

# 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.

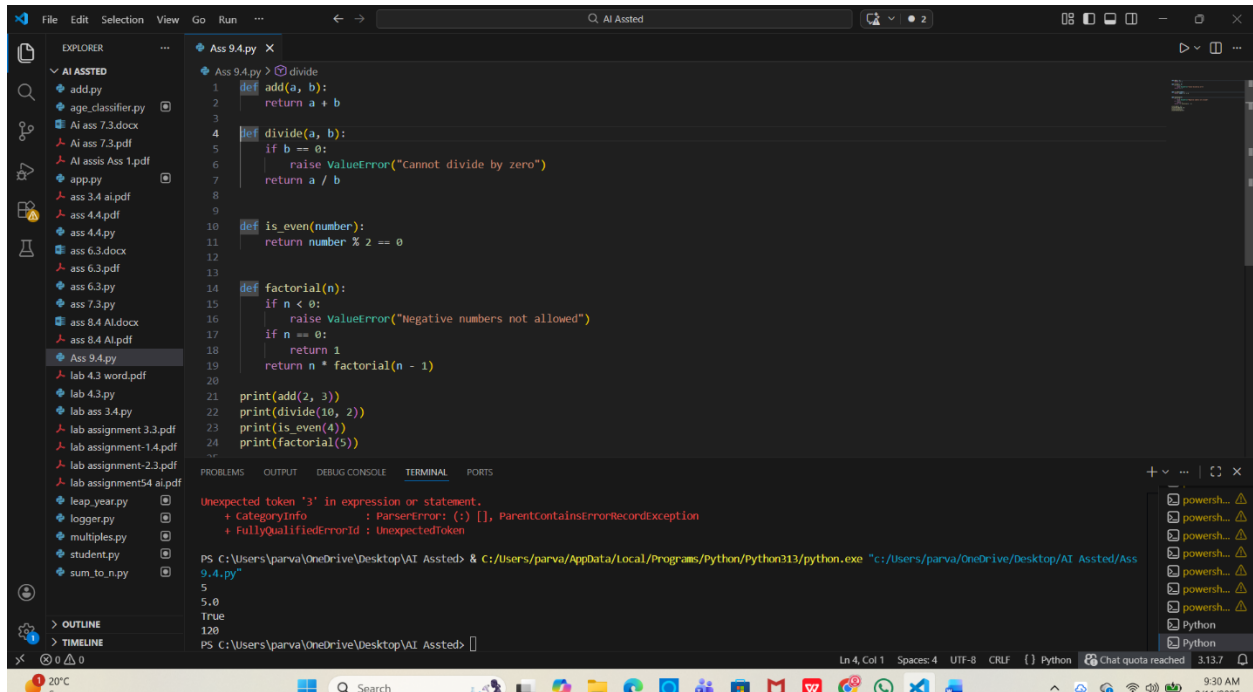
Using an AI-assisted coding tool:

- Ask the AI to automatically generate Google-style function docstrings for each function
- Each docstring should include:
  - o A brief description of the function
  - o Parameters with data types
  - o Return values
  - o 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



## 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:

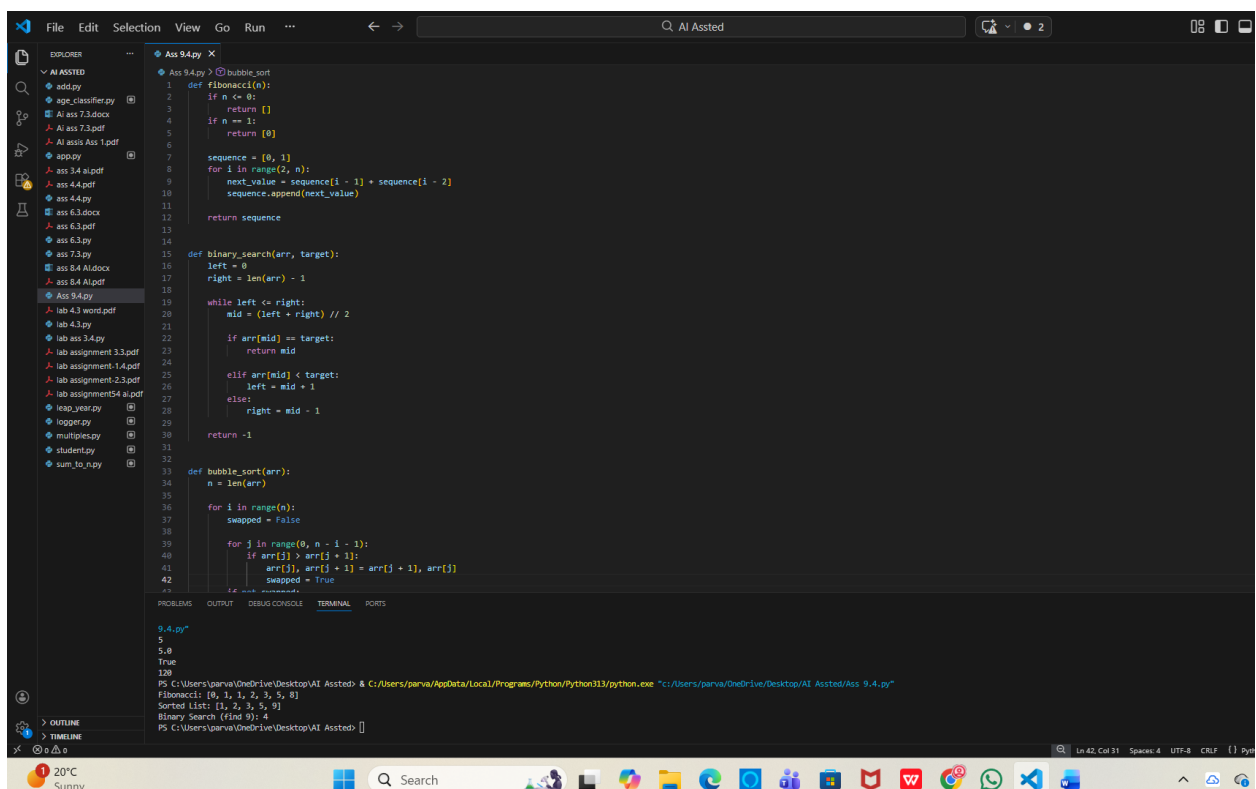
- Automatically insert inline comments only for complex or non-obvious logic

- 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

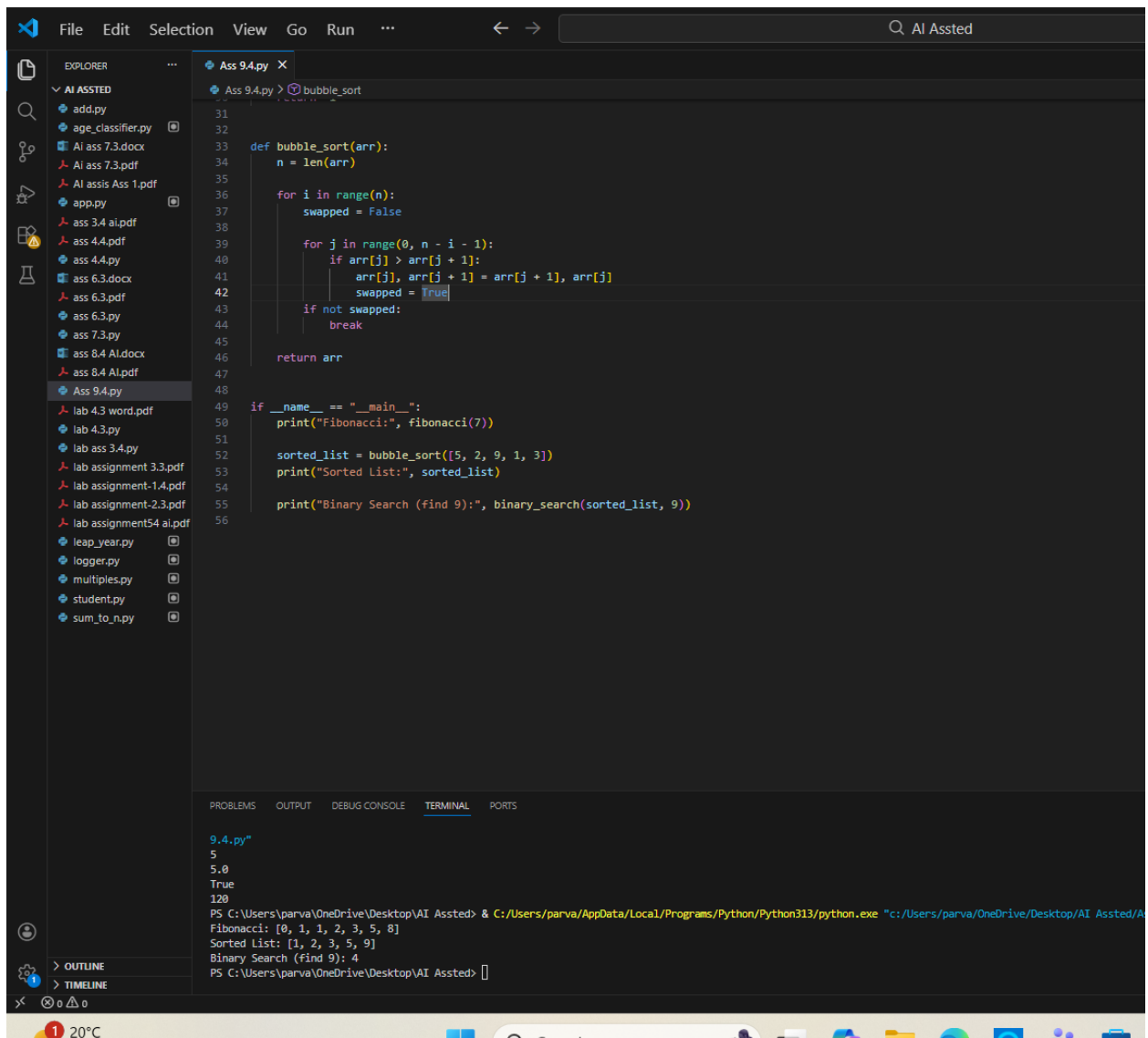


The screenshot shows a VS Code editor with a Python script named 'Ass 9.4.py'. The script contains three functions: 'fibonacci(n)', 'binary\_search(arr, target)', and 'bubble\_sort(arr)'. The 'fibonacci' function has inline comments explaining the logic. The 'binary\_search' function has inline comments explaining the logic. The 'bubble\_sort' function has inline comments explaining the logic. The terminal output shows the execution of the script, displaying the Fibonacci sequence, the sorted list, and the result of the binary search.

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

Terminal Output:

```
9.4.py  
5  
5.0  
True  
120  
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"  
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

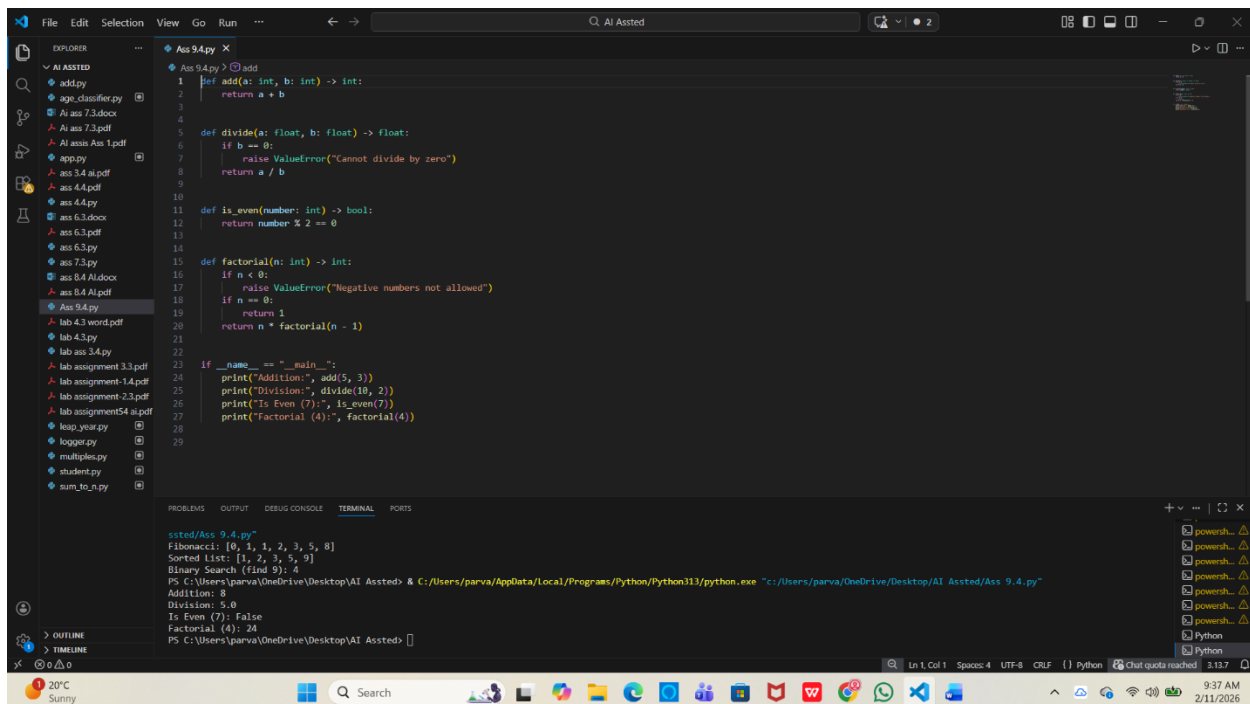
automatically generate a module-level docstring at the top of the file that includes:

- 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



```
File Edit Selection View Go Run ...
AI ASSTED
add.py
age_classifier.py
AI ass 7.3.docx
AI 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 2.3.pdf
lab assignment 4.4.pdf
lab assignment 2.3.pdf
lab assignment 5.4 AI.pdf
leap_year.py
logger.py
multiplies.py
student.py
sum_to_n.py

Ass 9.4.py X
1 def add(a: int, b: int) -> int:
2     return a + b
3
4
5 def divide(a: float, b: float) -> float:
6     if b == 0:
7         raise ValueError("Cannot divide by zero")
8     return a / b
9
10
11 def is_even(number: int) -> bool:
12     return number % 2 == 0
13
14
15 def factorial(n: int) -> int:
16     if n < 0:
17         raise ValueError("Negative numbers not allowed")
18     if n == 0:
19         return 1
20     return n * factorial(n - 1)
21
22
23 if __name__ == "__main__":
24     print("Addition:", add(5, 3))
25     print("Division:", divide(10, 2))
26     print("Is Even (7):", is_even(7))
27     print("Factorial (4):", factorial(4))
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
sstd/Ass 9.4.py
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> & C:\Users\parva\AppData\Local\Programs\Python\Python311\python.exe "c:\Users\parva\OneDrive\Desktop\AI Assted\Ass 9.4.py"
Addition: 8
Division: 5.0
Is Even (7): False
Factorial (4): 24
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

## 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

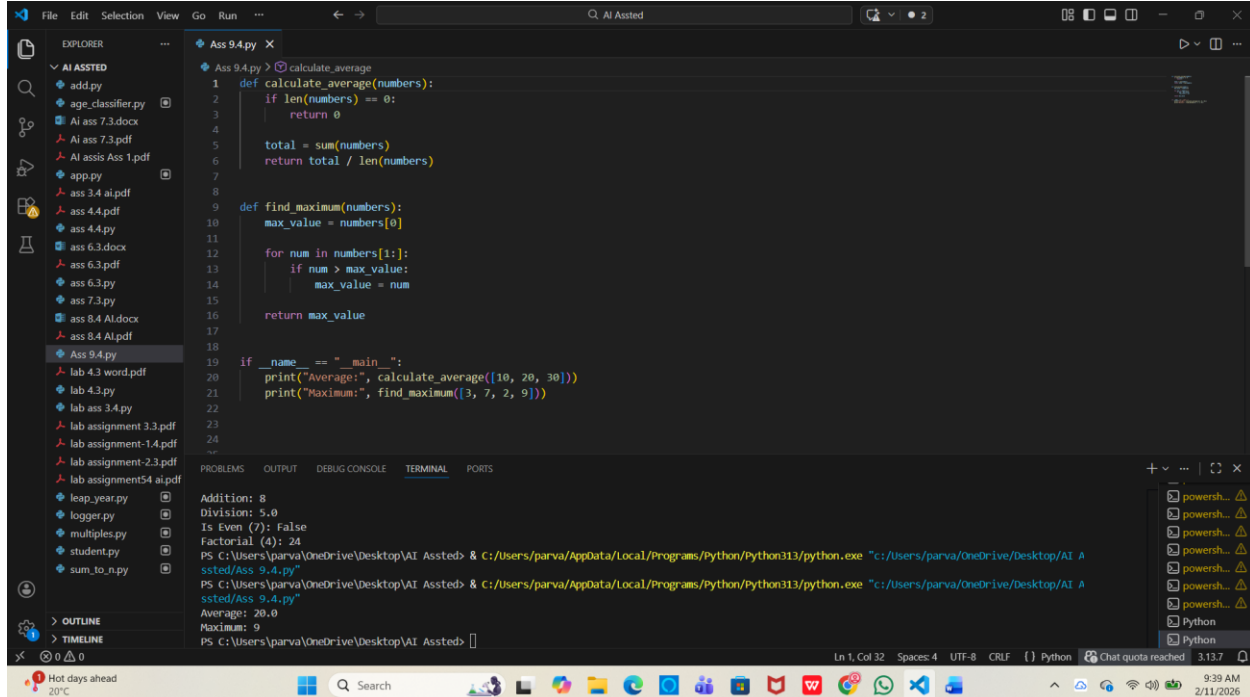
You are given a Python script where functions contain detailed inline comments explaining their logic.

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



The screenshot shows a VS Code editor with a Python script named 'Ass 9.4.py'. The script contains two functions: 'calculate\_average' and 'find\_maximum'. The 'calculate\_average' function has a detailed inline comment explaining its logic. The 'find\_maximum' function also has a detailed inline comment. The script is executed, and the output is displayed in the terminal window. The output shows the average of 10 and 20 is 15, and the maximum of 3, 7, 2, and 9 is 9.

```
1 def calculate_average(numbers):
2     """Calculate the average of a list of numbers.
3     If the list is empty, return 0.
4     """
5     total = sum(numbers)
6     return total / len(numbers)
7
8
9 def find_maximum(numbers):
10    """Find the maximum value in a list of numbers.
11    If the list is empty, return the first element.
12    """
13    max_value = numbers[0]
14    for num in numbers[1:]:
15        if num > max_value:
16            max_value = num
17    return max_value
18
19 if __name__ == "__main__":
20     print("Average:", calculate_average([10, 20, 30]))
21     print("Maximum:", find_maximum([3, 7, 2, 9]))
22
23
24
```

Terminal Output:

```
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"
Average: 15.0
Maximum: 9
```

## 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:
  - o Functions
  - o 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

```
1 import ast
2 import os
3
4 def generate_function_docstring(name, args):
5     params = ""
6     for arg in args:
7         params += f"    {arg} (type): Description of {arg}.n"
8
9     return f'''
10     Brief description of {name}.
11
12     Args:
13     {params if params else "    None"}
14
15     Returns:
16     {params if params else "    None"}
17     '''
18
19 def generate_class_docstring(name):
20     return f'''
21     Brief description of {name} class.
22
23     Attributes:
24     Describe attributes here.
25
26     Methods:
27     Describe methods here.
28     '''
29
30 def insert_docstrings():
31     filename = "Ass 9.4.py"
32     if not os.path.exists(filename):
33         print("File not found!")
34         return
35
36 # FIXED ENCODING
37 with open(filename, "r", encoding="utf-8") as file:
```

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>

```
31 def insert_docstrings():
32     with open(filename, "r", encoding="utf-8") as file:
33         source = file.read()
34
35     tree = ast.parse(source)
36     lines = source.split("\n")
37
38     offset = 0
39
40     for node in ast.walk(tree):
41         if isinstance(node, ast.FunctionDef):
42             if ast.get_docstring(node) is None and len(node.body) > 0:
43                 doc = generate_function_docstring(
44                     node.name, [arg.arg for arg in node.args.args])
45                 insert_at = node.body[0].lineno - 1 + offset
46                 lines.insert(insert_at, doc)
47                 offset += doc.count("\n")
48
49         elif isinstance(node, ast.ClassDef):
50             if ast.get_docstring(node) is None and len(node.body) > 0:
51                 doc = generate_class_docstring(node.name)
52                 insert_at = node.body[0].lineno - 1 + offset
53                 lines.insert(insert_at, doc)
54                 offset += doc.count("\n")
55
56     new_filename = "Ass 9.4_documented.py"
57
58     # FIXED ENCODING
59     with open(new_filename, "w", encoding="utf-8") as file:
60         file.write("\n".join(lines))
61
62     print("Documentation generated successfully!")
63     print("New file created:", new_filename)
64
65 if __name__ == "__main__":
66     insert_docstrings()
```

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>