Assignment – 9.5

2303A51176

Task 1 – String Utilities Function

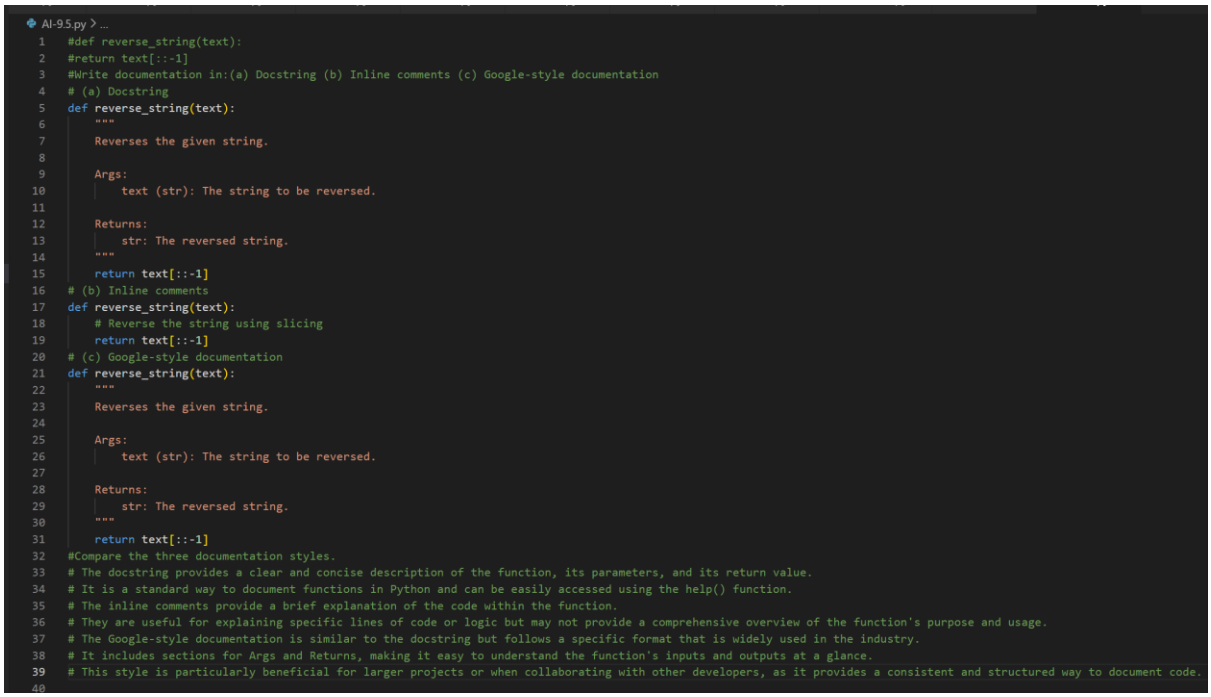
PROMPT :

#def reverse_string(text):

#return text[::-1]

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

CODE :



```
AI-9.5.py > ...
#def reverse_string(text):
#return text[::-1]
#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
# (a) Docstring
def reverse_string(text):
    """
    Reverses the given string.

    Args:
        text (str): The string to be reversed.

    Returns:
        str: The reversed string.
    """
    return text[::-1]
# (b) Inline comments
def reverse_string(text):
    # Reverse the string using slicing
    return text[::-1]
# (c) Google-style documentation
def reverse_string(text):
    """
    Reverses the given string.

    Args:
        text (str): The string to be reversed.

    Returns:
        str: The reversed string.
    """
    return text[::-1]
#Compare the three documentation styles.
# The docstring provides a clear and concise description of the function, its parameters, and its return value.
# It is a standard way to document functions in Python and can be easily accessed using the help() function.
# The inline comments provide a brief explanation of the code within the function.
# They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the function's purpose and usage.
# The Google-style documentation is similar to the docstring but follows a specific format that is widely used in the industry.
# It includes sections for Args and Returns, making it easy to understand the function's inputs and outputs at a glance.
# This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document code.
```

JUSTIFICATION :

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.

Task 2 – Password Strength Checker

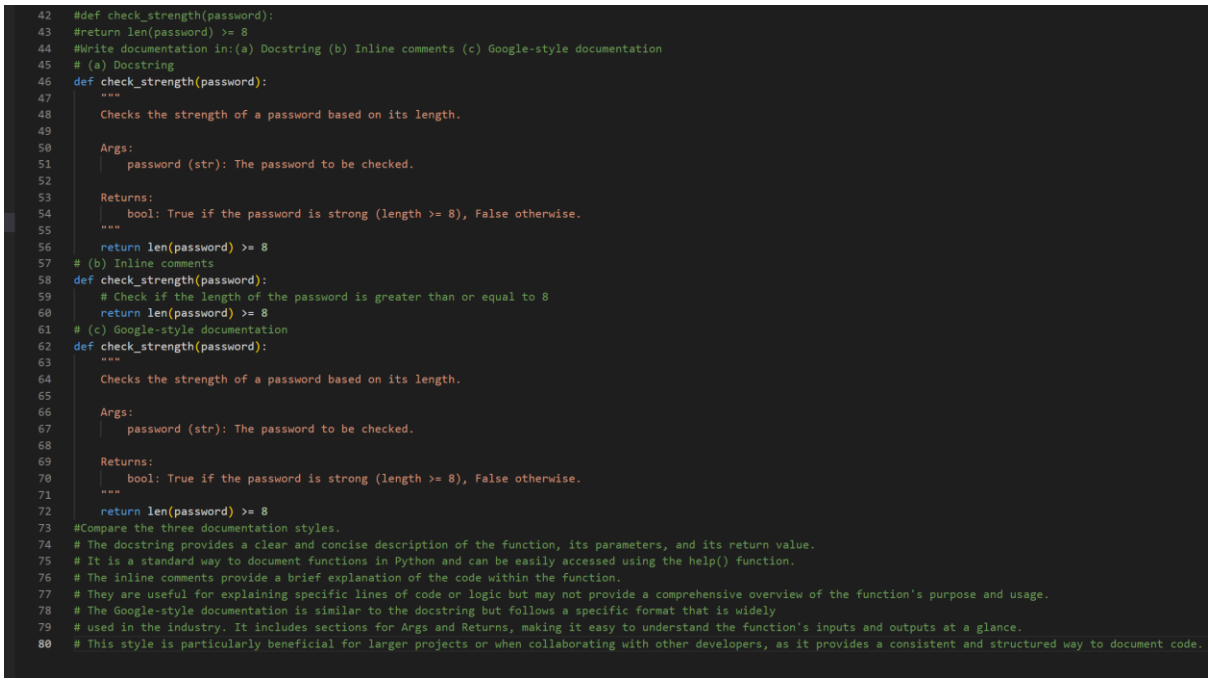
PROMPT :

#def check_strength(password):

#return len(password) >= 8

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

CODE :



```python
#def check_strength(password):
#return len(password) >= 8
#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
# (a) Docstring
def check_strength(password):
    """
    Checks 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.
    """
    return len(password) >= 8
# (b) Inline comments
def check_strength(password):
    # Check if the length of the password is greater than or equal to 8
    return len(password) >= 8
# (c) Google-style documentation
def check_strength(password):
    """
    Checks 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.
    """
    return len(password) >= 8
#Compare the three documentation styles.
# The docstring provides a clear and concise description of the function, its parameters, and its return value.
# It is a standard way to document functions in Python and can be easily accessed using the help() function.
# The inline comments provide a brief explanation of the code within the function.
# They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the function's purpose and usage.
# The Google-style documentation is similar to the docstring but follows a specific format that is widely
# used in the industry. It includes sections for Args and Returns, making it easy to understand the function's inputs and outputs at a glance.
# This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document code.
```

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.

Task 3 – Math Utilities Module

PROMPT :

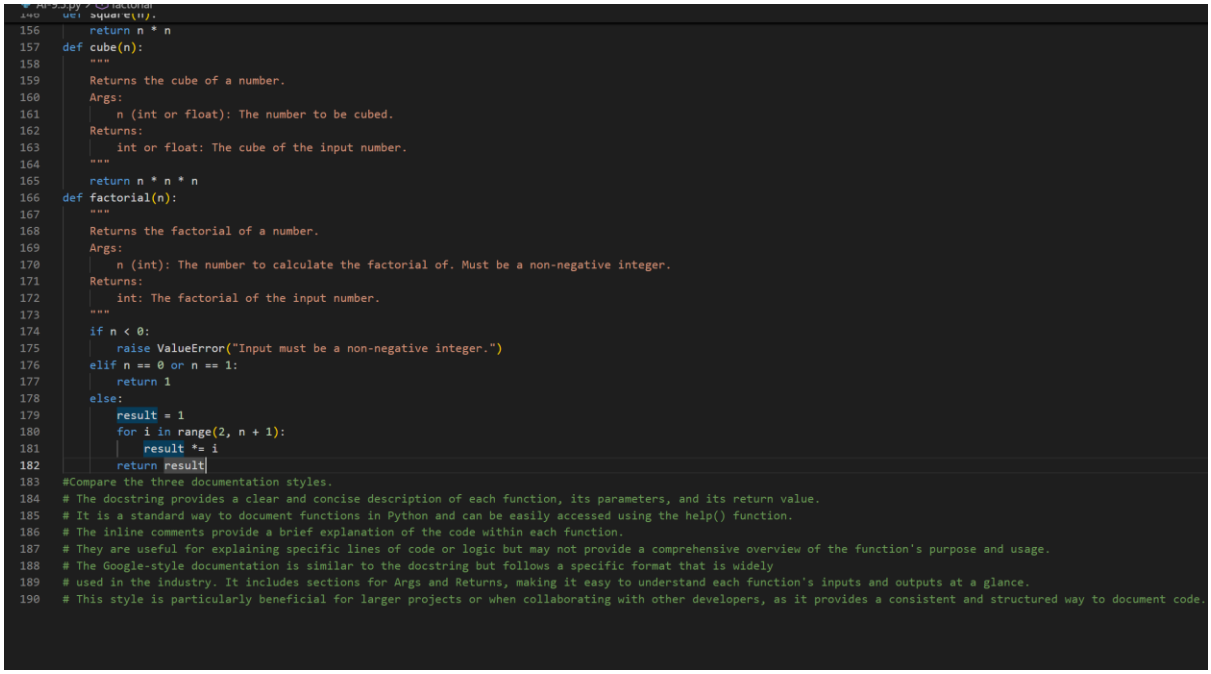
#Create a module math_utils.py with functions: square(n), cube(n), factorial(n)

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

CODE :

```python
def square(n):
    # Return the square of the input number
    return n * n
def cube(n):
    # Return the cube of the input number
    return n * n * n
def factorial(n):
    # Check if the input number is negative
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    # Return 1 for factorial of 0 and 1
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        # Calculate factorial using a loop
        for i in range(2, n + 1):
            result *= i
        return result
# (c) Google-style documentation
def square(n):
    """
    Returns 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 * n
def cube(n):
    """
    Returns 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 * n * n
def factorial(n):
    """
    Returns the factorial of a number.
```

```python

#Create a module math_utils.py with functions: square(n), cube(n), factorial(n)
#Write documentation in:(a) Docstring (b) Inline comments (c) Google-style documentation
# math_utils.py
# (a) Docstring
def square(n):
    """
    Returns 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 * n
def cube(n):
    """
    Returns 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 * n * n
def factorial(n):
    """
    Returns the factorial of a number.
    Args:
        n (int): The number to calculate the factorial of. Must be a non-negative integer.
    Returns:
        int: The factorial of the input number.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# (b) Inline comments
```



```
    return n * n
def cube(n):
    """
    Returns 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 * n * n
def factorial(n):
    """
    Returns the factorial of a number.
    Args:
        n (int): The number to calculate the factorial of. Must be a non-negative integer.
    Returns:
        int: The factorial of the input number.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
#Compare the three documentation styles.
# The docstring provides a clear and concise description of each function, its parameters, and its return value.
# It is a standard way to document functions in Python and can be easily accessed using the help() function.
# The inline comments provide a brief explanation of the code within each function.
# They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the function's purpose and usage.
# The Google-style documentation is similar to the docstring but follows a specific format that is widely
# used in the industry. It includes sections for Args and Returns, making it easy to understand each function's inputs and outputs at a glance.
# This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document code.
```

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.

Task 4 – Attendance Management Module

PROMPT :

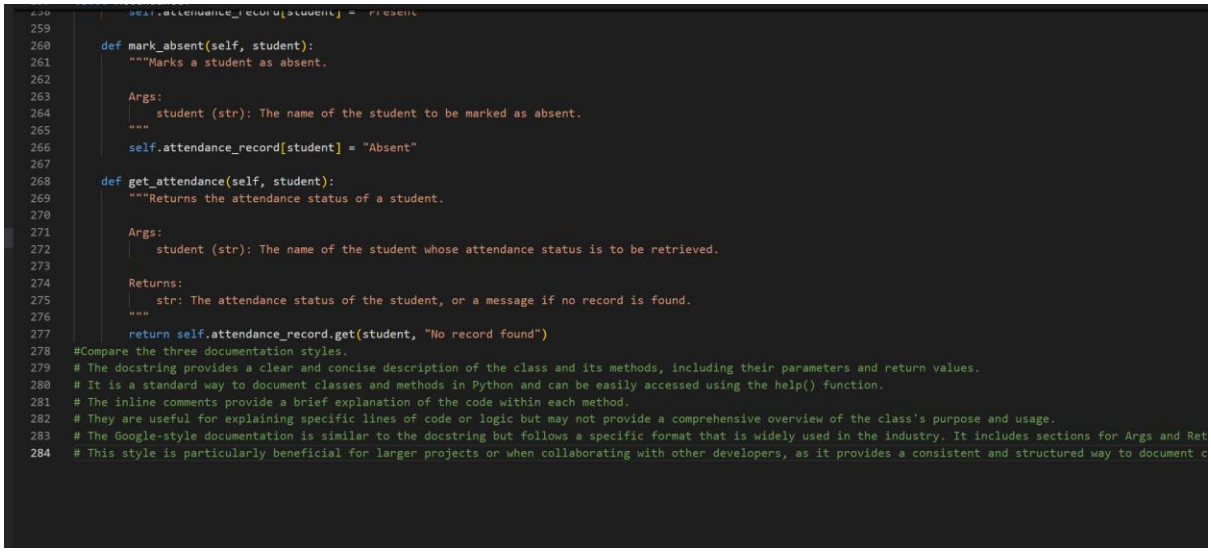
#Create a module attendance.py with functions: mark_present(student) , mark_absent(student) , get_attendance(student)

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

CODE :

```python
        return self.attendance_record.get(student, "No record found")
# (c) Google-style documentation
class Attendance:
    """
    A class to manage student attendance.

    Methods:
        mark_present(student): Marks a student as present.
        mark_absent(student): Marks a student as absent.
        get_attendance(student): Returns the attendance status of a student.
    """
    def __init__(self):
        """Initializes an empty attendance record."""
        self.attendance_record = {}

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

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

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

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

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

        Args:
            student (str): The name of the student whose attendance status is to be retrieved.

        Returns:
            str: The attendance status of the student, or a message if no record is found.
        """
        return self.attendance_record.get(student, "No record found")
#Compare the three documentation styles.
# The docstring provides a clear and concise description of the class and its methods, including their parameters and return values.
```

```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
# (a) Docstring
class Attendance:
    """
    A class to manage student attendance.

    Methods:
        mark_present(student): Marks a student as present.
        mark_absent(student): Marks a student as absent.
        get_attendance(student): Returns the attendance status of a student.
    """
    def __init__(self):
        self.attendance_record = {}

    def mark_present(self, student):
        """Marks a student as present."""
        self.attendance_record[student] = "Present"

    def mark_absent(self, student):
        """Marks a student as absent."""
        self.attendance_record[student] = "Absent"

    def get_attendance(self, student):
        """Returns the attendance status of a student."""
        return self.attendance_record.get(student, "No record found")
# (b) Inline comments
class Attendance:
    def __init__(self):
        # Initialize an empty dictionary to store attendance records
        self.attendance_record = {}

    def mark_present(self, student):
        # Mark the student as present in the attendance record
        self.attendance_record[student] = "Present"

    def mark_absent(self, student):
        # Mark the student as absent in the attendance record
        self.attendance_record[student] = "Absent"

    def get_attendance(self, student):
        # Return the attendance status of the student, or a message if no record is found
```



```
        self.attendance_record[student] = "Present"

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

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

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

        Args:
            student (str): The name of the student whose attendance status is to be retrieved.

        Returns:
            str: The attendance status of the student, or a message if no record is found.
        """
        return self.attendance_record.get(student, "No record found")
#Compare the three documentation styles.
# The docstring provides a clear and concise description of the class and its methods, including their parameters and return values.
# It is a standard way to document classes and methods in Python and can be easily accessed using the help() function.
# The inline comments provide a brief explanation of the code within each method.
# They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the class's purpose and usage.
# The Google-style documentation is similar to the docstring but follows a specific format that is widely used in the industry. It includes sections for Args and Ret
# This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document c
```

JUSTIFICATION :

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.

Task 5 – File Handling Function

PROMPT :

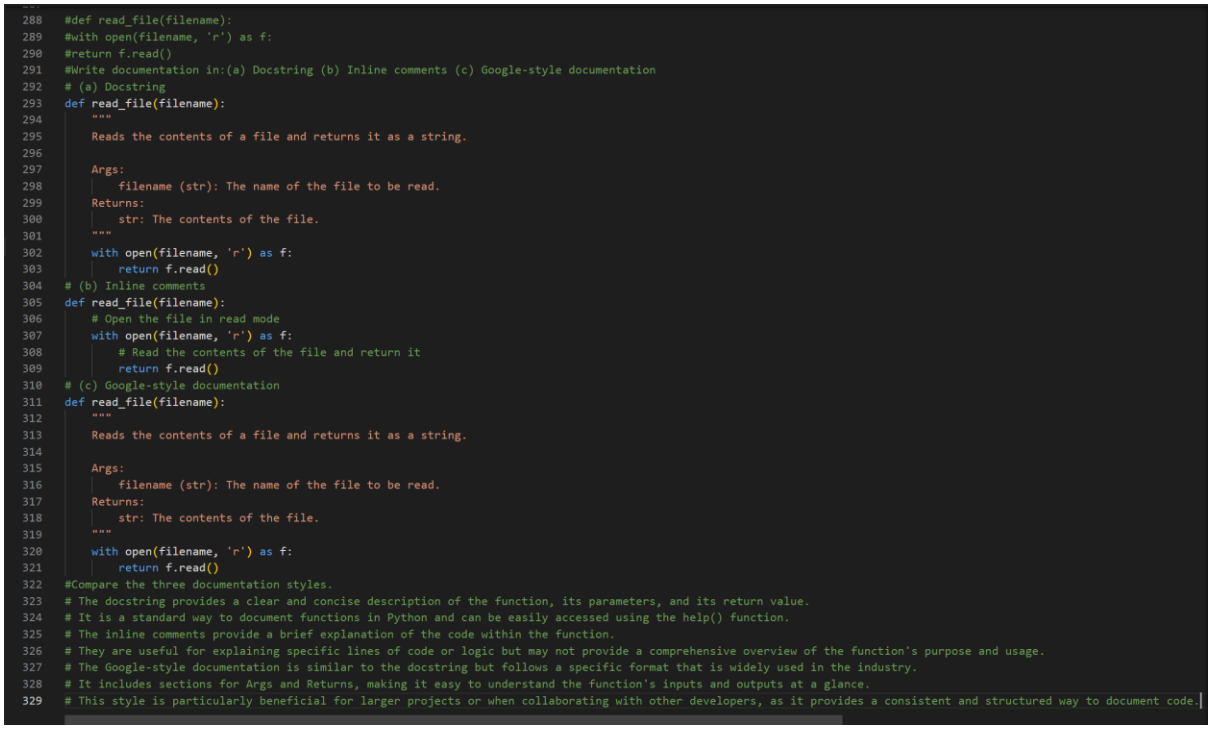
#def read_file(filename):

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

#return f.read()

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

CODE :



```python
#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
# (a) Docstring
def read_file(filename):
    """
    Reads the contents of a file and returns it as a string.

    Args:
        filename (str): The name of the file to be read.
    Returns:
        str: The contents of the file.
    """
    with open(filename, 'r') as f:
        return f.read()
# (b) Inline comments
def read_file(filename):
    # Open the file in read mode
    with open(filename, 'r') as f:
        # Read the contents of the file and return it
        return f.read()
# (c) Google-style documentation
def read_file(filename):
    """
    Reads the contents of a file and returns it as a string.

    Args:
        filename (str): The name of the file to be read.
    Returns:
        str: The contents of the file.
    """
    with open(filename, 'r') as f:
        return f.read()
#Compare the three documentation styles.
# The docstring provides a clear and concise description of the function, its parameters, and its return value.
# It is a standard way to document functions in Python and can be easily accessed using the help() function.
# The inline comments provide a brief explanation of the code within the function.
# They are useful for explaining specific lines of code or logic but may not provide a comprehensive overview of the function's purpose and usage.
# The Google-style documentation is similar to the docstring but follows a specific format that is widely used in the industry.
# It includes sections for Args and Returns, making it easy to understand the function's inputs and outputs at a glance.
# This style is particularly beneficial for larger projects or when collaborating with other developers, as it provides a consistent and structured way to document code.
```

JUSTIFICATION :

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.