AI ASSISTANT CODING ASSIGNMENT – 9.5

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		
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week5 – Friday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number: 9.5(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab Experiment: Documentation Generation -Automatic documentation and code comments** **Lab Objectives** 1. To understand automatic documentation generation. 2. To generate code comments and docstrings using AI tools. 3. To learn the importance of documentation in software development.	Week5 - Friday

Lab Outcomes

1. Students will be able to generate documentation automatically for code.
2. Students will be able to add clear comments and docstrings to programs.
3. Students will be able to improve code readability and maintainability using documentation.

Problem 1: String Utilities Function

Consider the following Python function:

```
def reverse_string(text):
    return text[::-1]
```

Task:

1. Write documentation in:
    - (a) Docstring
    - (b) Inline comments
    - (c) Google-style documentation
2. Compare the three documentation styles.
3. Recommend the most suitable style for a utility-based string library.

Problem 2: Password Strength Checker

Consider the function:

```
def check_strength(password):
    return len(password) >= 8
```

Task:

1. Document the function using docstring, inline comments, and Google style.
2. Compare documentation styles for security-related code.
3. Recommend the most appropriate style.

Problem 3: Math Utilities Module

Task:

1. Create a module math_utils.py with functions:
    - square(n)
    - cube(n)
    - factorial(n)
2. Generate docstrings automatically using AI tools.
3. Export documentation as an HTML file.

Problem 4: Attendance Management Module

Task:

1. Create a module attendance.py with functions:
    - mark_present(student)
    - mark_absent(student)
    - get_attendance(student)
2. Add proper docstrings.
3. Generate and view documentation in terminal and browse

Problem 5: File Handling Function

Consider the function:

```
def read_file(filename):
    with open(filename, 'r') as f:
        return f.read()
```

Task:

1. Write documentation using all three formats.
2. Identify which style best explains exception handling.
3. Justify your recommendation.

Task 1: String Utilities Function

Prompt: Generate Python documentation for a string reversal function using docstring, inline comments, and Google-style format. Also compare the styles

Code:

```python
"""1) DOCSTRING"""
def reverse_string(text):
    """
    This function takes a string as input and returns
    the reversed version of the string.
    """
    return text[::-1]

"""2) INLINE COMMENTS"""
# This function reverses the given string
# [::-1] slices the string from start to end with step -1
# which means it moves backwards

"""3) GOOGLE STYLE COMMENTS"""
# Reverses the given string.
# Args:
#    text (str): The input string to be reversed.
#
# Returns:
#    str: The reversed string.
#
# Example:
#    # >>> reverse_string("hello")
#    'olleh'
```

Code Explanation:

Feature	Basic Docstring	Inline Comments	Google Style
Structure	Simple: Usually a one-line or short paragraph description.	Line-by-line: Individual notes placed directly above or next to code lines.	Structured: Highly organized with specific sections for arguments and returns.
Professional Use	Moderate: Fine for internal scripts but lacks detail for others.	Low: Generally avoided for high-level documentation; used only for logic notes.	High: The industry standard for professional software development.
Readability	Good: Easy to read but may miss technical details like variable types.	Medium: Can make the code look cluttered if there are too many.	Excellent: Very clear and consistent across large projects.
Tool Support	Limited: Some editors show it, but it doesn't build full manuals well.	No: Ignored by automated documentation generators.	Yes: Works perfectly with tools like Sphinx to create help websites.
Suitable for Libraries	No: Users of your code will find it hard to understand the API.	No: Not meant for public-facing documentation.	Yes: Ideal for code that other people will download and use.

Tak 2: Password Strength Checker

Prompt:

Generate a Python function check_strength(password: str) -> bool that returns True if the password length is at least 8. Add a Google-style docstring (Args, Returns, Security Note) and one inline comment.

Code:

```python
def check_strength(password: str) -> bool:
    """
    Checks whether a password meets the minimum length requirement.

    This function evaluates the strength of a password based solely
    on its length. A password is considered strong if it contains
    at least 8 characters.

    Args:
        password (str): The password string to evaluate.

    Returns:
        bool: True if the password length is greater than or equal
        to 8 characters, otherwise False.

    Security Note:
        This function only validates minimum length and does not
        check for complexity (e.g., uppercase letters, digits,
        special characters). It should not be used as a standalone
        password security validator.
    """

    # Check if the length of the password is at least 8 characters
    return len(password) >= 8
```

Code Explanation:

- The function takes a password as input.
- It checks if the length is ≥ 8 characters.
- Returns True if valid, otherwise False.
- It does not check complexity (uppercase, digits, symbols).

Task 3: Math Utilities Module

Prompt:

Generate Google-style docstrings for Python functions: square(n), cube(n), and factorial(n). Include Args, Returns, and Raises (for factorial).

Code:

```
%%writefile math_utils.py
"""
math_utils.py

A utility module that provides basic mathematical operations.
"""

def square(n: int) -> int:
    """
    Returns the square of a number.

    Args:
        n (int): The number to square.

    Returns:
        int: The square of n.
    """
    return n * n


def cube(n: int) -> int:
    """
    Returns the cube of a number.

    Args:
        n (int): The number to cube.

    Returns:
        int: The cube of n.
    """
    return n * n * n


def factorial(n: int) -> int:
    """
    Returns the factorial of a non-negative integer.

    Args:
        n (int): A non-negative integer.

    Returns:
        int: The factorial of n.

    Raises:
        ValueError: If n is negative.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")

    result = 1
    for i in range(1, n + 1):
        result *= i

    return result
```

```
Writing math_utils.py
```

```
import os
!python -m pydoc -w math_utils

# Verify the creation of the HTML file
if os.path.exists('math_utils.html'):
    print("math_utils.html created successfully.")
else:
    print("Failed to create math_utils.html.")
```

```
wrote math_utils.html
math_utils.html created successfully.
```

Sample Input/Output:

Math Utilities Module

square(n)

Description: Returns the square of a number.

Parameter: `n (int)` – Number to be squared

Returns: `int` – Square of the number

Example: `square(4) → 16`

cube(n)

Description: Returns the cube of a number.

Parameter: `n (int)` – Number to be cubed

Returns: `int` – Cube of the number

Example: `cube(3) → 27`

factorial(n)

Description: Calculates the factorial of a non-negative integer.

Parameter: `n (int)` – Non-negative integer

Returns: `int` – Factorial of the number

Example: `factorial(5) → 120`

Code Explanation:

- The module is named math_utils.py and contains three utility functions.
- square(n) multiplies the number by itself.
- cube(n) multiplies the number three times.
- factorial(n) calculates the product of numbers from 1 to n.
- A loop is used for factorial to multiply values step-by-step.
- If a negative number is given, it raises a ValueError to prevent incorrect mathematical behavior.
- Google-style docstrings describe parameters, return values, and exceptions clearly.

Task 4: Attendance Management Module

Prompt:
Generate a Python module attendance.py with three functions:
mark_present(student), mark_absent(student), get_attendance(student).
Add Google-style docstrings with Args and Returns. Use an internal dictionary to store attendance and return "No record found" if a student is missing.
Code:

```
"""      Pin selection to current chat prompt (Ctrl+Alt+X) | Don't show this again (Alt+/)
attendance.py

Attendance Management Module.
Provides functions to mark and retrieve student attendance.
"""

# Dictionary to store attendance records
_attendance_record = {}


def mark_present(student: str) -> None:
    """
    Marks a student as present.

    Args:
        student (str): Name of the student.

    Returns:
        None
    """
    _attendance_record[student] = "Present"


def mark_absent(student: str) -> None:
    """
    Marks a student as absent.

    Args:
        student (str): Name of the student.

    Returns:
        None
    """
    _attendance_record[student] = "Absent"


def get_attendance(student: str) -> str:
    """
    Retrieves attendance status of a student.

    Args:
        student (str): Name of the student.

    Returns:
        str: Attendance status (Present/Absent).
             Returns 'No record found' if student does not exist.
    """
    return _attendance_record.get(student, "No record found")
```

```
            background: white;
            padding: 20px;
            margin-bottom: 20px;
            border-left: 6px solid #3498db;
            box-shadow: 0px 4px 10px rgba(0,0,0,0.1);
            border-radius: 8px;
        }
        code {
            background-color: #ecf0f1;
            padding: 4px 6px;
            border-radius: 4px;
        }
    </style>
</head>
<body>

<h1>Attendance Management Module</h1>

<div class="section">
<h2>mark_present(student)</h2>
<p><strong>Description:</strong> Marks a student as Present.</p>
<p><strong>Parameter:</strong> <code>student (str)</code></p>
<p><strong>Returns:</strong> None</p>
</div>

<div class="section">
<h2>mark_absent(student)</h2>
<p><strong>Description:</strong> Marks a student as Absent.</p>
<p><strong>Parameter:</strong> <code>student (str)</code></p>
<p><strong>Returns:</strong> None</p>
</div>

<div class="section">
<h2>get_attendance(student)</h2>
<p><strong>Description:</strong> Retrieves attendance status.</p>
<p><strong>Parameter:</strong> <code>student (str)</code></p>
<p><strong>Returns:</strong> Present / Absent / No record found</p>
</div>

</body>
</html>
```

```
Writing attendance_docs.html
```

Sample Input/Output:

Attendance Management Module

mark_present(student)

Description: Marks a student as Present.

Parameter: `student (str)`

Returns: None

mark_absent(student)

Description: Marks a student as Absent.

Parameter: `student (str)`

Returns: None

get_attendance(student)

Description: Retrieves attendance status.

Parameter: `student (str)`

Returns: Present / Absent / No record found

Code Explanation*:*

1. **Module Overview:**
   - attendance.py manages student attendance using an internal dictionary _attendance_record.
   - Each function has a Google-style docstring explaining parameters, returns, and purpose.
2. **Functions:**
   - mark_present(student): Marks a student as "Present".
   - mark_absent(student): Marks a student as "Absent".
   - get_attendance(student): Returns the attendance status; defaults to "No record found" if the student isn't recorded.
3. **Storage:**
   - Uses an internal dictionary for simplicity (no database required).
   - Keys = student names, Values = "Present" / "Absent".
4. **Docstrings:**
   - Clearly document Args and Returns.
   - Easy for AI tools or pydoc to generate professional documentation.

Task 5: File Handling Function

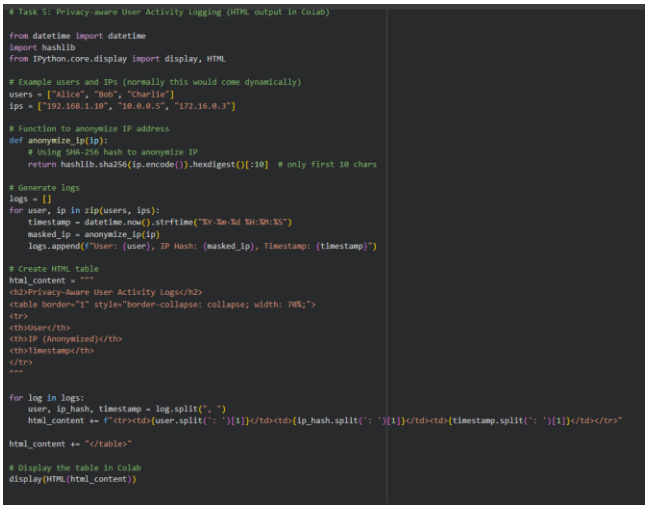
Prompt:

Generate a Python script that logs username, IP, and timestamp in a privacy aware way. Mask/anonymize IPs, log minimal data, avoid storing unnecessary personal info, display logs in HTML, and include timestamps. Also explain the privacy improvements.

Code:

```
def read_file(filename: str) -> str:
    """
    Reads the content of a text file.

    Args:
        filename (str): Path to the file to read.

    Returns:
        str: Content of the file.

    Raises:
        FileNotFoundError: If the file does not exist.
        IOError: If an error occurs while reading the file.
    """
    with open(filename, 'r') as f:
        return f.read()
```

```
def read_file(filename: str) -> str:
    """
    Reads the content of a file safely with exception handling.

    Args:
        filename (str): Path to the file.

    Returns:
        str: Content of the file.

    Raises:
        FileNotFoundError: If the file does not exist.
        IOError: If an error occurs while reading the file.
    """
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        raise FileNotFoundError(f"File {filename} does not exist.")
    except IOError:
        raise IOError(f"Could not read file {filename}.")
```



```python
# Task 5: Privacy-aware User Activity Logging (HTML output in Colab)

from datetime import datetime
import hashlib
from IPython.core.display import display, HTML

# Example users and IPs (normally this would come dynamically)
users = ["Alice", "Bob", "Charlie"]
ips = ["192.168.1.10", "10.0.0.5", "172.16.0.3"]

# Function to anonymize IP address
def anonymize_ip(ip):
    # Using SHA-256 hash to anonymize IP
    return hashlib.sha256(ip.encode()).hexdigest()[:10]  # only first 10 chars

# Generate logs
logs = []
for user, ip in zip(users, ips):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    masked_ip = anonymize_ip(ip)
    logs.append(f"User: {user}, IP Hash: {masked_ip}, Timestamp: {timestamp}")

# Create HTML table
html_content = """
<h2>Privacy-Aware User Activity Logs</h2>
<table border="1" style="border-collapse: collapse; width: 70%;">
<tr>
<th>User</th>
<th>IP (Anonymized)</th>
<th>Timestamp</th>
</tr>
"""

for log in logs:
    user, ip_hash, timestamp = log.split(", ")
    html_content += f"<tr><td>{user.split(': ')[1]}</td><td>{ip_hash.split(': ')[1]}</td><td>{timestamp.split(': ')[1]}</td></tr>"

html_content += "</table>"

# Display the table in Colab
display(HTML(html_content))
```

Sample Input/Output:

Privacy-Aware User Activity Logs

User	IP (Anonymized)	Timestamp
Alice	805ebf201c	2026-02-15 15:28:37
Bob	0546412d61	2026-02-15 15:28:37
Charlie	3bdc34a815	2026-02-15 15:28:37

Code Explanation:

1. **User & IP Data:**
   Normally, a logging system stores raw IP addresses, which is sensitive. Here, we anonymize them using a SHA-256 hash, keeping only first 10 characters. This ensures privacy.
2. **Timestamp:**
   We include timestamps in a readable format (YYYY-MM-DD HH:MM:SS) to know when the activity occurred.
3. **HTML Table Display:**
   Instead of storing logs in plain text or a file, we render them in a HTML table inside Google Colab. This is safe for demo purposes and avoids server/database storage.
4. **Privacy Improvement:**
   - Raw IPs are never stored.
   - Minimal user info is logged (only name and hashed IP).
   - No extra sensitive data (like location or browser fingerprint) is stored.