

NAME:CH.Kruthankiran

H.NO:2303A51404

BATCH:26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech	Assignment Type: Lab		Academic Year: 2025-2026
Course Coordinator Name	Dr. Rishabh Mittal		
Instructor(s)Name	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.No.	Question		<i>Expected Time to complete</i>
1	Lab 9 – Documentation Generation: Automatic Documentation and Code		Week 5

	<p>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 	
	<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

```

AI 9.4.py > factorial
1 def add_numbers(a, b):
2     return a + b
3 def calculate_average(numbers):
4     if not numbers:
5         raise ValueError("The list cannot be empty.")
6     total = sum(numbers)
7     return total / len(numbers)
8 def is_prime(n):
9     if n <= 1:
10        return False
11
12    for i in range(2, int(n ** 0.5) + 1):
13        if n % i == 0:
14            return False
15
16    return True
17 def factorial(n):
18
19    if n < 0:
20        raise ValueError("Factorial is not defined for negative numbers.")
21    if n == 0:
22        return 1
23
24    return n * factorial(n - 1)
25 if __name__ == "__main__":
26     print("Add Numbers:", add_numbers(10, 5))
27     print("Average:", calculate_average([5, 10, 15]))
28     print("Is Prime (11):", is_prime(11))
29     print("Factorial (5):", factorial(5))
30

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```

● PS C:\Users\kruth\OneDrive\Desktop\java> & 'c:\Users\kruth\AppData\Local\Microsoft\WindowsApps\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57458' '--' '9.4.py'
Add Numbers: 15
Average: 10.0
Is Prime (11): True
Factorial (5): 120
○ PS C:\Users\kruth\OneDrive\Desktop\java>

```

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

A screenshot of the Visual Studio Code interface. The main area shows a Python script named AI 9.4(i).py. The code defines three functions: fibonacci, binary_search, and bubble_sort. The fibonacci function generates a sequence of numbers. The binary_search function performs a search on a sorted array. The bubble_sort function sorts an array in ascending order. The code uses standard Python syntax with some inline comments. Below the code editor is a terminal window showing the execution of the script and its output. The terminal output shows the results of each function: Fibonacci(6) returns [0, 1, 1, 2, 3, 5], Binary Search(find 7) returns 3, and Bubble Sort([5, 2, 9, 1]) returns [1, 2, 5, 9].

```
AI 9.4(i).py > 📁 bubble_sort
1 def fibonacci(n):
2     if n <= 0:
3         return []
4     elif n == 1:
5         return [0]
6     sequence = [0, 1]
7     for i in range(2, n):
8         sequence.append(sequence[i - 1] + sequence[i - 2])
9     return sequence
10 def binary_search(arr, target):
11     left = 0
12     right = len(arr) - 1
13     while left <= right:
14         mid = (left + right) // 2
15         if arr[mid] == target:
16             return mid
17         elif arr[mid] < target:
18             left = mid + 1
19         else:
20             right = mid - 1
21     return -1
22 def bubble_sort(arr):
23     n = len(arr)
24     for i in range(n):
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     return arr
29 if __name__ == "__main__":
30     print("Fibonacci (6):", fibonacci(6))
31     print("Binary Search (find 7):", binary_search([1, 3, 5, 7, 9], 7))
32     print("Bubble Sort:", bubble_sort([5, 2, 9, 1]))
33
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
PS C:\Users\kruth\OneDrive\Desktop\java> & 'c:\Users\kruth\AppData\Local\Microsoft\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '9.4.py'
Add Numbers: 15
● PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive\Desktop\ossApps\python3.11.exe' 'c:\Users\kruth\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '9.4.py'
Fibonacci (6): [0, 1, 1, 2, 3, 5]
Binary Search (find 7): 3
Bubble Sort: [1, 2, 5, 9]
○ PS C:\Users\kruth\OneDrive\Desktop\java>
```

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

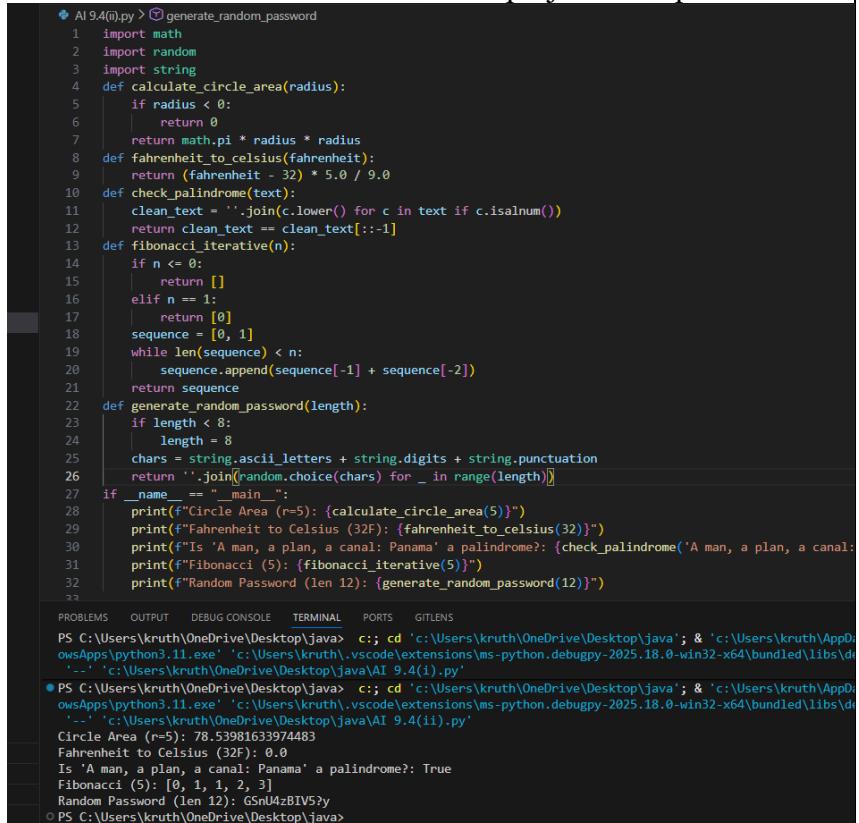
Provide a complete Python module to an AI tool and instruct it to automatically generate a **module-level docstring** at the top of the file that includes:

- 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



```
AI 9.4(ii).py > generate_random_password
1 import math
2 import random
3 import string
4 def calculate_circle_area(radius):
5     if radius < 0:
6         return 0
7     return math.pi * radius * radius
8 def fahrenheit_to_celsius(fahrenheit):
9     return (fahrenheit - 32) * 5.0 / 9.0
10 def check_palindrome(text):
11     clean_text = ''.join(c.lower() for c in text if c.isalnum())
12     return clean_text == clean_text[::-1]
13 def fibonacci_iterative(n):
14     if n <= 0:
15         return []
16     elif n == 1:
17         return [0]
18     sequence = [0, 1]
19     while len(sequence) < n:
20         sequence.append(sequence[-1] + sequence[-2])
21     return sequence
22 def generate_random_password(length):
23     if length < 8:
24         length = 8
25     chars = string.ascii_letters + string.digits + string.punctuation
26     return ''.join(random.choice(chars) for _ in range(length))
27 if __name__ == "__main__":
28     print("Circle Area (r=5): {calculate_circle_area(5)}")
29     print("Fahrenheit to Celsius (32F): {fahrenheit_to_celsius(32)}")
30     print("Is 'A man, a plan, a canal: Panama' a palindrome?: {check_palindrome('A man, a plan, a canal: Panama')}")
31     print("Fibonacci (5): {fibonacci_iterative(5)}")
32     print("Random Password (len 12): {generate_random_password(12)}")
33
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive\Desktop\java'; & 'c:\Users\kruth\AppData\Local\Programs\Python\Python3.11.exe' 'c:\Users\kruth\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug\__' 'c:\Users\kruth\OneDrive\Desktop\java\AI 9.4(i).py'
● PS C:\Users\kruth\OneDrive\Desktop\java> c;; cd 'c:\Users\kruth\OneDrive\Desktop\java'; & 'c:\Users\kruth\AppData\Local\Programs\Python\Python3.11.exe' 'c:\Users\kruth\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug\__' 'c:\Users\kruth\OneDrive\Desktop\java\AI 9.4(ii).py'
Circle Area (r=5): 78.53981633974483
Fahrenheit to Celsius (32F): 0.0
Is 'A man, a plan, a canal: Panama' a palindrome?: True
Fibonacci (5): [0, 1, 1, 2, 3]
Random Password (len 12): GSnU4zBV57
○ PS C:\Users\kruth\OneDrive\Desktop\java>
```

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

```
AI 9.4(iii).py > ...
1 def calculate_discount(price, discount_percent):
2     if price < 0 or discount_percent < 0 or discount_percent > 100:
3         return 0
4
5     discount_amount = price * (discount_percent / 100)
6     return price - discount_amount
7
8
9 def is_even(number):
10    return number % 2 == 0
11
12 def calculate_simple_interest(principal, rate, time):
13    if principal < 0 or rate < 0 or time < 0:
14        return 0
15
16    return (principal * rate * time) / 100
17
18 def find_largest(numbers):
19    if not numbers:
20        return None
21
22    largest = numbers[0]
23
24    for num in numbers:
25        if num > largest:
26            largest = num
27
28    return largest
29 if __name__ == "__main__":
30     print("Discounted Price:", calculate_discount(200, 15))
31     print("Is Even (10):", is_even(10))
32     print("Simple Interest:", calculate_simple_interest(1500, 4, 3))
33     print("Largest Number:", find_largest([5, 12, 3, 21, 8]))
34

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

PS C:\Users\kruth\OneDrive\Desktop\java> c:&; cd 'c:\Users\kruth\OneDrive\Desktop\owsApps\python3.11.exe' 'c:\Users\kruth\vscode\extensions\ms-python.debugpy-2025.1.1' 'c:\Users\kruth\OneDrive\Desktop\java\AI 9.4(ii).py'
● PS C:\Users\kruth\OneDrive\Desktop\java> c:&; cd 'c:\Users\kruth\OneDrive\Desktop\owsApps\python3.11.exe' 'c:\Users\kruth\vscode\extensions\ms-python.debugpy-2025.1.1' 'c:\Users\kruth\OneDrive\Desktop\java\AI 9.4(iii).py'
Discounted Price: 170.0
Is Even (10): True
Simple Interest: 180.0
Largest Number: 21
○ PS C:\Users\kruth\OneDrive\Desktop\java>
```

Task 5: Building a Mini Automatic Documentation Generator

	<p>Scenario</p> <p>Your team wants a simple internal tool that helps developers start documenting new Python files quickly, without writing documentation from scratch.</p> <p>Task Description</p> <p>Design a small Python utility that:</p> <ul style="list-style-type: none">• Reads a given .py file• Automatically detects:<ul style="list-style-type: none">◦ Functions◦ Classes• Inserts placeholder Google-style docstrings for each detected function or class <p>AI tools may be used to assist in generating or refining this utility.</p> <p>Note: The goal is documentation scaffolding, not perfect documentation.</p> <p>Expected Outcome</p> <ul style="list-style-type: none">• A working Python script that processes another .py file• Automatically inserted placeholder docstrings	
--	---	--

- Clear demonstration of how AI can assist in documentation automation

```

AI 9.4(iv).py > generate_class_docstring
16 def generate_class_docstring(class_node):
20     Add class attributes here.
21     Methods:
22         Describe important methods here.
23         .....
24     def process_file(file_path):
25         source = Path(file_path).read_text()
26         tree = ast.parse(source)
27         lines = source.split("\n")
28         offset = 0
29         for node in ast.walk(tree):
30             if isinstance(node, (ast.FunctionDef, ast.ClassDef)):
31                 if ast.get_docstring(node):
32                     continue
33                 insert_line = node.body[0].lineno - 1 + offset
34                 if isinstance(node, ast.FunctionDef):
35                     docstring = generate_function_docstring(node)
36                 else:
37                     docstring = generate_class_docstring(node)
38                 lines.insert(insert_line, docstring)
39                 offset += docstring.count("\n") + 1
40             new_file_path = file_path.replace(".py", "_documented.py")
41             Path(new_file_path).write_text("\n".join(lines))
42             print(f"Documentation scaffolding complete.")
43             print(f"Updated file saved as: {new_file_path}")
44         if __name__ == "__main__":
45             if len(sys.argv) != 2:
46                 print("Usage: python mini_doc_generator.py <python_file.py>")
47             else:
48                 process_file(sys.argv[1])
49

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```

PS C:\Users\kruth\OneDrive\Desktop\java> c:; cd 'c:\Users\kruth\OneDrive\owsApps\python3.11.exe' 'c:\Users\kruth\.vscode\extensions\ms-python.debug' -- 'c:\Users\kruth\OneDrive\Desktop\java\AI 9.4(iii).py'
Is Even (10): True
Simple Interest: 180.0
Largest Number: 21
● PS C:\Users\kruth\OneDrive\Desktop\java> c:; cd 'c:\Users\kruth\OneDrive\owsApps\python3.11.exe' 'c:\Users\kruth\.vscode\extensions\ms-python.debug' -- 'c:\Users\kruth\OneDrive\Desktop\java\AI 9.4(iv).py'
Usage: python mini_doc_generator.py <python_file.py>
○ PS C:\Users\kruth\OneDrive\Desktop\java>

```

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