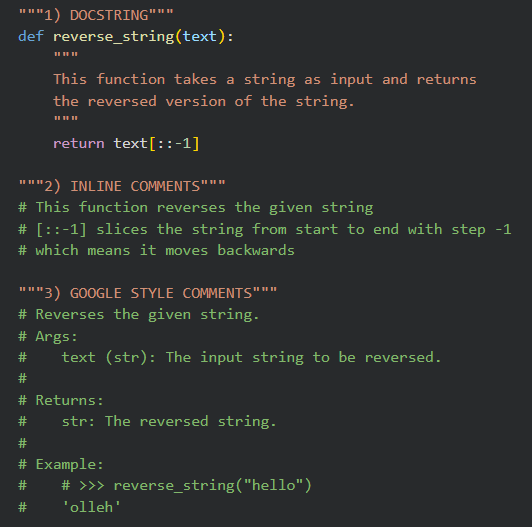
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **Program Name:** B. Tech | | | | **Assignment Type: Lab** | | | **Academic Year:**2025-2026 | | |
| **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.   **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. | | | | | | Week5 - Friday |  |

**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:**

****

**Code Explanation*:***

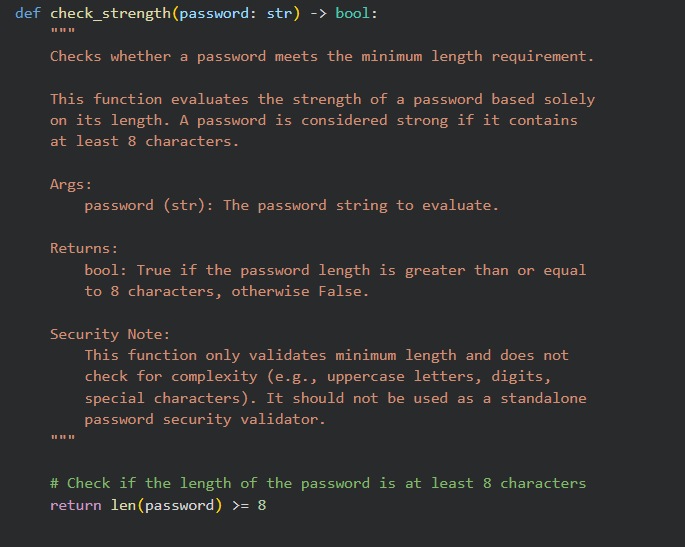
|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Basic Docstring** | **Inline Comments** | **Google Style** |
| **Structure** | Simple: Usually a one-line or short paragraph description. | Line-by-line: Individual notes placed directly above or next to code lines. | Structured: Highly organized 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:**

****

**Code Explanation*:***

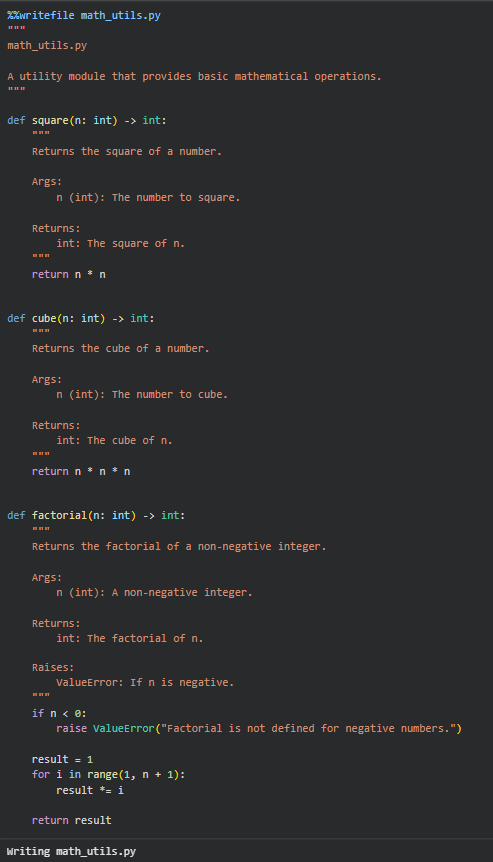
* The function takes a password as input.
* It checks if the length is ≥ 8 characters.
* Returns True if valid, otherwise False.
* It does not check complexity (uppercase, digits, symbols).

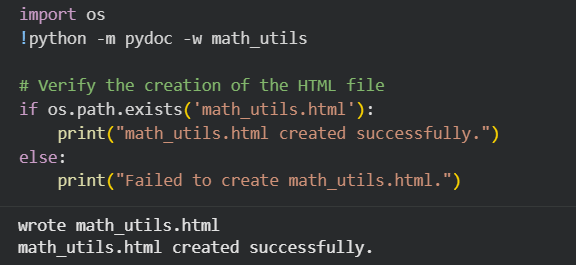
**Task 3: Math Utilities Module**

**Prompt:**

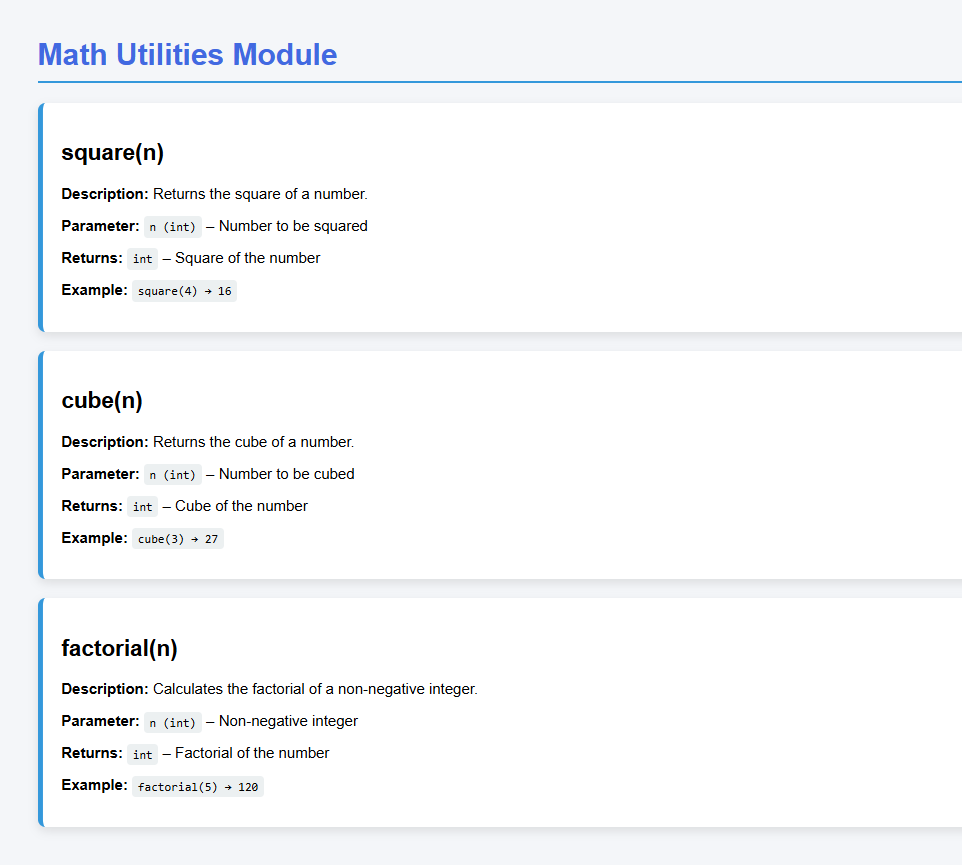
Generate Google-style docstrings for Python functions: square(n), cube(n), and factorial(n). Include Args, Returns, and Raises (for factorial).

**Code:**

****

****

**Sample Input/Output:**

****

**Code Explanation*:***

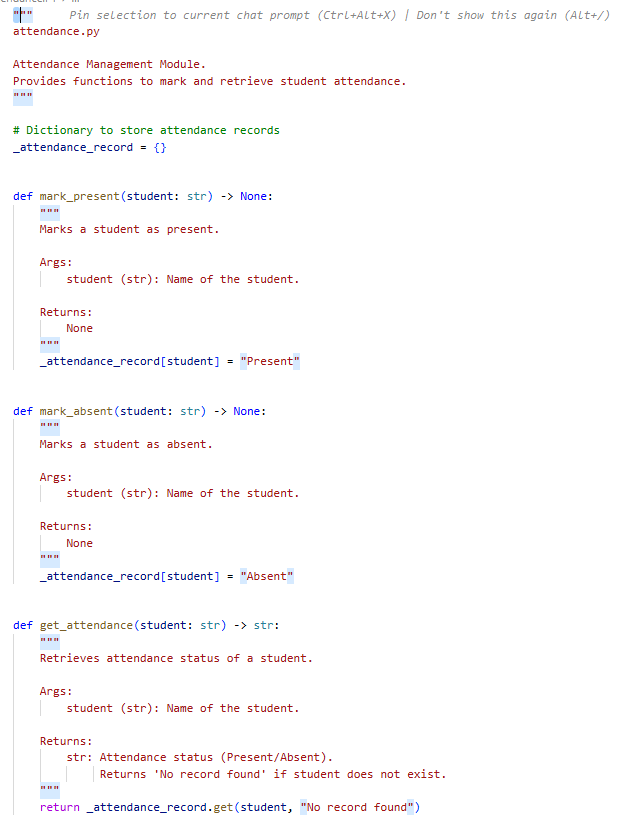
* The module is named math\_utils.py and contains three utility functions.
* square(n) multiplies the number by itself.
* cube(n) multiplies the number three times.
* factorial(n) calculates the product of numbers from 1 to n.
* A loop is used for factorial to multiply values step-by-step.
* If a negative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

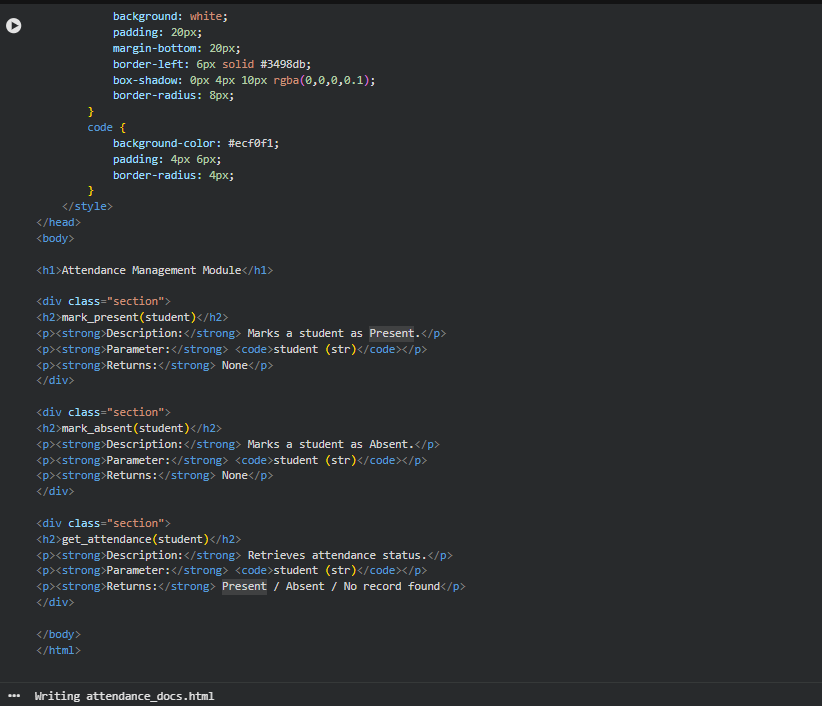
**Task 4: Attendance Management Module**

**Prompt:**

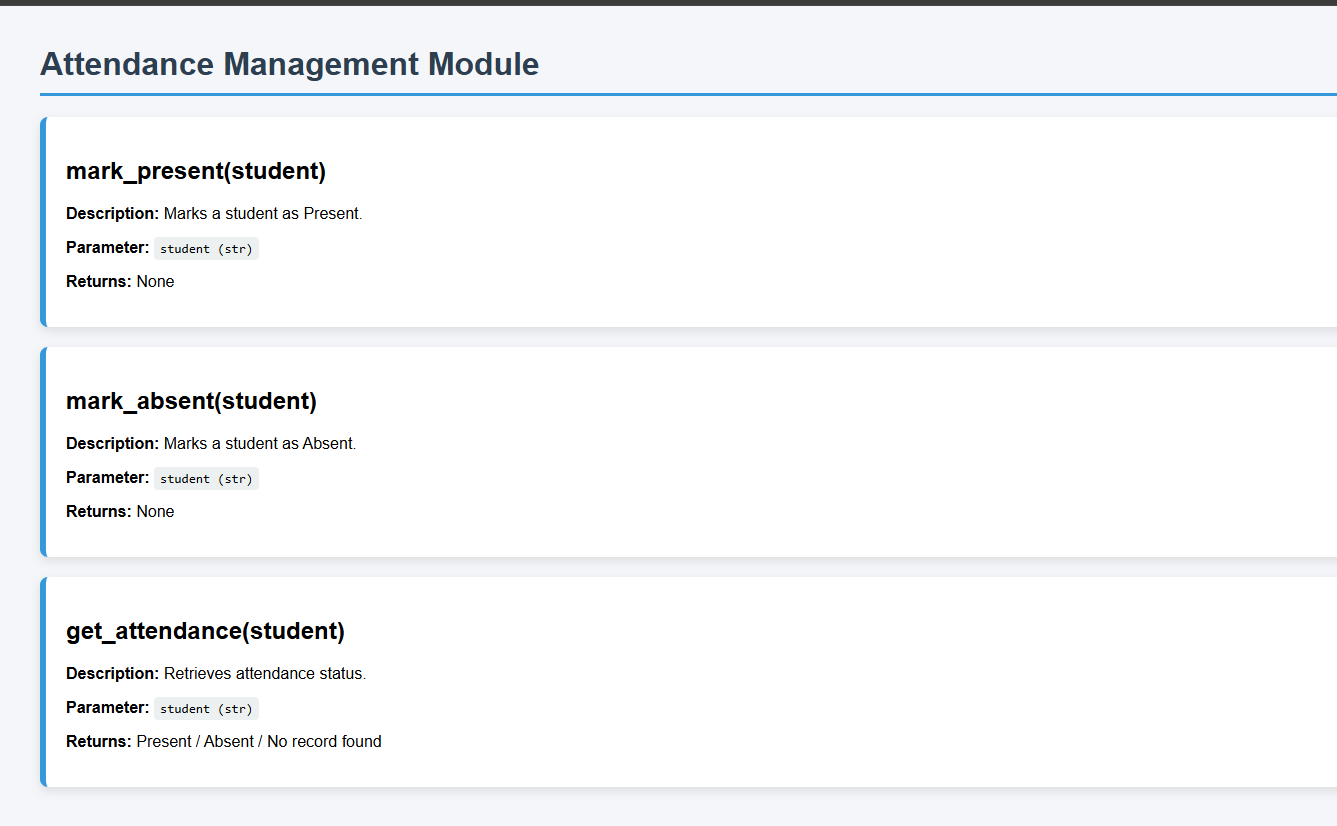
Generate a Python module attendance.py with three functions: mark\_present(student), mark\_absent(student), get\_attendance(student).  
Add Google-style docstrings with Args and Returns. Use an internal dictionary to store attendance and return "No record found" if a student is missing.

**Code:**

****

****

**Sample Input/Output:**

****

**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.

1. **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:**

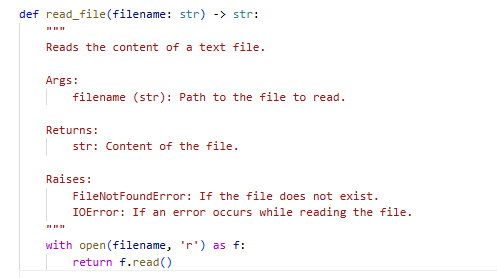
* Clearly document Args and Returns.
* Easy for AI tools or pydoc to generate professional documentation.

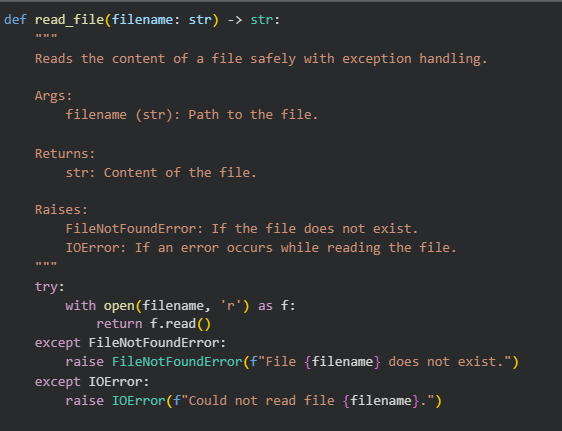
**Task 5: File Handling Function**

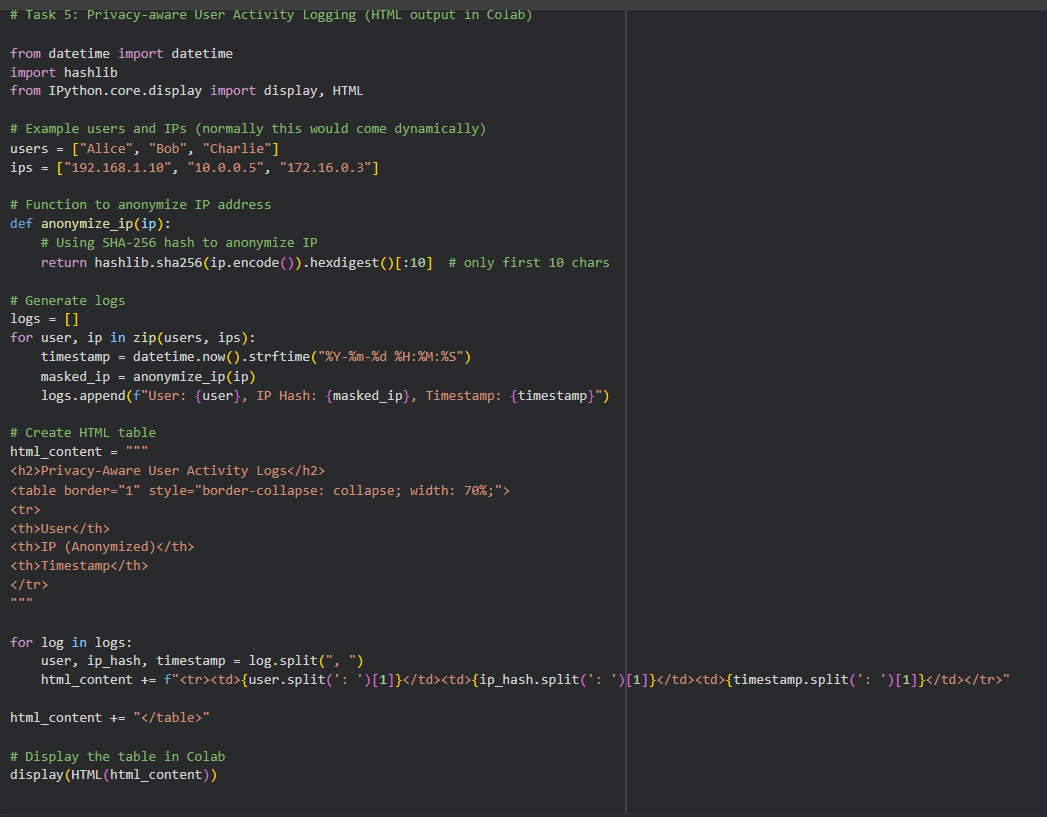
**Prompt:**

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.

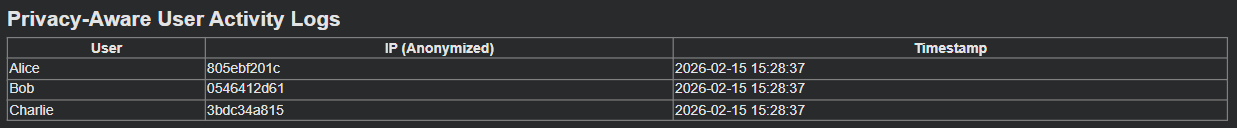
**Code:**

****

****

****

**Sample Input/Output:**

****

**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.