted List: [1, 2, 3, 4, 5, 6, 7, 8, 9]
PS C:\Users\parva\OneDrive\Desktop\AI Assisted>
PS C:\Users\parva\OneDrive\Desktop\AI Assisted> []

20°C Sunny
```

The screenshot shows a code editor interface with the following details:

- File Explorer (Left):** Shows a folder named "AI Assted" containing various files including "add.py", "age\_classifier.py", "Ai ass 7.3.docx", "Ai assis Ass 1.pdf", "app.py", "ass 3.4 ai.pdf", "ass 4.4.pdf", "ass 4.4.py", "ass 6.3.docx", "ass 6.3.pdf", "ass 6.3.py", "ass 7.3.py", "ass 8.4 Ai.docx", "ass 8.4 Ai.pdf", "Ass 9.4.py", "lab 4.3 word.pdf", "lab 4.3.py", "lab ass 3.4.py", "lab assignment 3.3.pdf", "lab assignment-1.4.pdf", "lab assignment-2.3.pdf", "lab assignment54 ai.pdf", "leap\_year.py", "logger.py", "multiples.py", "student.py", and "sum\_to\_n.py".
- Code Editor (Center):** Displays the content of "Ass 9.4.py".

```
def bubble_sort(arr):
    n = len(arr)

    for i in range(n):
        swapped = False

        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break

    return arr

if __name__ == "__main__":
    print("Fibonacci:", fibonacci(7))

    sorted_list = bubble_sort([5, 2, 9, 1, 3])
    print("Sorted List:", sorted_list)

    print("Binary Search (find 9):", binary_search(sorted_list, 9))
```
- Terminal (Bottom):** Shows the command-line output of running the script.

```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/AI Assted/Ass 9.4.py"
9.4.py"
5
5.0
True
120
PS C:\Users\parva\OneDrive\Desktop\AI Assted> Fibonacci: [0, 1, 1, 2, 3, 5, 8]
Sorted List: [1, 2, 3, 5, 9]
Binary Search (find 9): 4
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

### 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

```
Ass 9.4.py
def add(a: int, b: int) -> int:
    return a + b

def divide(a: float, b: float) -> float:
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

def is_even(number: int) -> bool:
    return number % 2 == 0

def factorial(n: int) -> int:
    if n < 0:
        raise ValueError("Negative numbers not allowed")
    if n == 0:
        return 1
    return n * factorial(n - 1)

if __name__ == "__main__":
    print("Addition:", add(5, 3))
    print("Division:", divide(10, 2))
    print("Is Even (7):", is_even(7))
    print("Factorial (4):", factorial(4))

Fibonacci: [0, 1, 2, 3, 5, 8]
Sorted List: [1, 2, 3, 5, 9]
Binary Search (Find 9): 4
PS C:\Users\parva\OneDrive\Desktop\AI Assteds> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assteds/Ass 9.4.py"
Addition: 8
Division: 5.0
Is Even (7): False
Factorial (4): 24
PS C:\Users\parva\OneDrive\Desktop\AI Assteds>
```

### 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

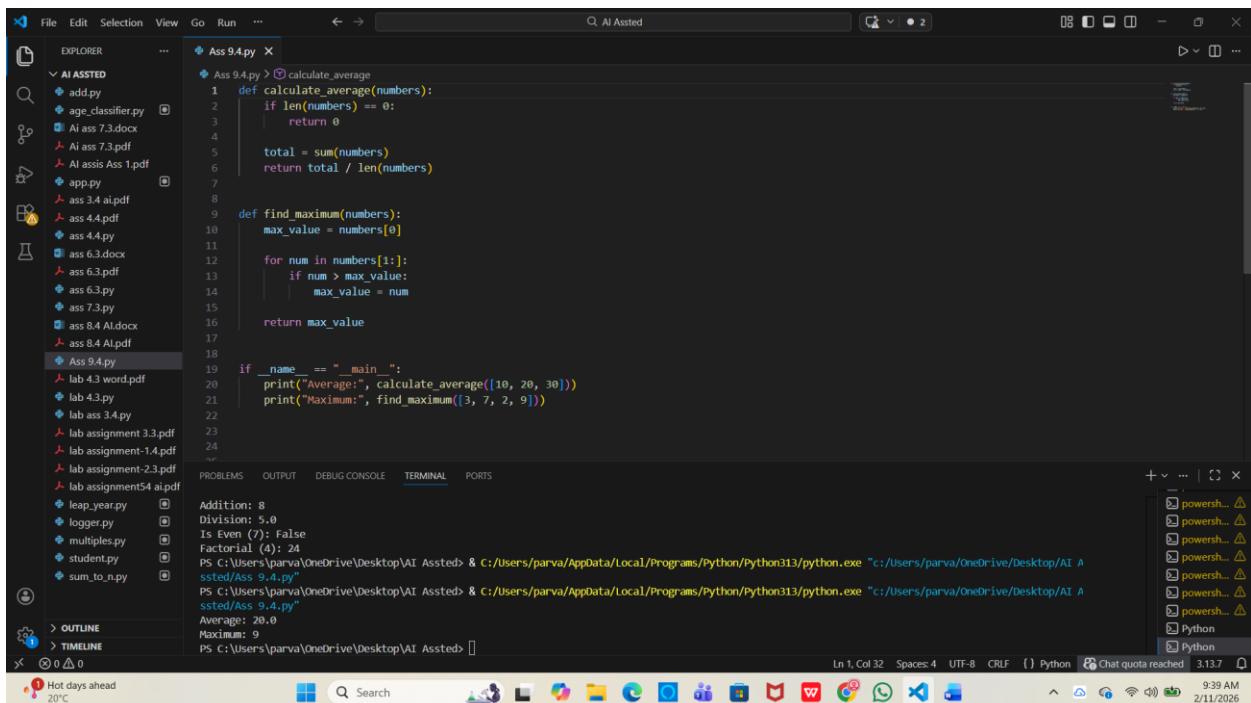
You are given a Python script where functions contain detailed inline comments explaining their logic.

## 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



## 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

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar displays a file tree with various Python files, PDFs, and other documents. The main editor area contains a Python script named `Ass 9.4.py`. The script defines a function `generate_class_docstring` which generates a docstring for a class based on its name and arguments. It also includes functions for inserting docstrings into files and checking if a file exists. The bottom status bar shows the command line output of running the script, indicating successful documentation generation and a new file creation. The bottom right corner shows the Python icon in the taskbar.

```
File Edit Selection View Go Run ... ← → Q AI Assted

EXPLORER Ass 9.4.py
AI ASSTED
adtpy
age_classifier.py
Ai ass 7.3.docx
Ai ass 7.3.pdf
Ai ass Ass 1.pdf
app.py
ass 3.4.pdf
ass 4.4.pdf
ass 4.4py
ass 6.3.pdf
ass 6.3.docx
ass 6.3.py
ass 7.3.py
ass 8.4 Al.docx
ass 8.4 Al.pdf
Ass 9.4_documented.py
Ass 9.4.py
lab 4.3 word.pdf
lab 4.3.py
lab ass 3.4.py
lab assignment 3.3.pdf
lab assignment 1.4.pdf
lab assignment 2.3.pdf
lab assignment54 ai.pdf
leap_year.py
logger.py
multiples.py
student.py
sum_to_n.py

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\parva\OneDrive\Desktop\AI Assted & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/Ass 9.4.py"
Documentation generated successfully!
New file created: Ass 9.4_documented.py
PS C:\Users\parva\OneDrive\Desktop\AI Assted> [20°C Sunny] Search
```

The screenshot shows a Python code editor interface with the following details:

- File Explorer:** Shows files in the current directory, including `Ass 9.4.py`, `Ass 9.4.py` (marked as modified), `add.py`, `age_classifier.py`, `ass 7.3.docx`, `Ass 7.3.pdf`, `Ass Ass Ass 1.pdf`, `app.py`, `ass 3.4 ai.pdf`, `ass 4.4.pdf`, `ass 4.4.py`, `ass 6.3.docx`, `ass 6.3.pdf`, `ass 6.3.py`, `ass 7.3.py`, `ass 8.4 AI.docx`, `ass 8.4 AI.pdf`, `Ass 9.4_documented.py`, and `Ass 9.4.py`.
- Code Editor:** The main pane displays the `Ass 9.4.py` script. The script uses the `ast` module to walk through an abstract syntax tree (AST) and insert class docstrings into the code. It handles `FunctionDef` and `ClassDef` nodes, generating docstrings based on function arguments and class names. It also handles `multiple.py` and `student.py` files by writing them to a new file named `Ass 9.4_documented.py`. The script concludes with a check for the `__name__` variable and an `if __name__ == "__main__":` block.
- Terminal:** The bottom terminal window shows the command line output of running the script:

```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "C:/Users/parva/OneDrive/Desktop/AI Assted/Ass 9.4.py"
[Documentation generated successfully!]
New file created: Ass 9.4_documented.py
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. Status indicators show 30 columns, 4 rows, and 2 tabs open. A Chat quota reached message is visible.
- Icons:** Standard icons for file operations like Open, Save, Copy, Paste, etc.