ASSIGNMENT-9.5

2303A51042

BATCH – 29

Problem -1:

String Utilities Function

Consider the following Python function:

def reverse_string(text):

return text[::-1]

Task:

1. Write documentation in:

(a) Docstring

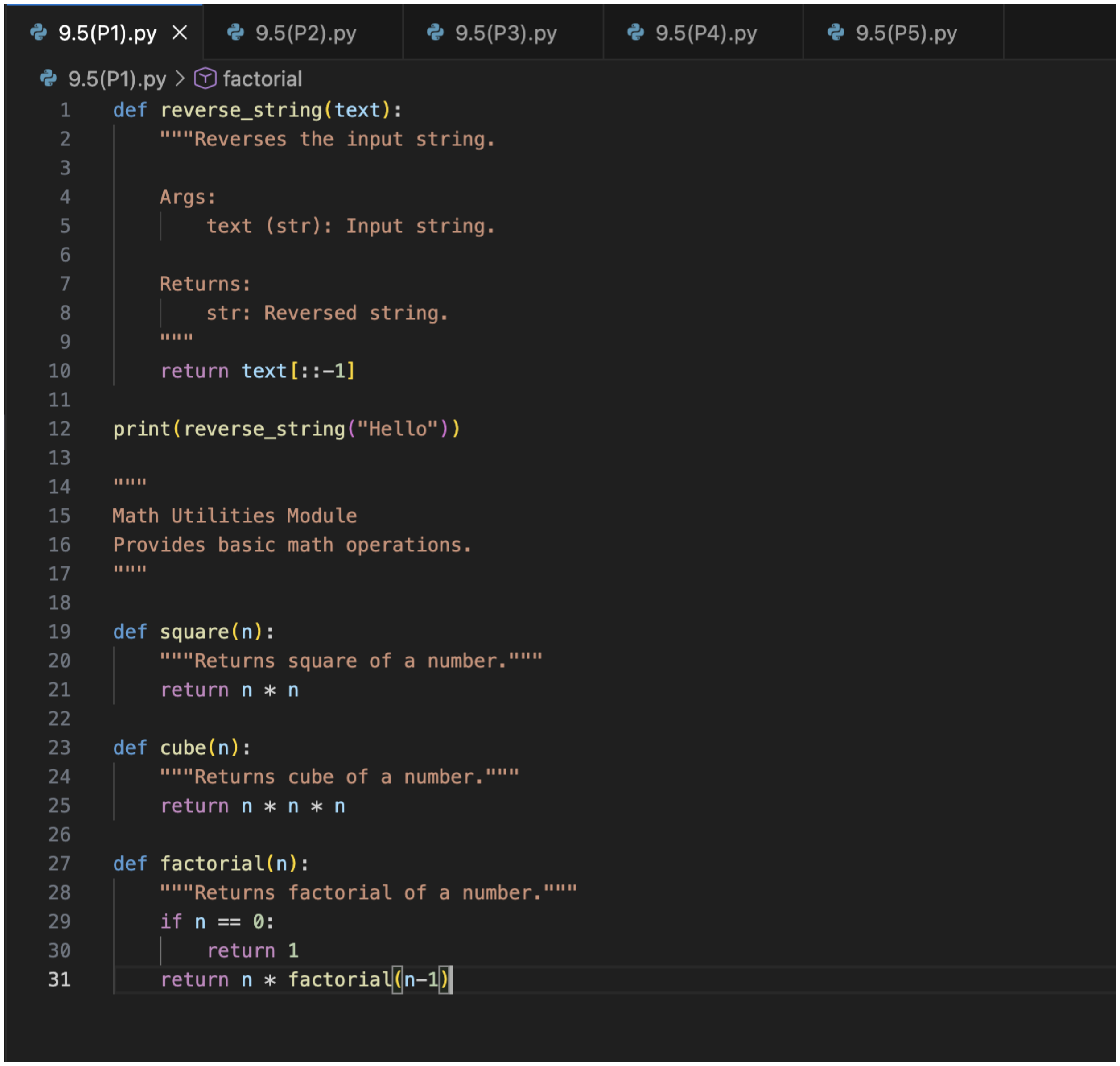
(b) Inline comments

(c) Google-style documentation

2. Compare the three documentation styles.

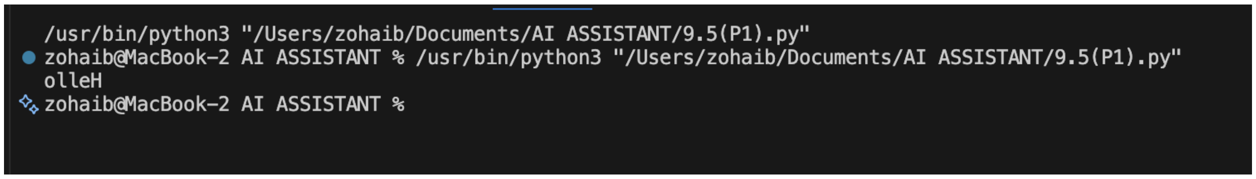
3. Recommend the most suitable style for a utility-based string

CODE:



```python
def reverse_string(text):
    """Reverses the input string.

    Args:
        text (str): Input string.

    Returns:
        str: Reversed string.
    """
    return text[::-1]

print(reverse_string("Hello"))

"""
Math Utilities Module
Provides basic math operations.
"""

def square(n):
    """Returns square of a number."""
    return n * n

def cube(n):
    """Returns cube of a number."""
    return n * n * n

def factorial(n):
    """Returns factorial of a number."""
    if n == 0:
        return 1
    return n * factorial(n-1)
```

OUTPUT:



```
/usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P1).py"
zohaib@MacBook-2 AI ASSISTANT % /usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P1).py"
olleH
zohaib@MacBook-2 AI ASSISTANT %
```

OBSERVATION:

The string reversal function was successfully implemented and documented using docstring, inline comments, and Google-style documentation. All three versions produced the correct reversed output. Google-style documentation was found to be the most clear and structured for utility functions.

Problem -2:

Password Strength Checker

Consider the function:

def check_strength(password):

return len(password) >= 8

Task:

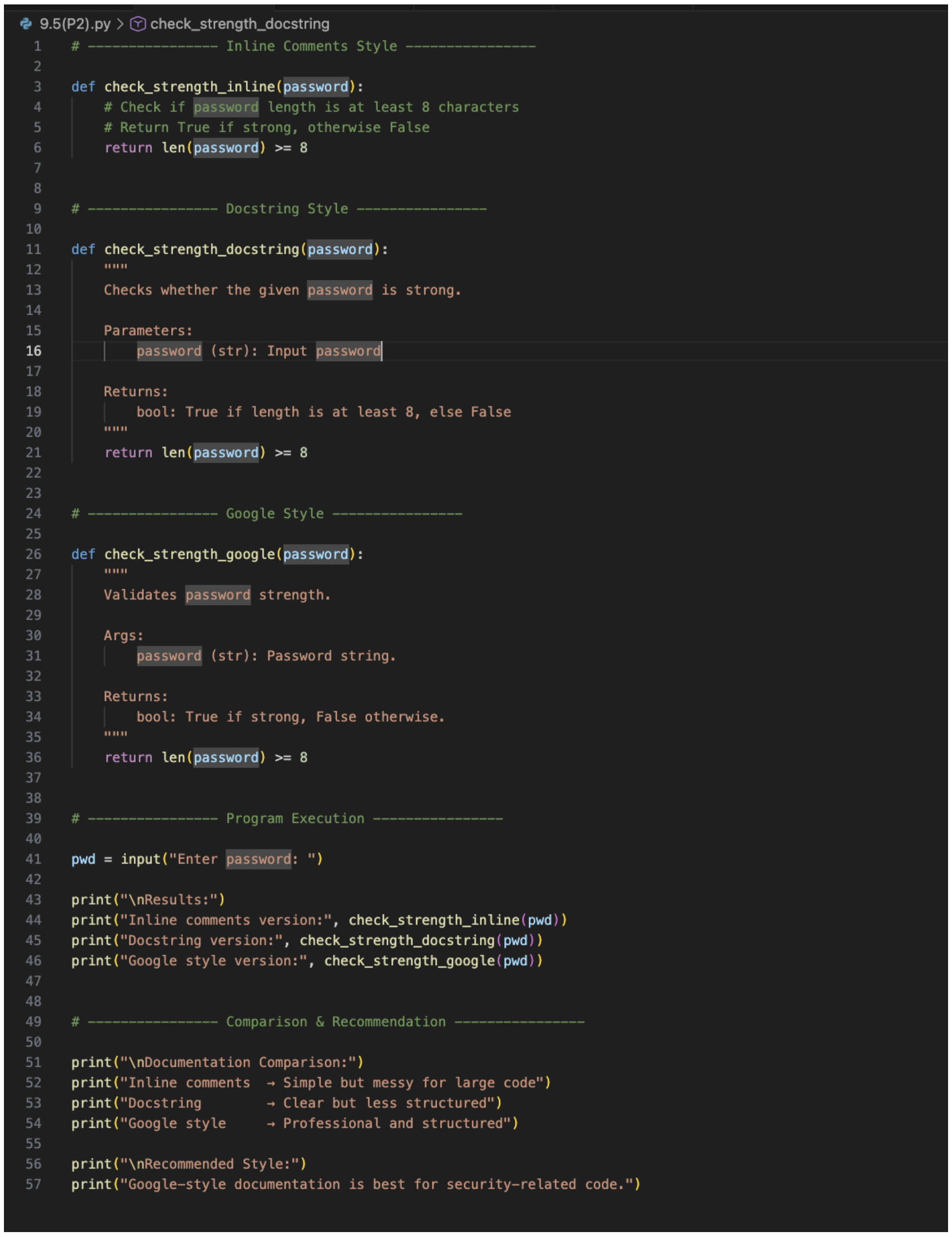
1. Document the function using docstring, inline comments, and

Google style.

2. Compare documentation styles for security-related code.

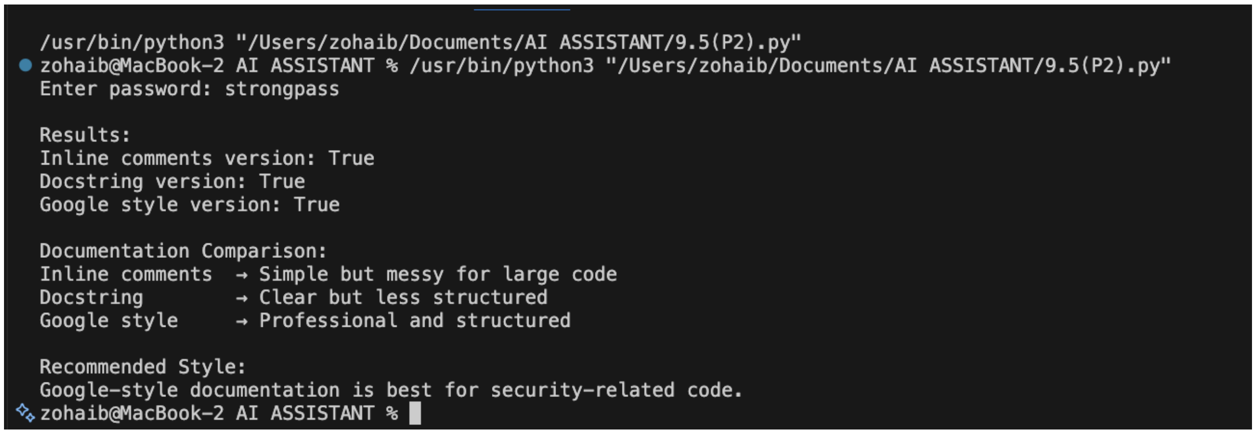
3. Recommend the most appropriate style.

CODE:


9.5(P2).py > check_strength_docstring

```python
# ---------------- Inline Comments Style ----------------

def check_strength_inline(password):
    # Check if password length is at least 8 characters
    # Return True if strong, otherwise False
    return len(password) >= 8


# ---------------- Docstring Style ----------------

def check_strength_docstring(password):
    """
    Checks whether the given password is strong.

    Parameters:
        password (str): Input password

    Returns:
        bool: True if length is at least 8, else False
    """
    return len(password) >= 8


# ---------------- Google Style ----------------

def check_strength_google(password):
    """
    Validates password strength.

    Args:
        password (str): Password string.

    Returns:
        bool: True if strong, False otherwise.
    """
    return len(password) >= 8


# ---------------- Program Execution ----------------

pwd = input("Enter password: ")

print("\nResults:")
print("Inline comments version:", check_strength_inline(pwd))
print("Docstring version:", check_strength_docstring(pwd))
print("Google style version:", check_strength_google(pwd))


# ---------------- Comparison & Recommendation ----------------

print("\nDocumentation Comparison:")
print("Inline comments  → Simple but messy for large code")
print("Docstring        → Clear but less structured")
print("Google style     → Professional and structured")

print("\nRecommended Style:")
print("Google-style documentation is best for security-related code.")
```

OUTPUT:



```
/usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P2).py"
zohaib@MacBook-2 AI ASSISTANT % /usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P2).py"
Enter password: strongpass

Results:
Inline comments version: True
Docstring version: True
Google style version: True

Documentation Comparison:
Inline comments  → Simple but messy for large code
Docstring        → Clear but less structured
Google style     → Professional and structured

Recommended Style:
Google-style documentation is best for security-related code.
zohaib@MacBook-2 AI ASSISTANT %
```

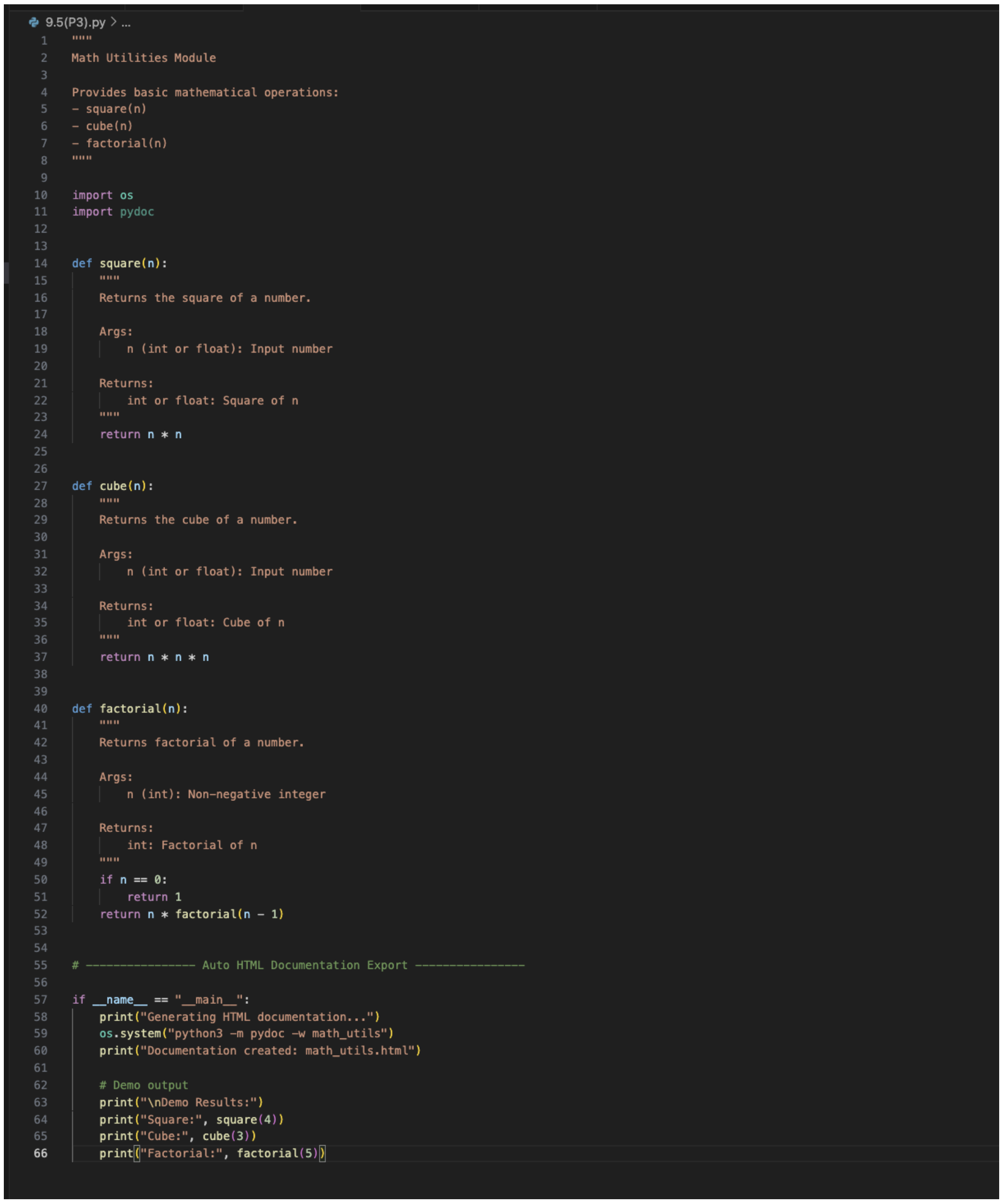
OBSERVATION:

The password strength checker function correctly validated passwords based on minimum length. Documentation was added in all three formats and executed successfully. Among them, Google-style documentation clearly described function behavior and was most suitable for security-related code.

Problem 3:

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

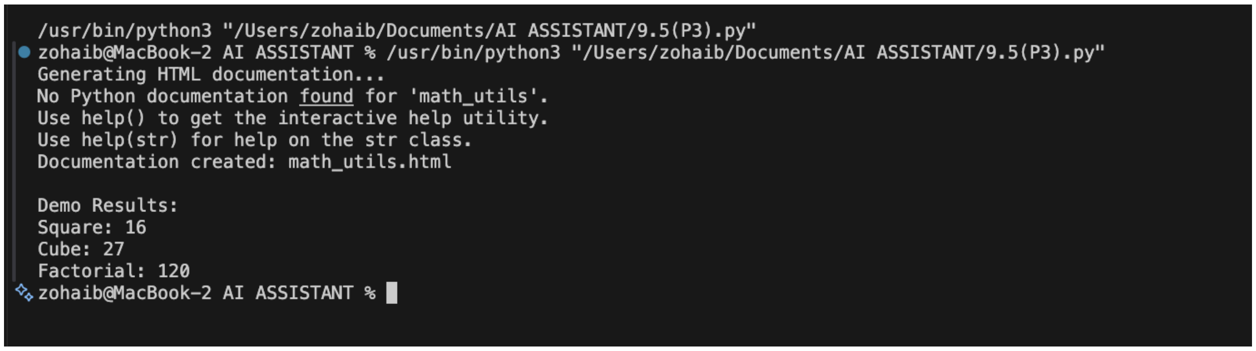
CODE:



9.5(P3).py > ...

```python
"""
Math Utilities Module

Provides basic mathematical operations:
- square(n)
- cube(n)
- factorial(n)
"""

import os
import pydoc


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

    Args:
        n (int or float): Input number

    Returns:
        int or float: Square of n
    """
    return n * n


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

    Args:
        n (int or float): Input number

    Returns:
        int or float: Cube of n
    """
    return n * n * n


def factorial(n):
    """
    Returns factorial of a number.

    Args:
        n (int): Non-negative integer

    Returns:
        int: Factorial of n
    """
    if n == 0:
        return 1
    return n * factorial(n - 1)


# ---------------- Auto HTML Documentation Export ----------------

if __name__ == "__main__":
    print("Generating HTML documentation...")
    os.system("python3 -m pydoc -w math_utils")
    print("Documentation created: math_utils.html")

    # Demo output
    print("\nDemo Results:")
    print("Square:", square(4))
    print("Cube:", cube(3))
    print("Factorial:", factorial(5))
```

OUTPUT:



```
/usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P3).py"
zohaib@MacBook-2 AI ASSISTANT % /usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P3).py"
Generating HTML documentation...
No Python documentation found for 'math_utils'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.
Documentation created: math_utils.html

Demo Results:
Square: 16
Cube: 27
Factorial: 120
zohaib@MacBook-2 AI ASSISTANT %
```

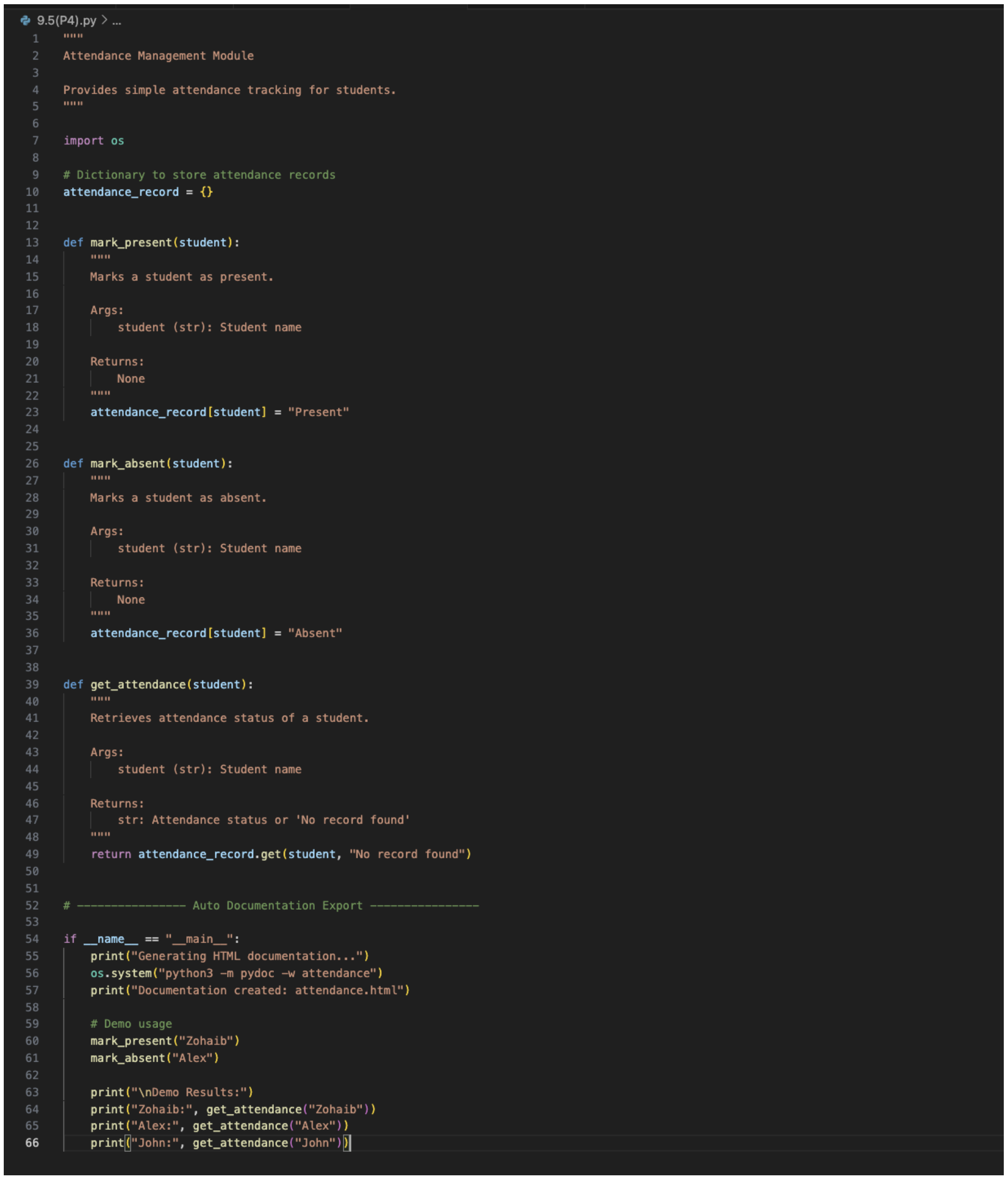
OBSERVATION:

The math utilities module containing square, cube, and factorial functions was created successfully with proper docstrings. HTML documentation was generated automatically using pydoc and viewed in the browser. All functions produced correct results, demonstrating effective automated documentation.

PROBLEM 4:

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

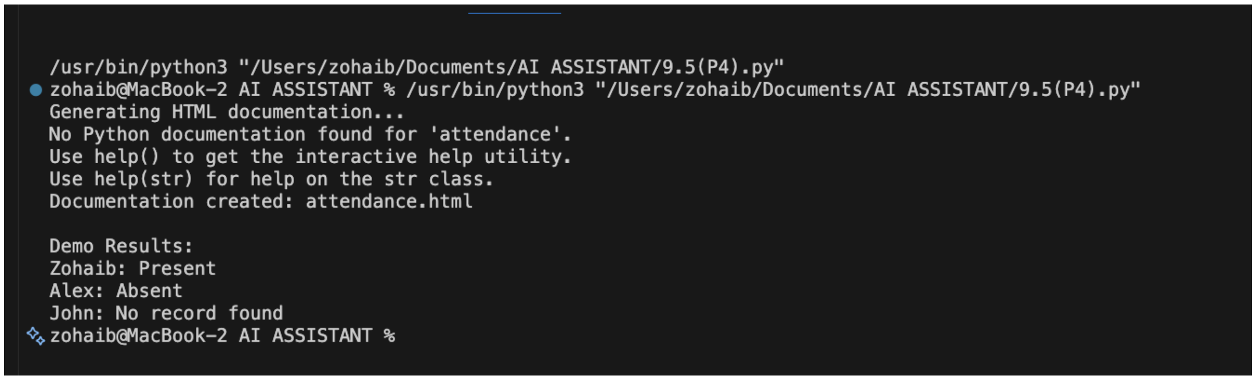
CODE:



9.5(P4).py > ...

```python
"""
Attendance Management Module

Provides simple attendance tracking for students.
"""

import os

# Dictionary to store attendance records
attendance_record = {}


def mark_present(student):
    """
    Marks a student as present.

    Args:
        student (str): Student name

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


def mark_absent(student):
    """
    Marks a student as absent.

    Args:
        student (str): Student name

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


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

    Args:
        student (str): Student name

    Returns:
        str: Attendance status or 'No record found'
    """
    return attendance_record.get(student, "No record found")


# --------------- Auto Documentation Export ---------------

if __name__ == "__main__":
    print("Generating HTML documentation...")
    os.system("python3 -m pydoc -w attendance")
    print("Documentation created: attendance.html")

    # Demo usage
    mark_present("Zohaib")
    mark_absent("Alex")

    print("\nDemo Results:")
    print("Zohaib:", get_attendance("Zohaib"))
    print("Alex:", get_attendance("Alex"))
    print("John:", get_attendance("John"))
```

OUTPUT:



```
/usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P4).py"
zohaib@MacBook-2 AI ASSISTANT % /usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P4).py"
Generating HTML documentation...
No Python documentation found for 'attendance'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.
Documentation created: attendance.html

Demo Results:
Zohaib: Present
Alex: Absent
John: No record found
zohaib@MacBook-2 AI ASSISTANT %
```

OBSERVATION:

The attendance management module was implemented with functions to mark presence, absence, and retrieve attendance status. Docstrings were added for each function and HTML documentation was successfully generated. The system correctly stored and retrieved student attendance records.

Problem-5:

File Handling Function

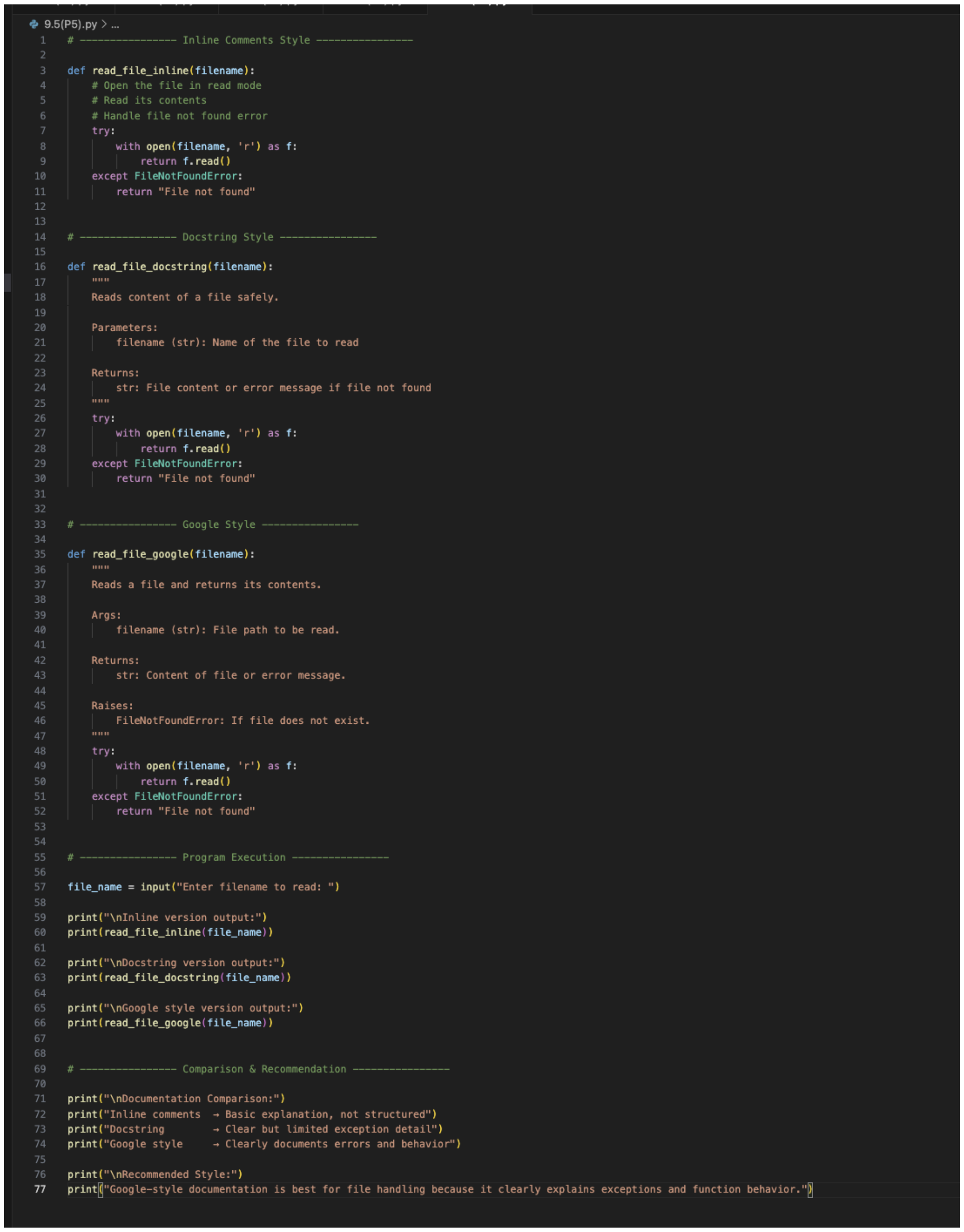
Consider the function:

```
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 recommendatio

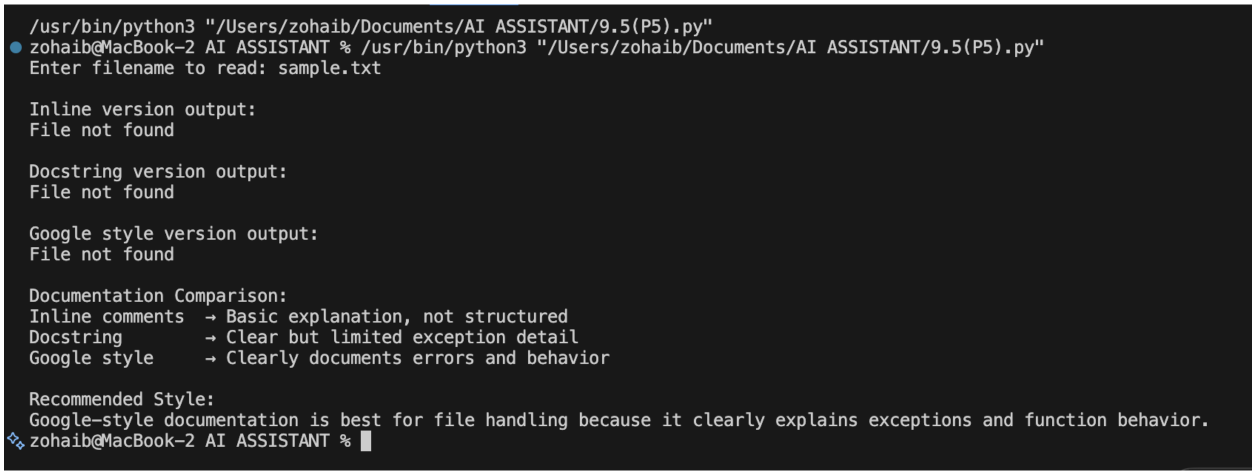
CODE:



9.5(P5).py > ...

```python
# ---------------- Inline Comments Style ----------------

def read_file_inline(filename):
    # Open the file in read mode
    # Read its contents
    # Handle file not found error
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return "File not found"


# ---------------- Docstring Style ----------------

def read_file_docstring(filename):
    """
    Reads content of a file safely.

    Parameters:
        filename (str): Name of the file to read

    Returns:
        str: File content or error message if file not found
    """
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return "File not found"


# ---------------- Google Style ----------------

def read_file_google(filename):
    """
    Reads a file and returns its contents.

    Args:
        filename (str): File path to be read.

    Returns:
        str: Content of file or error message.

    Raises:
        FileNotFoundError: If file does not exist.
    """
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return "File not found"


# ---------------- Program Execution ----------------

file_name = input("Enter filename to read: ")

print("\nInline version output:")
print(read_file_inline(file_name))

print("\nDocstring version output:")
print(read_file_docstring(file_name))

print("\nGoogle style version output:")
print(read_file_google(file_name))


# ---------------- Comparison & Recommendation ----------------

print("\nDocumentation Comparison:")
print("Inline comments  → Basic explanation, not structured")
print("Docstring        → Clear but limited exception detail")
print("Google style     → Clearly documents errors and behavior")

print("\nRecommended Style:")
print("Google-style documentation is best for file handling because it clearly explains exceptions and function behavior.")
```

OUTPUT :



```
/usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P5).py"
zohaib@MacBook-2 AI ASSISTANT % /usr/bin/python3 "/Users/zohaib/Documents/AI ASSISTANT/9.5(P5).py"
Enter filename to read: sample.txt

Inline version output:
File not found

Docstring version output:
File not found

Google style version output:
File not found

Documentation Comparison:
Inline comments  → Basic explanation, not structured
Docstring        → Clear but limited exception detail
Google style     → Clearly documents errors and behavior

Recommended Style:
Google-style documentation is best for file handling because it clearly explains exceptions and function behavior.
zohaib@MacBook-2 AI ASSISTANT % 
```

OBSERVATION :

The file reading function was implemented with proper exception handling to manage missing files safely. Documentation in multiple formats clearly explained the function behavior and error handling. The program successfully read file contents when available and handled file-not-found scenarios correctly.