ASSIGNMENT:09.5

HALLTICKET:2303A51598

BATCH:29

SHAIK NEHA FARIYAL

Problem 1: String Utilities Function:

PROMPT:

Consider the following Python function:

def reverse_string(text):

return text[::-1]

Task:

1. Write documentation in:

o (a) Docstring

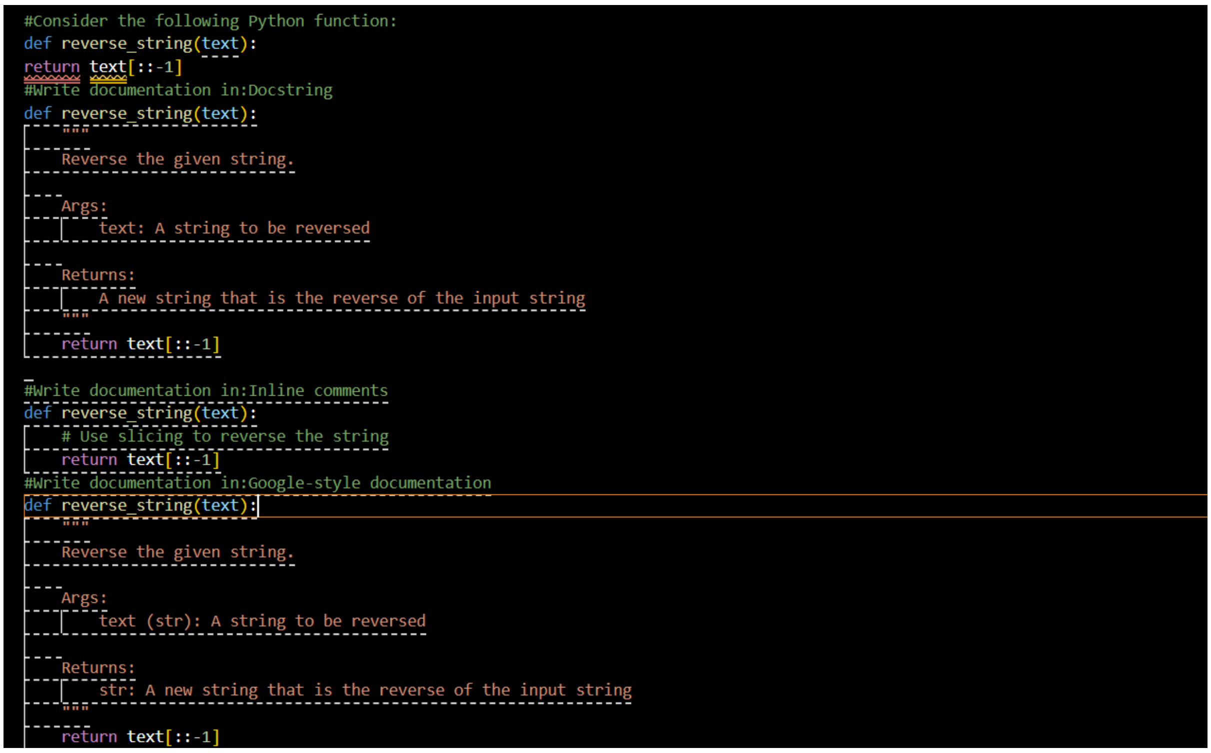
o (b) Inline comments

o (c) Google-style documentation

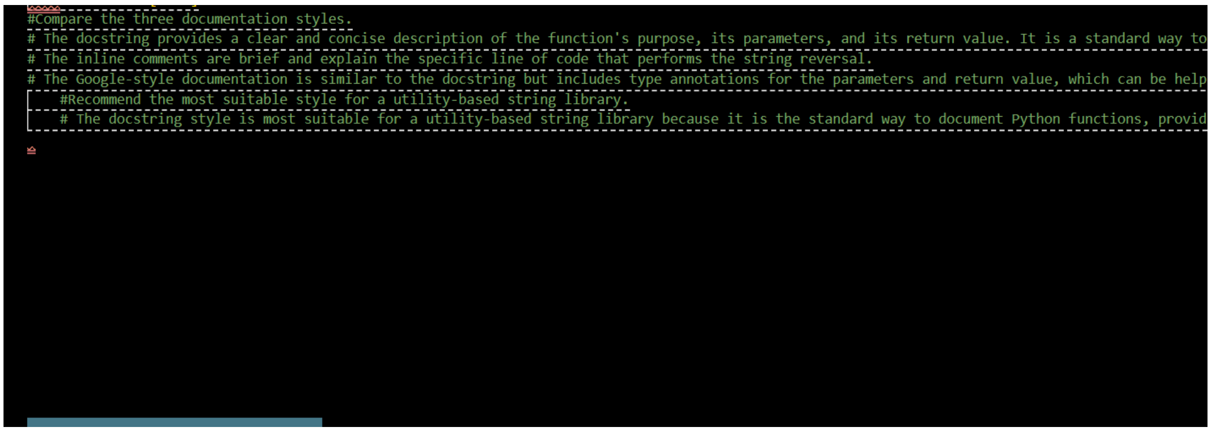
2. Compare the three documentation styles.

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

library



```
#Consider the following Python function:
def reverse_string(text):
return text[::-1]
#Write documentation in:Docstring
def reverse_string(text):
    """
    Reverse the given string.

    Args:
        text: A string to be reversed

    Returns:
        A new string that is the reverse of the input string
    """
    return text[::-1]

#Write documentation in:Inline comments
def reverse_string(text):
    # Use slicing to reverse the string
    return text[::-1]
#Write documentation in:Google-style documentation
def reverse_string(text):
    """
    Reverse the given string.

    Args:
        text (str): A string to be reversed

    Returns:
        str: A new string that is the reverse of the input string
    """
    return text[::-1]
```



```
#Compare the three documentation styles.
# The docstring provides a clear and concise description of the function's purpose, its parameters, and its return value. It is a standard way to
# The inline comments are brief and explain the specific line of code that performs the string reversal.
# The Google-style documentation is similar to the docstring but includes type annotations for the parameters and return value, which can be help
    #Recommend the most suitable style for a utility-based string library.
    # The docstring style is most suitable for a utility-based string library because it is the standard way to document Python functions, provid
```

JUSTIFICATION:

This task Google-style documentation is most suitable for a utility-based string library because it clearly defines parameters and return values in a structured format. It improves readability and supports automatic documentation generation tools. It is widely used in professional development environments. Therefore, it enhances maintainability and scalability of utility libraries.

Problem 2: Password Strength Checker

PROMPT:

Consider the function:

def check_strength(password):

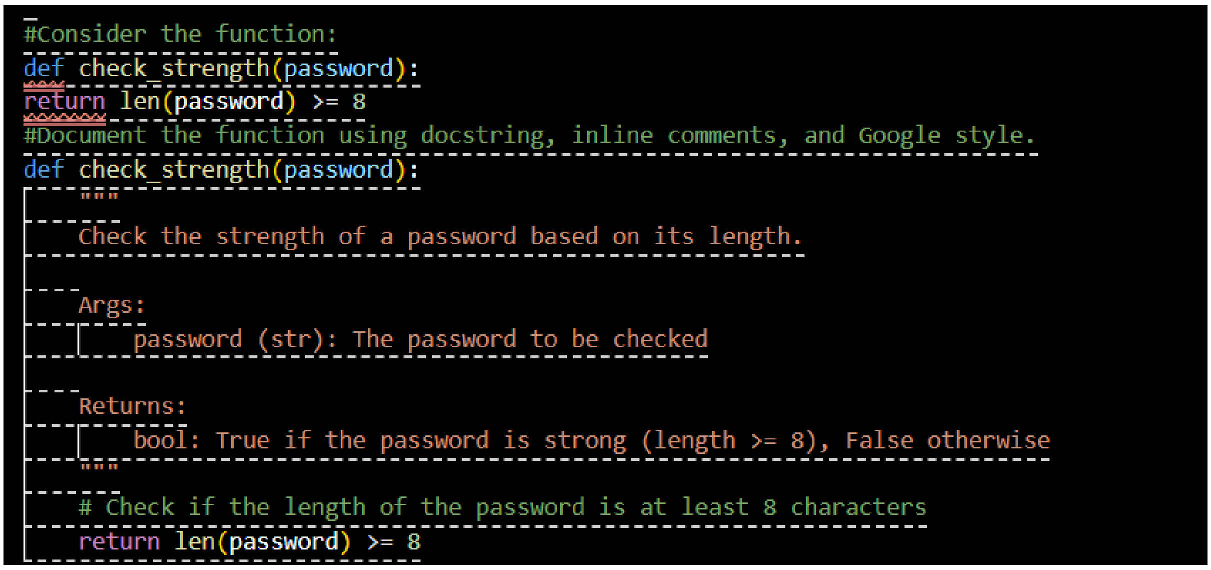
return len(password) >= 8

Task:

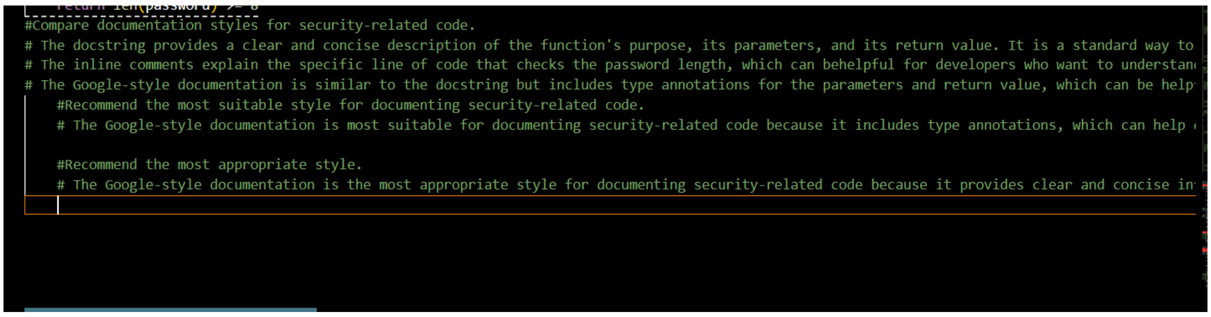
1. Document the function using docstring, inline comments, and Google style.

2. Compare documentation styles for security-related code.

3. Recommend the most appropriate style.



```python
#Consider the function:
def check_strength(password):
return len(password) >= 8
#Document the function using docstring, inline comments, and Google style.
def check_strength(password):
    """
    Check the strength of a password based on its length.

    Args:
        password (str): The password to be checked

    Returns:
        bool: True if the password is strong (length >= 8), False otherwise
    """
    # Check if the length of the password is at least 8 characters
    return len(password) >= 8
```



```python
#Compare documentation styles for security-related code.
# The docstring provides a clear and concise description of the function's purpose, its parameters, and its return value. It is a standard way to
# The inline comments explain the specific line of code that checks the password length, which can behelpful for developers who want to understan
# The Google-style documentation is similar to the docstring but includes type annotations for the parameters and return value, which can be help
    #Recommend the most suitable style for documenting security-related code.
    # The Google-style documentation is most suitable for documenting security-related code because it includes type annotations, which can help

    #Recommend the most appropriate style.
    # The Google-style documentation is the most appropriate style for documenting security-related code because it provides clear and concise in
```

JUSTIFICATION:

For security-related code, structured documentation like Google-style is preferred because it clearly explains validation logic and return behavior. Security functions require precise parameter and output descriptions to avoid misuse. Proper documentation also helps in auditing and maintenance. Hence, structured documentation improves reliability and clarity.

Problem 3: Math Utilities Module

PROMPT:

Task:

1. Create a module math_utils.py with functions:

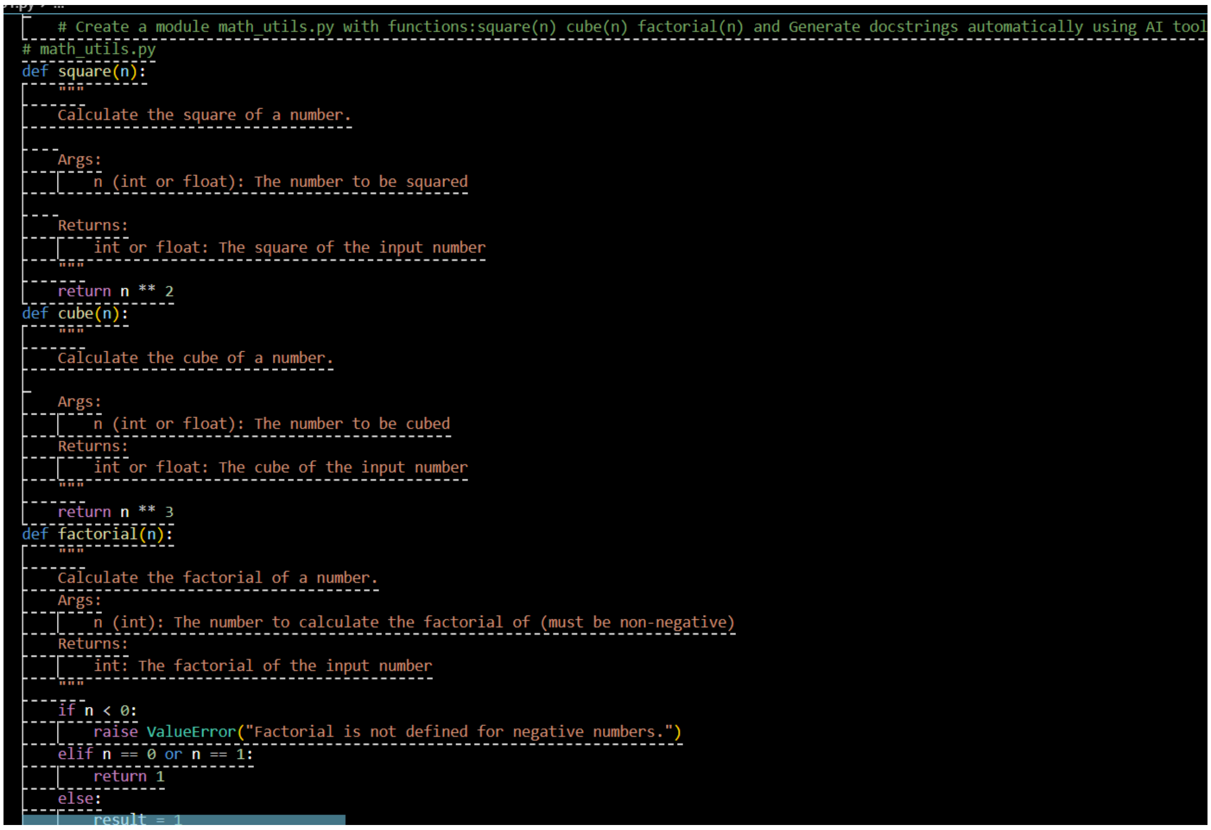
o square(n)

o cube(n)

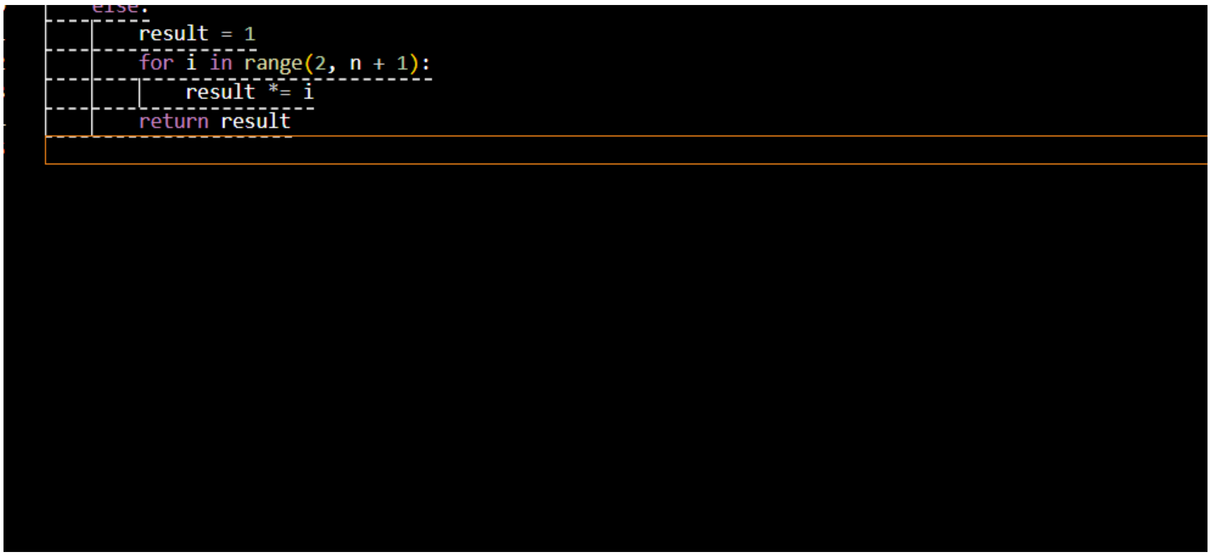
o factorial(n)

2. Generate docstrings automatically using AI tools.

3. Export documentation as an HTML file.



```
    # Create a module math_utils.py with functions:square(n) cube(n) factorial(n) and Generate docstrings automatically using AI tool
# math_utils.py
def square(n):
    """
    Calculate the square of a number.

    Args:
        n (int or float): The number to be squared

    Returns:
        int or float: The square of the input number
    """
    return n ** 2
def cube(n):
    """
    Calculate the cube of a number.

    Args:
        n (int or float): The number to be cubed
    Returns:
        int or float: The cube of the input number
    """
    return n ** 3
def factorial(n):
    """
    Calculate the factorial of a number.
    Args:
        n (int): The number to calculate the factorial of (must be non-negative)
    Returns:
        int: The factorial of the input number
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
```



```
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
```

JUSTIFICATION:

Documenting mathematical functions improves clarity about input constraints and expected outputs. It helps prevent misuse, especially in functions like factorial where invalid inputs may cause errors. Automatically generated docstrings save time and maintain consistency. Exporting HTML documentation improves accessibility and professional presentation.

Problem 4: Attendance Management Module

Prompt:

1. Create a module attendance.py with functions:

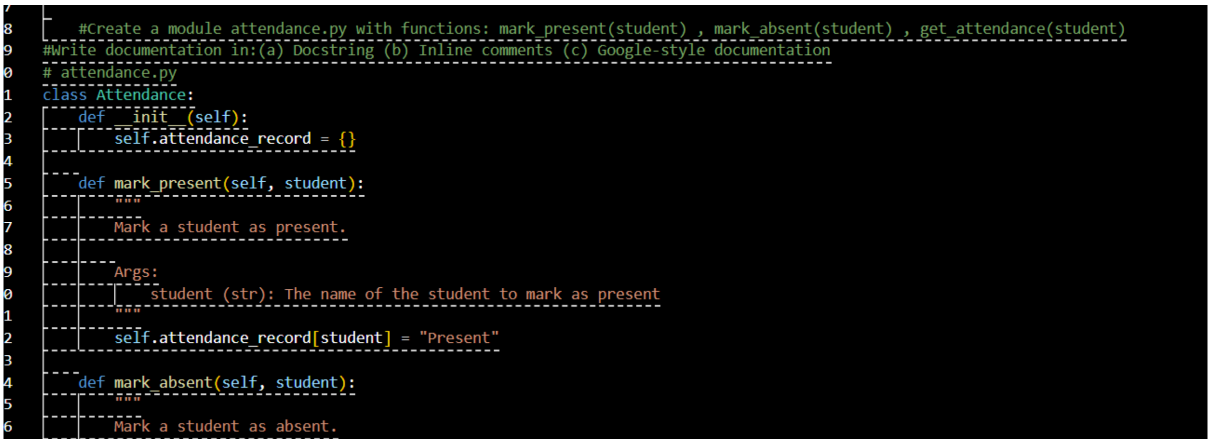
o mark_present(student)

o mark_absent(student)

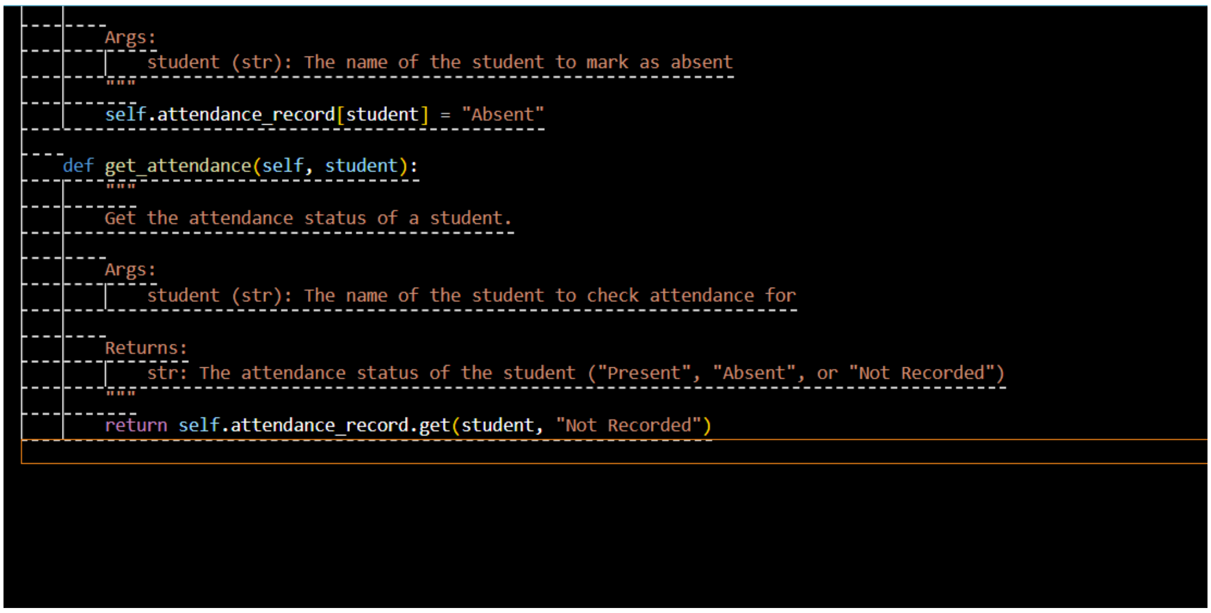
o get_attendance(student)

2. Add proper docstrings.

3. Generate and view documentation in terminal and browser



```python
    #Create a module attendance.py with functions: mark_present(student) , mark_absent(student) , get_attendance(student)
#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
# attendance.py
class Attendance:
    def __init__(self):
        self.attendance_record = {}

    def mark_present(self, student):
        """
        Mark a student as present.

        Args:
            student (str): The name of the student to mark as present
        """
        self.attendance_record[student] = "Present"

    def mark_absent(self, student):
        """
        Mark a student as absent.
```



```python
        Args:
            student (str): The name of the student to mark as absent
        """
        self.attendance_record[student] = "Absent"

    def get_attendance(self, student):
        """
        Get the attendance status of a student.

        Args:
            student (str): The name of the student to check attendance for

        Returns:
            str: The attendance status of the student ("Present", "Absent", or "Not Recorded")
        """
        return self.attendance_record.get(student, "Not Recorded")
```

JUSTIFICATIONS:

Attendance systems require clear documentation to define function behavior and data handling. Proper docstrings improve maintainability and team collaboration. Generated documentation allows easy viewing in terminal and browser. This ensures better usability and transparency in academic or organizational systems.

Problem 5: File Handling Function

Consider the function:

def read_file(filename):

with open(filename, 'r') as f:

return f.read()

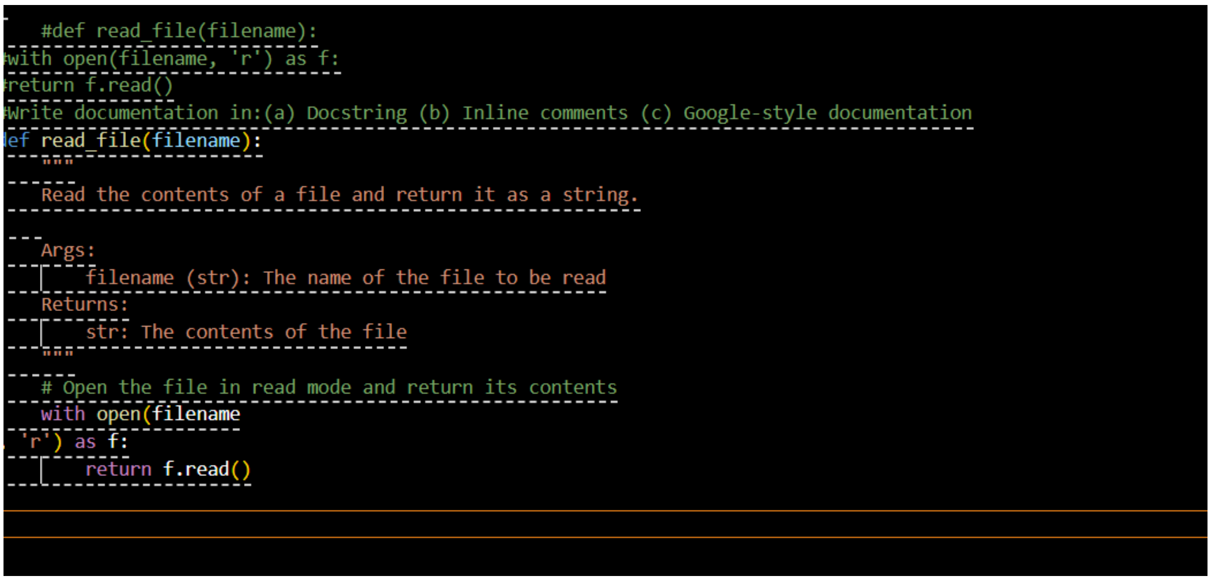
PROMPT:

def read_file(filename):

with open(filename, 'r') as f:

return f.read()

Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation



```
    #def read_file(filename):
#with open(filename, 'r') as f:
#return f.read()
#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
def read_file(filename):
    """
    Read the contents of a file and return it as a string.

    Args:
        filename (str): The name of the file to be read
    Returns:
        str: The contents of the file
    """
    # Open the file in read mode and return its contents
    with open(filename, 'r') as f:
        return f.read()
```

JUSTIFICATION:

This task Google-style documentation best explains exception handling because it clearly defines raised exceptions and return values. File operations are prone to errors like FileNotFoundError, so structured documentation improves understanding. It helps developers anticipate edge cases. Therefore, it enhances robustness and maintainability of file-handling code.