

# AI Assistant Coding

## Assignment 9.5

Name : N. Eshwar

HT. No : 2303A52072

Batch: 32

**Problem 1:** String Utilities Function Consider the following Python function:

```
def reverse_string(text):
```

```
    return text[::-1]
```

### Task:

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

### Code:

```
def reverse_string(s: str) -> str:
    """Reverse a given string.

    Args:
        s (str): The string to reverse.

    Returns:
        str: The reversed string.
    """
    return s[::-1] # This slicing technique reverses the string by stepping
                  through it backwards.
```

Aspect	Docstring	Inline Comments	Google-Style Doc
Readability	High	Medium	Very High
Explains logic	Low	High	Medium
Explains parameters	Low	No	High
Tool support (IDE/docs)	Yes	No	Yes
Suitable for libraries	Medium	Low	High

**Recommendation:** Use Google-style docstrings + minimal inline comments only when logic is non-obvious.

---

**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.

**Code:**

```
def check_password_strength(password: str) -> str:
    """Check the strength of a given password.

    Args:
        password (str): The password to check.

    Returns:
        str: A message indicating the strength of the password.
    """
    if len(password) < 6: # Passwords shorter than 6 characters are considered weak.
        return "Weak password: Too short."
    elif len(password) < 12: # Passwords between 6 and 12 characters are considered moderate.
        return "Moderate password: Could be stronger."
    else:
        return "Strong password: Good job!" # Passwords 12 characters or longer are considered strong.
```

Aspect	Docstring	Inline Comments	Google-Style Doc
Clarity	Medium	High (logic only)	Very High
Explains security rules	Low	High	High
Describes inputs/outputs	Low	No	Yes
Maintainability	Medium	Low	High
Suitable for security code	Medium	Medium	Best

**Recommendation:** Google-Style Documentation

---

### 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.

#### Code:

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

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

    Returns:
        int: The square of the number.
    """
    return n * n # This multiplies the number by itself to get the square.
def cube(n: int) -> int:
    """Calculate the cube of a number.

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

    Returns:
        int: The cube of the number.
    """
    return n * n * n # This multiplies the number by itself three times to
get the cube.
def factorial(n: int) -> int:
    """Calculate the factorial of a number.

    Args:
        n (int): The number to calculate the factorial of.

    Returns:
        int: The factorial of the number.
    """
    if n == 0:
        return 1 # Base case: factorial of 0 is 1
    else:
        return n * factorial(n - 1) # Recursive case: n! = n * (n - 1)!
```

[index](#)  
**math\_utils** [d:\course\aiac\lab9\\_24\\_2\\_2026\math\\_utils.py](#)

## Functions

**cube**(n: int) -> int  
Calculate the cube of a number.

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

Returns:  
int: The cube of the number.

**factorial**(n: int) -> int  
Calculate the factorial of a number.

Args:  
n (int): The number to calculate the factorial of.

Returns:  
int: The factorial of the number.

**square**(n: int) -> int  
Calculate the square of a number.

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

Returns:  
int: The square of the number.

---

## 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

### Code:

```

def mark_present(student: str, attendance: dict) -> None:
    """Marks a student as present in the attendance dictionary.
    param student: The name of the student to mark as present.
    param attendance: The dictionary to update with the student's attendance
    status.
    """
    attendance[student] = 'Present'
def mark_absent(student: str, attendance: dict) -> None:
    """Marks a student as absent in the attendance dictionary.
    param student: The name of the student to mark as absent.
    param attendance: The dictionary to update with the student's attendance
    status."""
    attendance[student] = 'Absent'
def get_attendance(student: str, attendance: dict) -> str:
    """Returns the attendance status of a student.
    param student: The name of the student whose attendance status is to be
    retrieved.
    param attendance: The dictionary containing the attendance records.
    return: The attendance status of the student, or 'Not Recorded'
    if the student is not found in the attendance dictionary."""
    return attendance.get(student, 'Not Recorded')

```

## Output: Terminal

```

# Help on module attendance:

# NAME
#     attendance

# FUNCTIONS
#     get_attendance(student: str, attendance: dict) -> str
#         Returns the attendance status of a student.
#         param student: The name of the student whose attendance status is to
be retrieved.
#         param attendance: The dictionary containing the attendance records.
#         return: The attendance status of the student, or 'Not Recorded'
#         if the student is not found in the attendance dictionary.

#     mark_absent(student: str, attendance: dict) -> None
#         Marks a student as absent in the attendance dictionary.
#         param student: The name of the student to mark as absent.
#         param attendance: The dictionary to update with the student's
attendance status.

#     mark_present(student: str, attendance: dict) -> None
#         Marks a student as present in the attendance dictionary.
#         param student: The name of the student to mark as present.

```

```
#         param attendance: The dictionary to update with the student's
attendance status.

# FILE
#         d:\course\aiac\lab9_24_2_2026\attendance.py
```

## Browser:

Python 3.14.0 [tags/v3.14.0:ebf955d, MSC v.1944 64 bit (AMD64)]  
Windows-11

### attendance

#### Functions

```
get_attendance(student: str, attendance: dict) -> str
    Returns the attendance status of a student.
    param student: The name of the student whose attendance status is to be retrieved.
    param attendance: The dictionary containing the attendance records.
    return: The attendance status of the student, or 'Not Recorded'
    if the student is not found in the attendance dictionary.

mark_absent(student: str, attendance: dict) -> None
    Marks a student as absent in the attendance dictionary.
    param student: The name of the student to mark as absent.
    param attendance: The dictionary to update with the student's attendance status.

mark_present(student: str, attendance: dict) -> None
    Marks a student as present in the attendance dictionary.
    param student: The name of the student to mark as present.
    param attendance: The dictionary to update with the student's attendance status.
```

#### Data

```
NAME = 1
```

---

**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.

## Code:

```
def read_file(file_path: str) -> str:
    """Reads the content of a file and returns it as a string.
    param file_path: The path to the file to be read.
    return: The content of the file as a string.
    Exceptions: Raises FileNotFoundError if the file does not exist.
    """
    try:
        with open(file_path, 'r') as file:
            content = file.read()
        return content
    except FileNotFoundError:
        raise FileNotFoundError(f"The file at {file_path} was not found.")
read_file('example.txt')
```

## Recommended Style: Google-Style Documentation

### Justification:

File handling is error-prone (missing files, permission issues)

Google-style documentation:

Clearly explains exceptions

Improves code reliability and usability

Helps developers handle errors correctly

Is widely used in production and open-source projects