ASSIGNMENT-9.5

NAME : U.VIGNESH

HT.NO-2303A51964

BATCH-24

Problem 1: String Utilities Function

Consider the following Python function:

def reverse_string(text):

return text[::-1]

Task:

1. Write documentation in:

o (a) Docstring

o (b) Inline comments

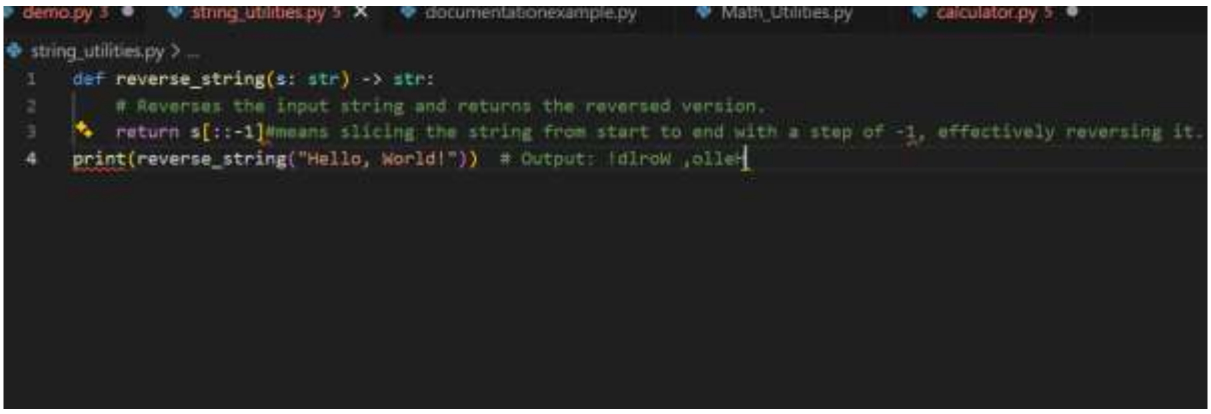
o (c) Google-style documentation

2. Compare the three documentation styles.

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

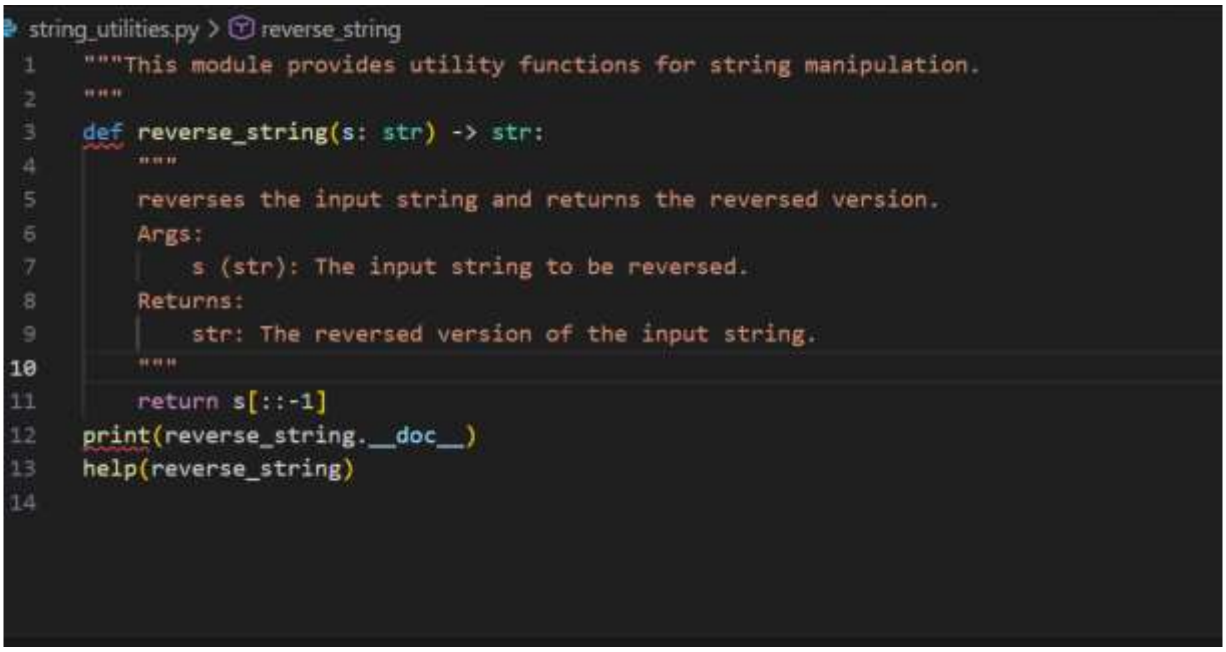
library.

INLINE:



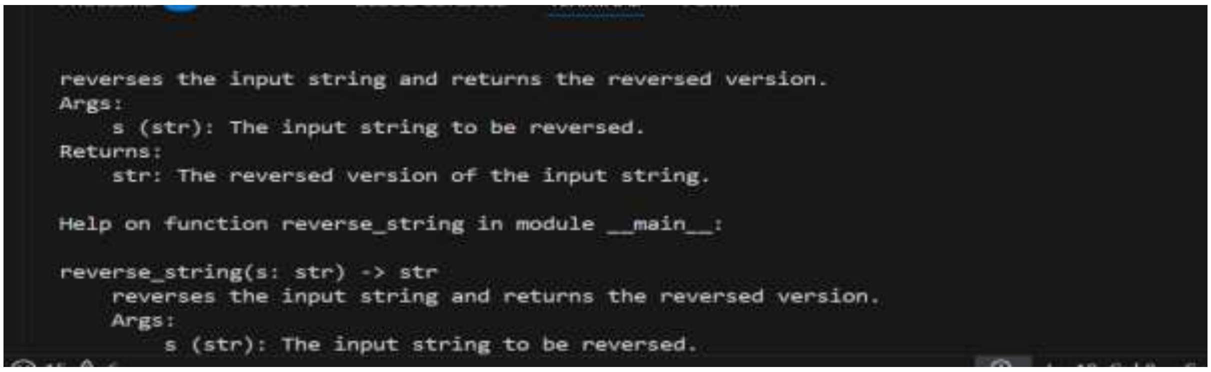
```
def reverse_string(s: str) -> str:
    # Reverses the input string and returns the reversed version.
    return s[::-1]#means slicing the string from start to end with a step of -1, effectively reversing it.
print(reverse_string("Hello, World!"))  # Output: !dlroW ,olleH
```

DocString:



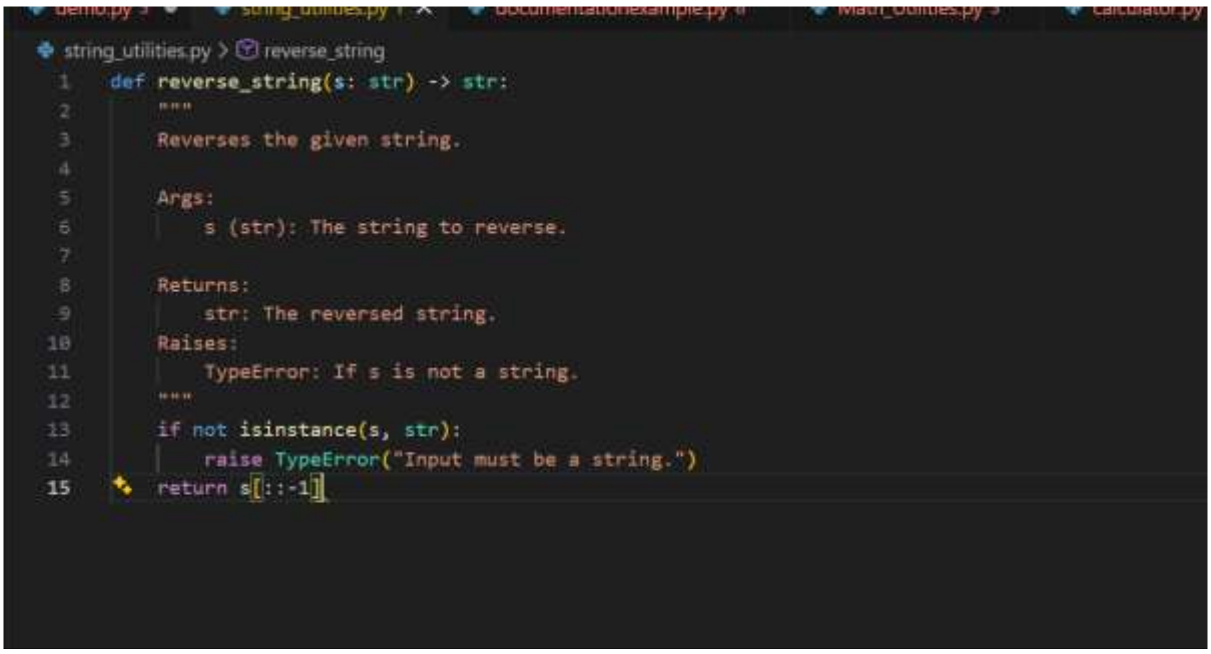
string_utilities.py > reverse_string

```python
"""This module provides utility functions for string manipulation.
"""
def reverse_string(s: str) -> str:
    """
    reverses the input string and returns the reversed version.
    Args:
        s (str): The input string to be reversed.
    Returns:
        str: The reversed version of the input string.
    """
    return s[::-1]
print(reverse_string.__doc__)
help(reverse_string)
```



```
reverses the input string and returns the reversed version.
Args:
    s (str): The input string to be reversed.
Returns:
    str: The reversed version of the input string.

Help on function reverse_string in module __main__:

reverse_string(s: str) -> str
    reverses the input string and returns the reversed version.
    Args:
        s (str): The input string to be reversed.
```

Google Style Documentation:



string_utilities.py > reverse_string

```python
def reverse_string(s: str) -> str:
    """
    Reverses the given string.

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

    Returns:
        str: The reversed string.
    Raises:
        TypeError: If s is not a string.
    """
    if not isinstance(s, str):
        raise TypeError("Input must be a string.")
    return s[::-1]
```

Comparing Three documentation styles.

The three documentation styles for the reverse_string function are all valid and provide useful information about the function.
The first style is concise and straightforward, but it lacks details about the parameters and return value.
The second style is more detailed and includes information about the parameters and return value, but it does not mention any potential exceptions that could be raised.
The third style is the most comprehensive, as it includes information about the parameters, return value, and potential exceptions that could be raised.
However, the third style is the most comprehensive and follows best practices for documentation.

Problem 2: Password Strength Checker

Consider the function:

def check_strength(password):

return len(password) >= 8

Task:

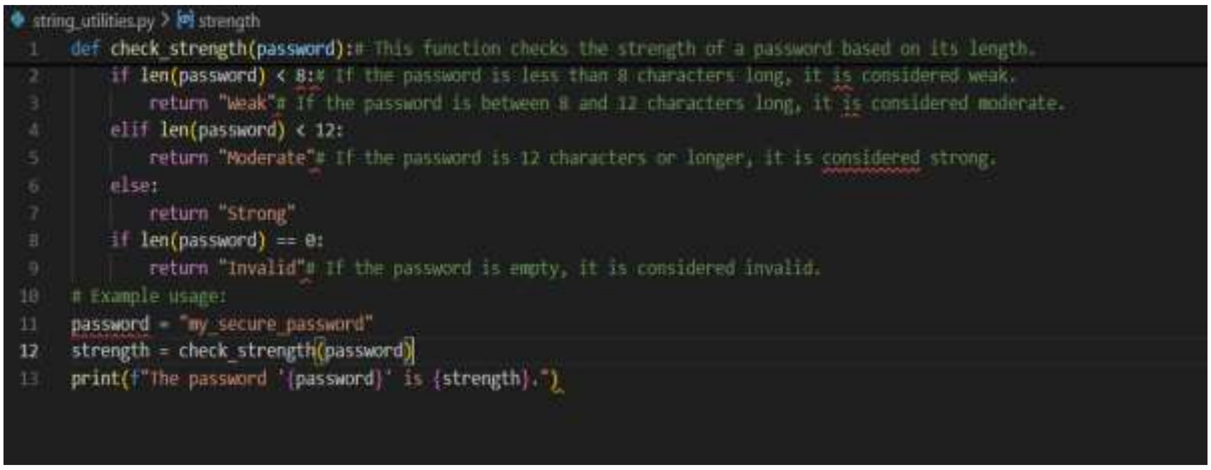
1. Document the function using docstring, inline comments, and

Google style.

2. Compare documentation styles for security-related code.

3. Recommend the most appropriate style.

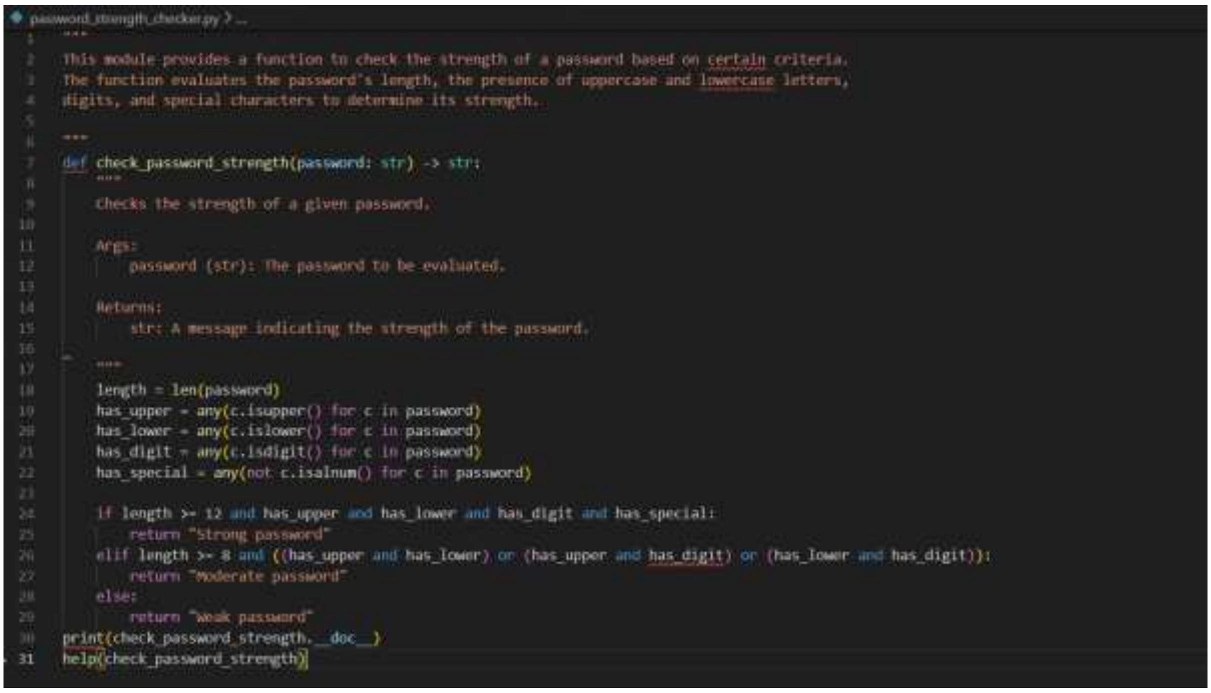
Inline:



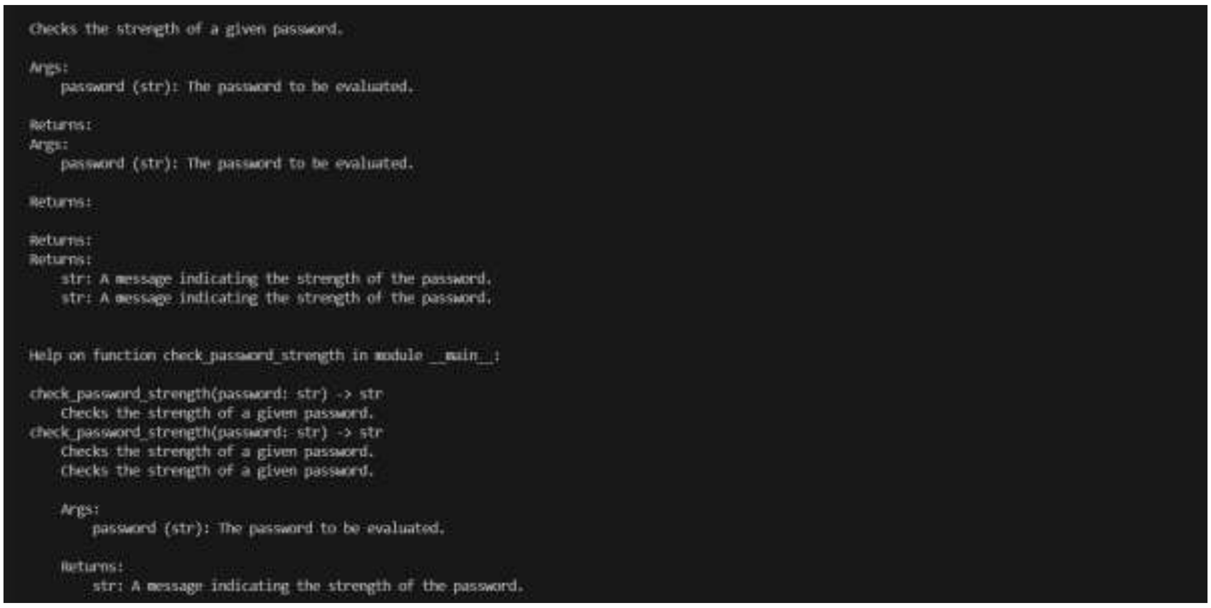
string_utilities.py > strength

```python
def check_strength(password):# This function checks the strength of a password based on its length.
    if len(password) < 8:# If the password is less than 8 characters long, it is considered weak.
        return "Weak"# If the password is between 8 and 12 characters long, it is considered moderate.
    elif len(password) < 12:
        return "Moderate"# If the password is 12 characters or longer, it is considered strong.
    else:
        return "Strong"
    if len(password) == 0:
        return "Invalid"# If the password is empty, it is considered invalid.
# Example usage:
password = "my_secure_password"
strength = check_strength(password)
print(f"The password '{password}' is {strength}.")
```

Docstring:

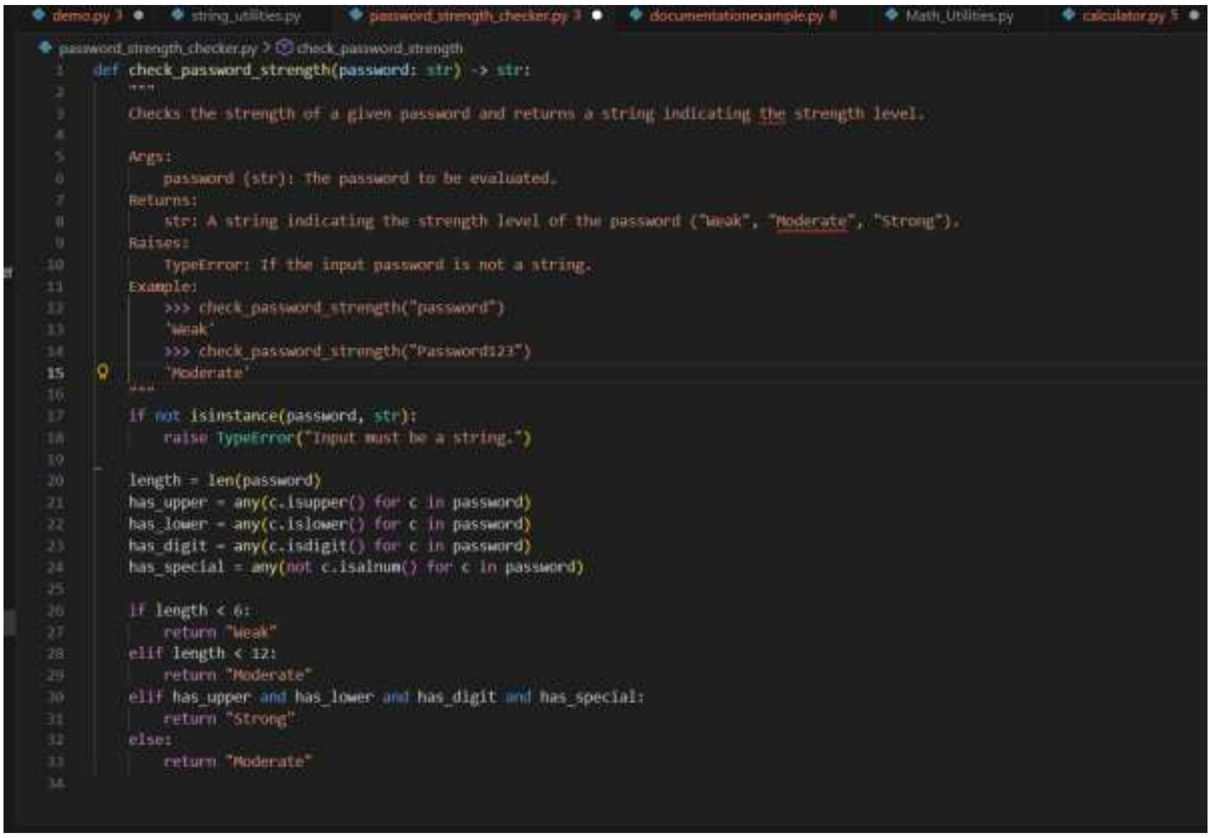

password_strength_checker.py > ...

```python
"""
This module provides a function to check the strength of a password based on certain criteria.
The function evaluates the password's length, the presence of uppercase and lowercase letters,
digits, and special characters to determine its strength.

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

    Args:
        password (str): The password to be evaluated.

    Returns:
        str: A message indicating the strength of the password.

    """
    length = len(password)
    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(not c.isalnum() for c in password)

    if length >= 12 and has_upper and has_lower and has_digit and has_special:
        return "Strong password"
    elif length >= 8 and ((has_upper and has_lower) or (has_upper and has_digit) or (has_lower and has_digit)):
        return "Moderate password"
    else:
        return "Weak password"
print(check_password_strength.__doc__)
help(check_password_strength)
```

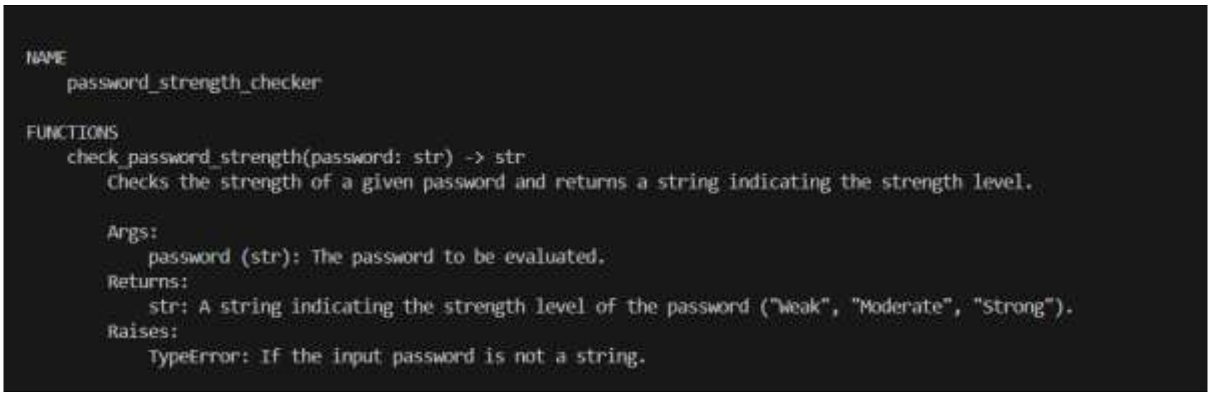


```
Checks the strength of a given password.

Args:
    password (str): The password to be evaluated.

Returns:
Args:
    password (str): The password to be evaluated.

Returns:

Returns:
Returns:
    str: A message indicating the strength of the password.
    str: A message indicating the strength of the password.


Help on function check_password_strength in module __main__:

check_password_strength(password: str) -> str
    Checks the strength of a given password.
check_password_strength(password: str) -> str
    Checks the strength of a given password.
    Checks the strength of a given password.

    Args:
        password (str): The password to be evaluated.

    Returns:
        str: A message indicating the strength of the password.
```

Google style.


demo.py 3
string_utilities.py
password_strength_checker.py 3
documentationexample.py 8
Math_Utilities.py
calculator.py 5
password_strength_checker.py > check_password_strength

```python
def check_password_strength(password: str) -> str:
    """
    Checks the strength of a given password and returns a string indicating the strength level.

    Args:
        password (str): The password to be evaluated.
    Returns:
        str: A string indicating the strength level of the password ("Weak", "Moderate", "Strong").
    Raises:
        TypeError: If the input password is not a string.
    Example:
        >>> check_password_strength("password")
        'Weak'
        >>> check_password_strength("Password123")
        'Moderate'
    """
    if not isinstance(password, str):
        raise TypeError("Input must be a string.")

    length = len(password)
    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(not c.isalnum() for c in password)

    if length < 6:
        return "Weak"
    elif length < 12:
        return "Moderate"
    elif has_upper and has_lower and has_digit and has_special:
        return "Strong"
    else:
        return "Moderate"
```



```
NAME
    password_strength_checker

FUNCTIONS
    check_password_strength(password: str) -> str
        Checks the strength of a given password and returns a string indicating the strength level.

        Args:
            password (str): The password to be evaluated.
        Returns:
            str: A string indicating the strength level of the password ("Weak", "Moderate", "Strong").
        Raises:
            TypeError: If the input password is not a string.
```

Most appropriate style.

The documentation style used in the `check_password_strength` function is comprehensive and follows best practices for documenting security-related code.
It includes a clear description of the function's purpose, detailed information about the parameters and return value,
and mentions potential exceptions that could be raised. Additionally, it provides examples of how to use the function,
which can be very helpful for users.
This style is appropriate for security-related code because it ensures that users understand the
function's behavior and potential pitfalls, which is crucial when dealing with security-sensitive operations.

Therefore, I recommend this documentation style for security-related code.

Problem 3: Math Utilities Module

Task:

1. Create a module math_utils.py with functions:

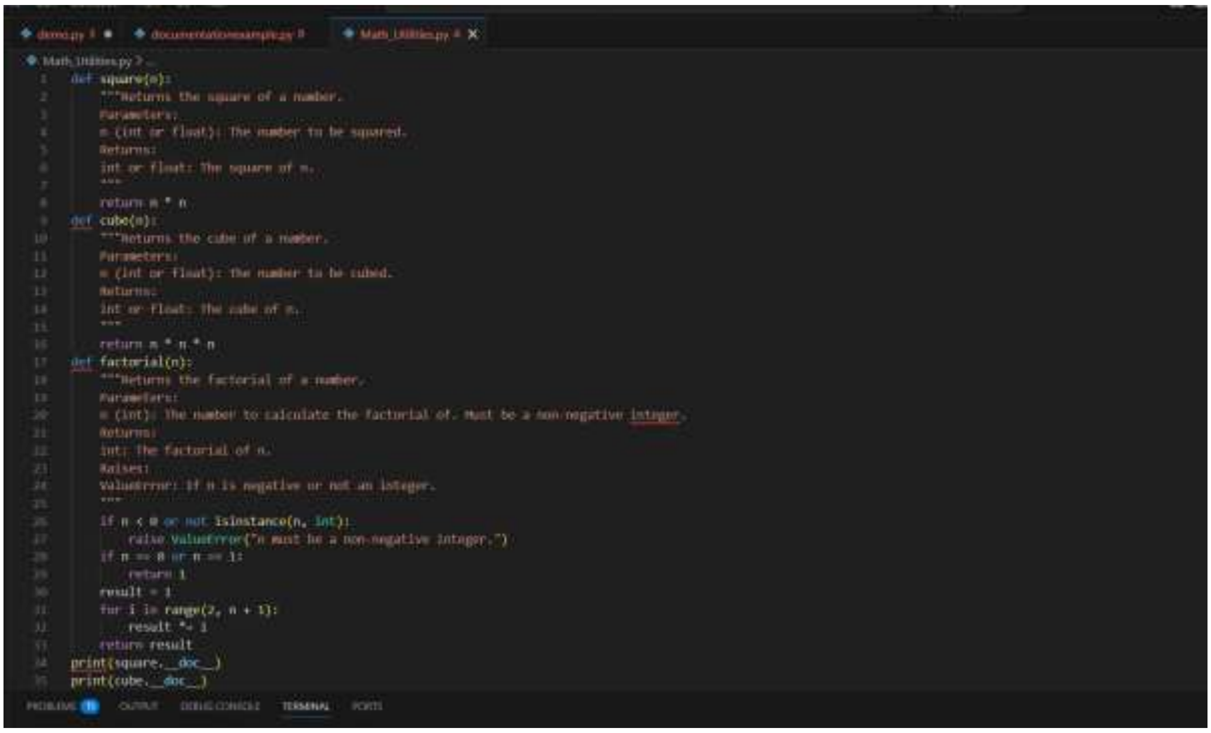
o square(n)

o cube(n)

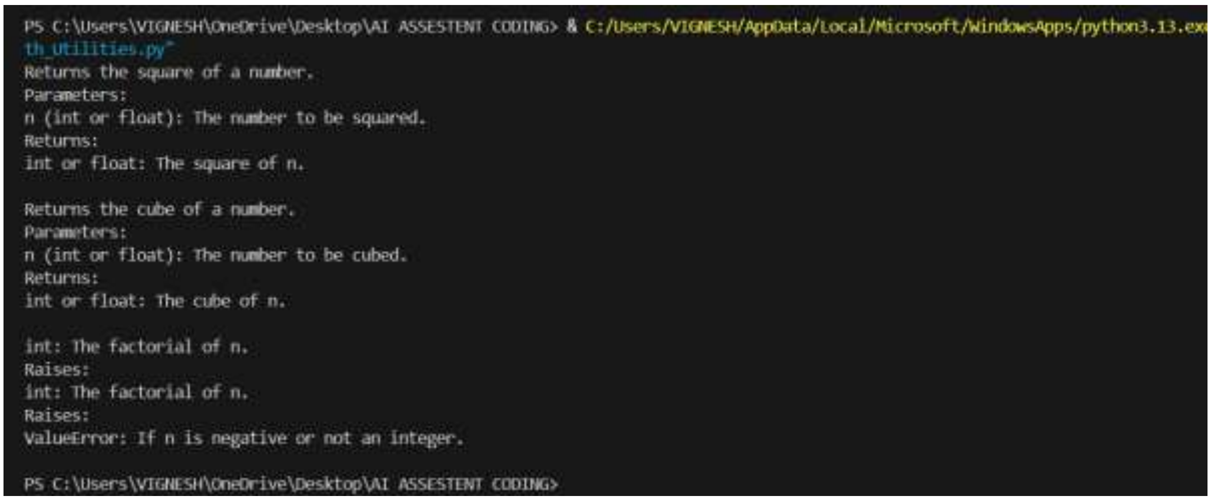
o factorial(n)

2. Generate docstrings automatically using AI tools.

3. Export documentation as an HTML file.

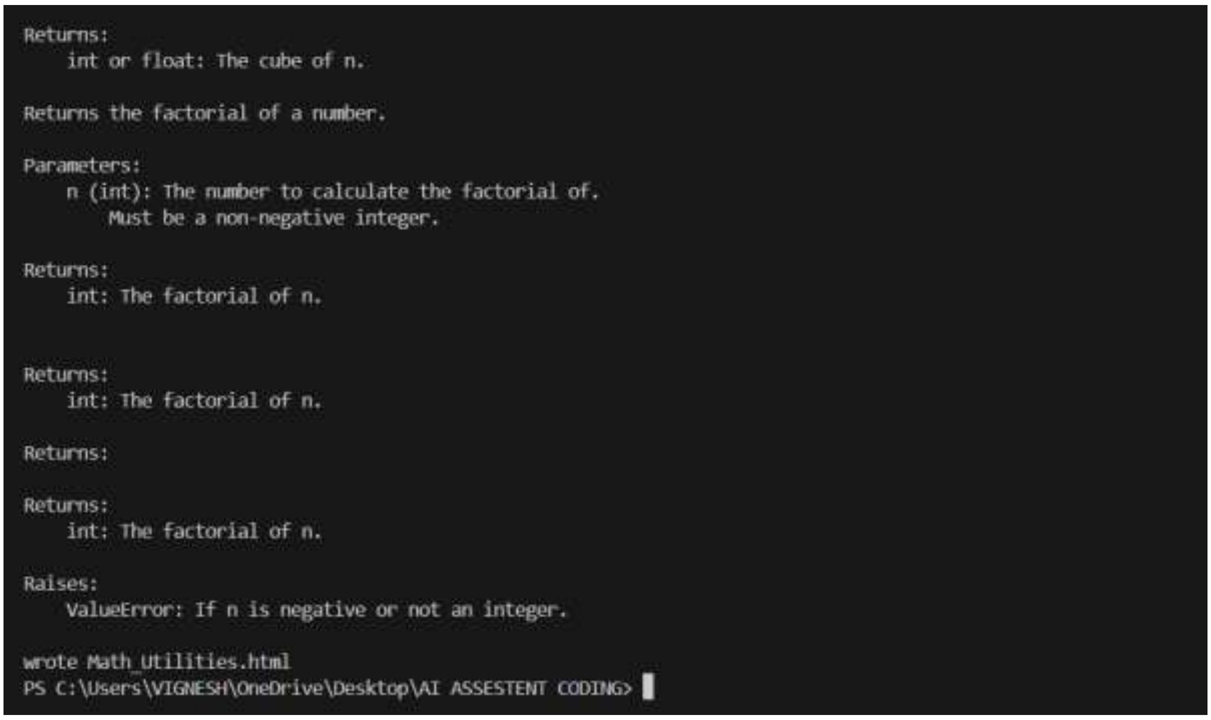


```python
def square(n):
    """Returns the square of a number.
    Parameters:
    n (int or float): The number to be squared.
    Returns:
    int or float: The square of n.
    """
    return n * n
def cube(n):
    """Returns the cube of a number.
    Parameters:
    n (int or float): The number to be cubed.
    Returns:
    int or float: The cube of n.
    """
    return n * n * n
def factorial(n):
    """Returns the factorial of a number.
    Parameters:
    n (int): The number to calculate the factorial of. Must be a non-negative integer.
    Returns:
    int: The factorial of n.
    Raises:
    ValueError: If n is negative or not an integer.
    """
    if n < 0 or not isinstance(n, int):
        raise ValueError("n must be a non-negative integer.")
    if n == 0 or n == 1:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
print(square.__doc__)
print(cube.__doc__)
```



```
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING> & C:/Users/VIGNESH/AppData/Local/Microsoft/WindowsApps/python3.13.ex
th_Utilities.py"
Returns the square of a number.
Parameters:
n (int or float): The number to be squared.
Returns:
int or float: The square of n.

Returns the cube of a number.
Parameters:
n (int or float): The number to be cubed.
Returns:
int or float: The cube of n.

int: The factorial of n.
Raises:
int: The factorial of n.
Raises:
ValueError: If n is negative or not an integer.

PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING>
```

Documentation as an HTML file:



```
Returns:
    int or float: The cube of n.

Returns the factorial of a number.

Parameters:
    n (int): The number to calculate the factorial of.
        Must be a non-negative integer.

Returns:
    int: The factorial of n.


Returns:
    int: The factorial of n.

Returns:

Returns:
    int: The factorial of n.

Raises:
    ValueError: If n is negative or not an integer.

wrote Math_Utilities.html
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING> █
```

Problem 4: Attendance Management Module

Task:

1. Create a module attendance.py with functions:

o mark_present(student)

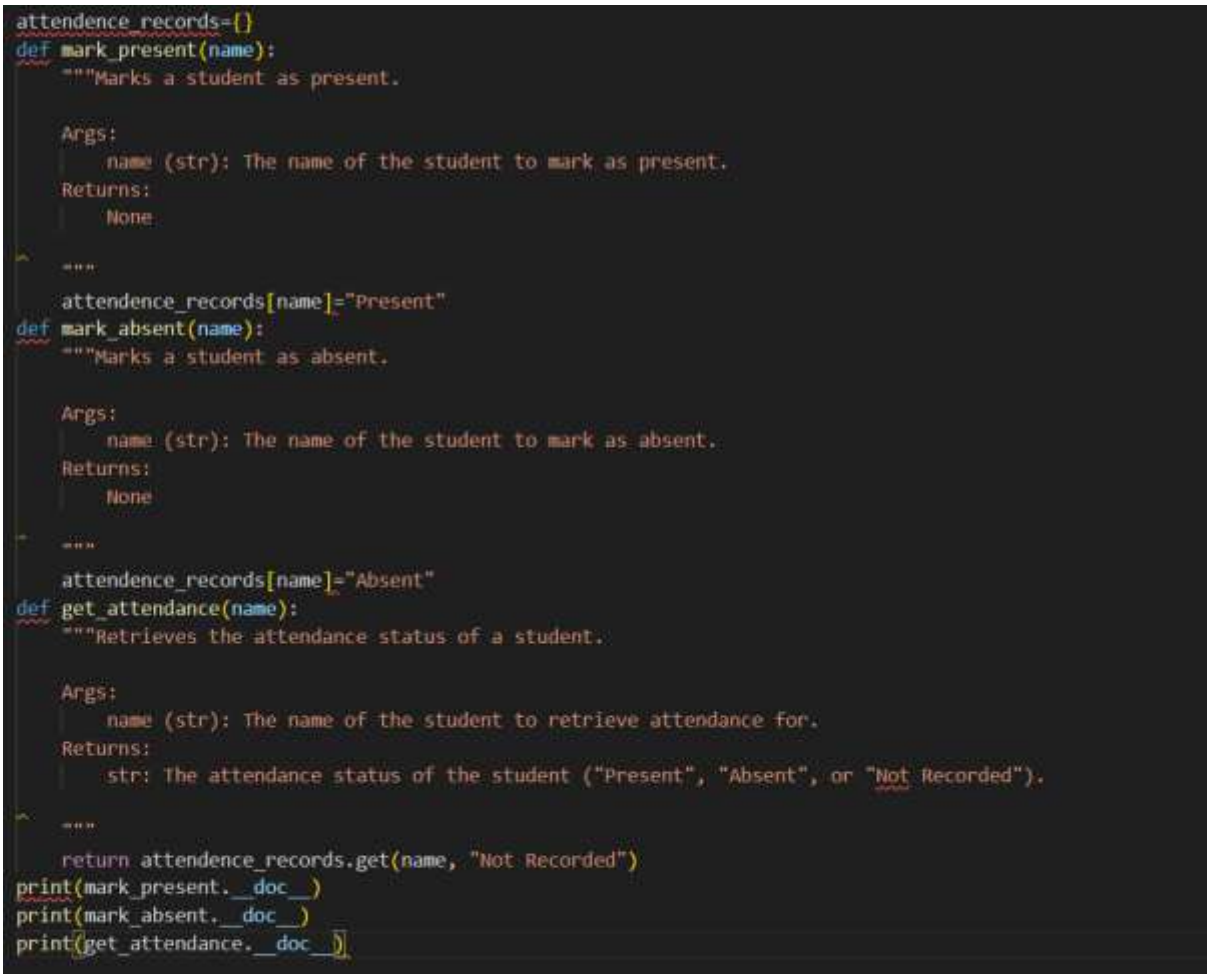
o mark_absent(student)

o get_attendance(student)

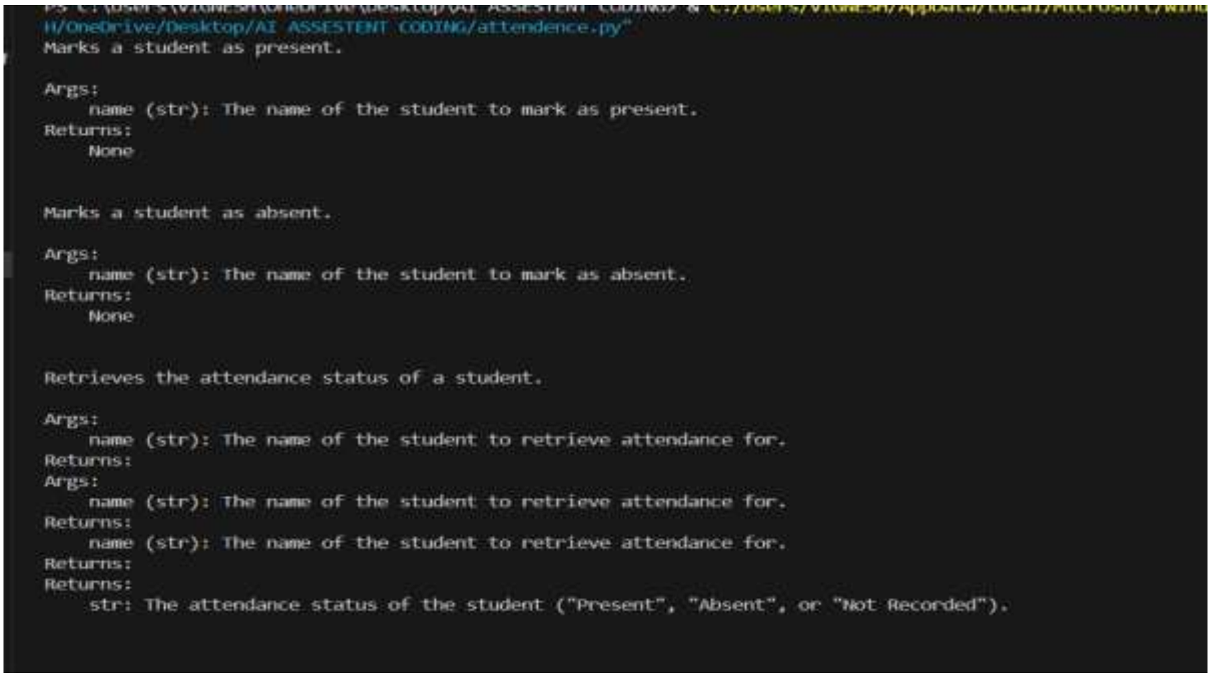
2. Add proper docstrings.

3. Generate and view documentation in terminal and browse

Docstrings.

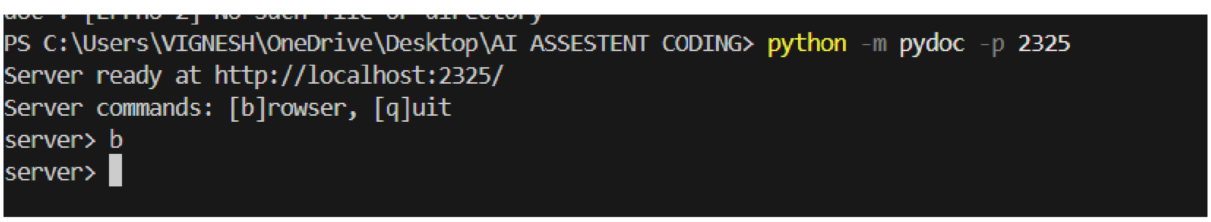


```python
attendence_records={}
def mark_present(name):
    """Marks a student as present.

    Args:
        name (str): The name of the student to mark as present.
    Returns:
        None

    """
    attendence_records[name]="Present"
def mark_absent(name):
    """Marks a student as absent.

    Args:
        name (str): The name of the student to mark as absent.
    Returns:
        None

    """
    attendence_records[name]="Absent"
def get_attendance(name):
    """Retrieves the attendance status of a student.

    Args:
        name (str): The name of the student to retrieve attendance for.
    Returns:
        str: The attendance status of the student ("Present", "Absent", or "Not Recorded").

    """
    return attendence_records.get(name, "Not Recorded")
print(mark_present.__doc__)
print(mark_absent.__doc__)
print(get_attendance.__doc__)
```

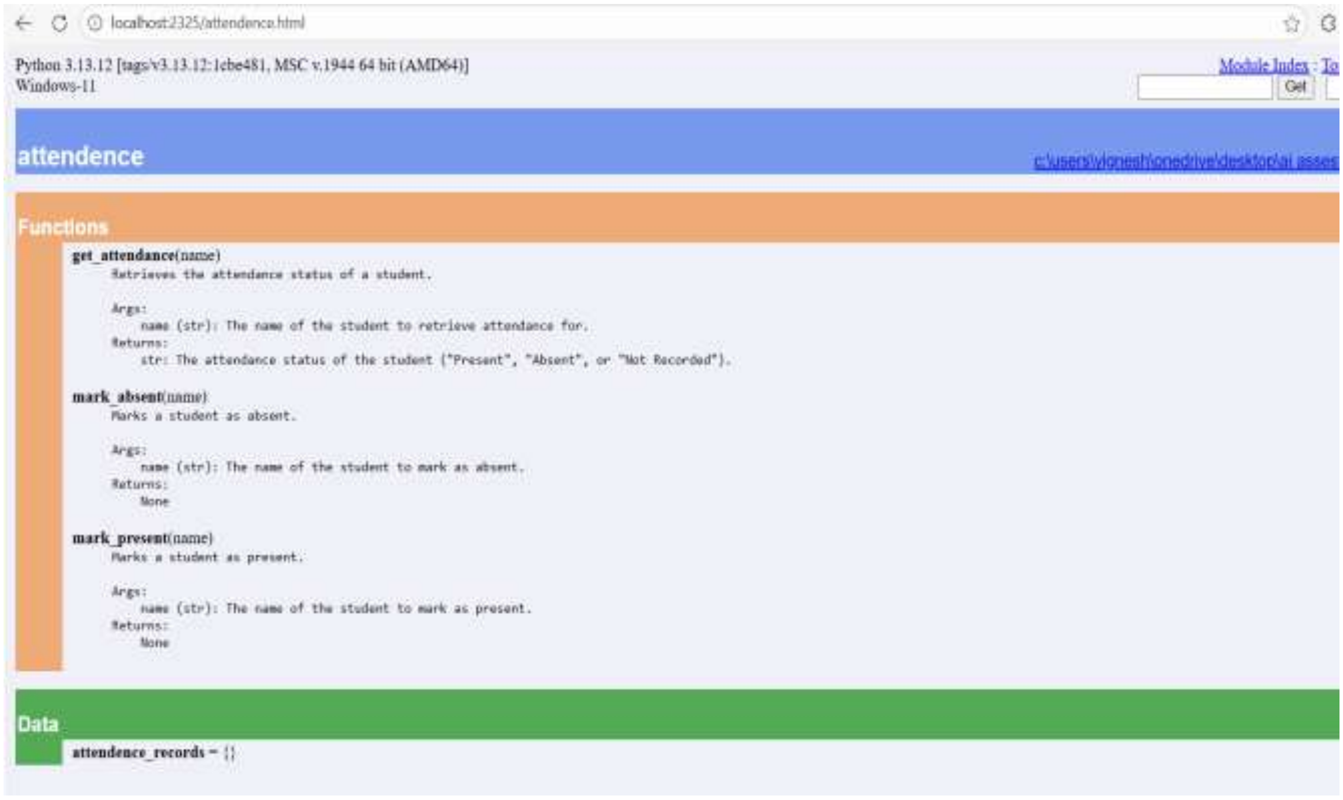


```
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING> & C:/Users/VIGNESH/AppData/Local/Microsoft/Wind
H/OneDrive/Desktop/AI ASSESTENT CODING/attendence.py"
Marks a student as present.

Args:
    name (str): The name of the student to mark as present.
Returns:
    None


Marks a student as absent.

Args:
    name (str): The name of the student to mark as absent.
Returns:
    None


Retrieves the attendance status of a student.

Args:
    name (str): The name of the student to retrieve attendance for.
Returns:
Args:
    name (str): The name of the student to retrieve attendance for.
Returns:
    name (str): The name of the student to retrieve attendance for.
Returns:
Returns:
    str: The attendance status of the student ("Present", "Absent", or "Not Recorded").
```

View documentation in terminal and browse:



```
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING> python -m pydoc -p 2325
Server ready at http://localhost:2325/
Server commands: [b]rowser, [q]uit
server> b
server>
```



```
localhost:2325/attendence.html

Python 3.13.12 [tags/v3.13.12:1cbe481, MSC v.1944 64 bit (AMD64)]
Windows-11

attendence                                      c:\users\vignesh\onedrive\desktop\ai asses

Functions
    get_attendance(name)
        Retrieves the attendance status of a student.

        Args:
            name (str): The name of the student to retrieve attendance for.
        Returns:
            str: The attendance status of the student ("Present", "Absent", or "Not Recorded").

    mark_absent(name)
        Marks a student as absent.

        Args:
            name (str): The name of the student to mark as absent.
        Returns:
            None

    mark_present(name)
        Marks a student as present.

        Args:
            name (str): The name of the student to mark as present.
        Returns:
            None

Data
    attendence_records = {}
```

Problem 5: File Handling Function

Consider the function:

def read_file(filename):

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

return f.read()

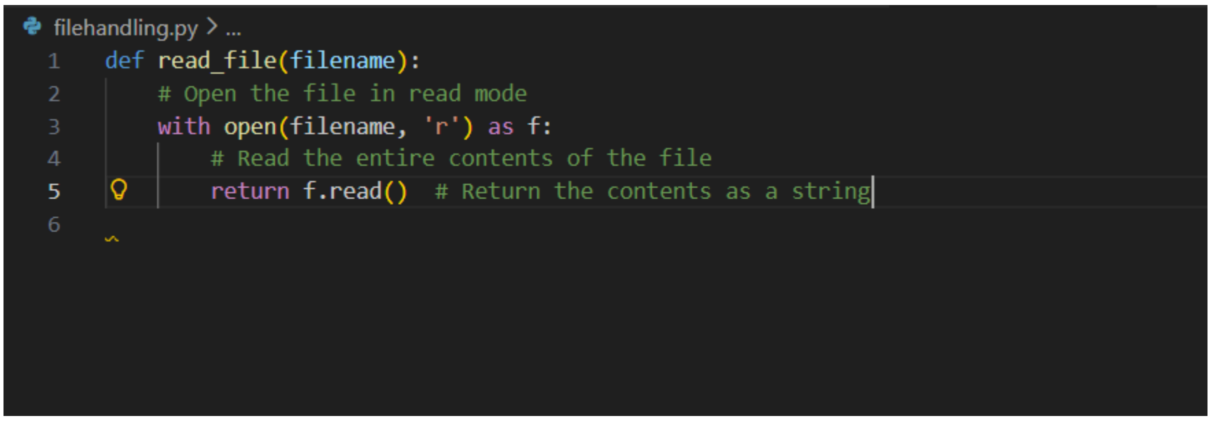
Task:

1. Write documentation using all three formats.

2. Identify which style best explains exception handling.

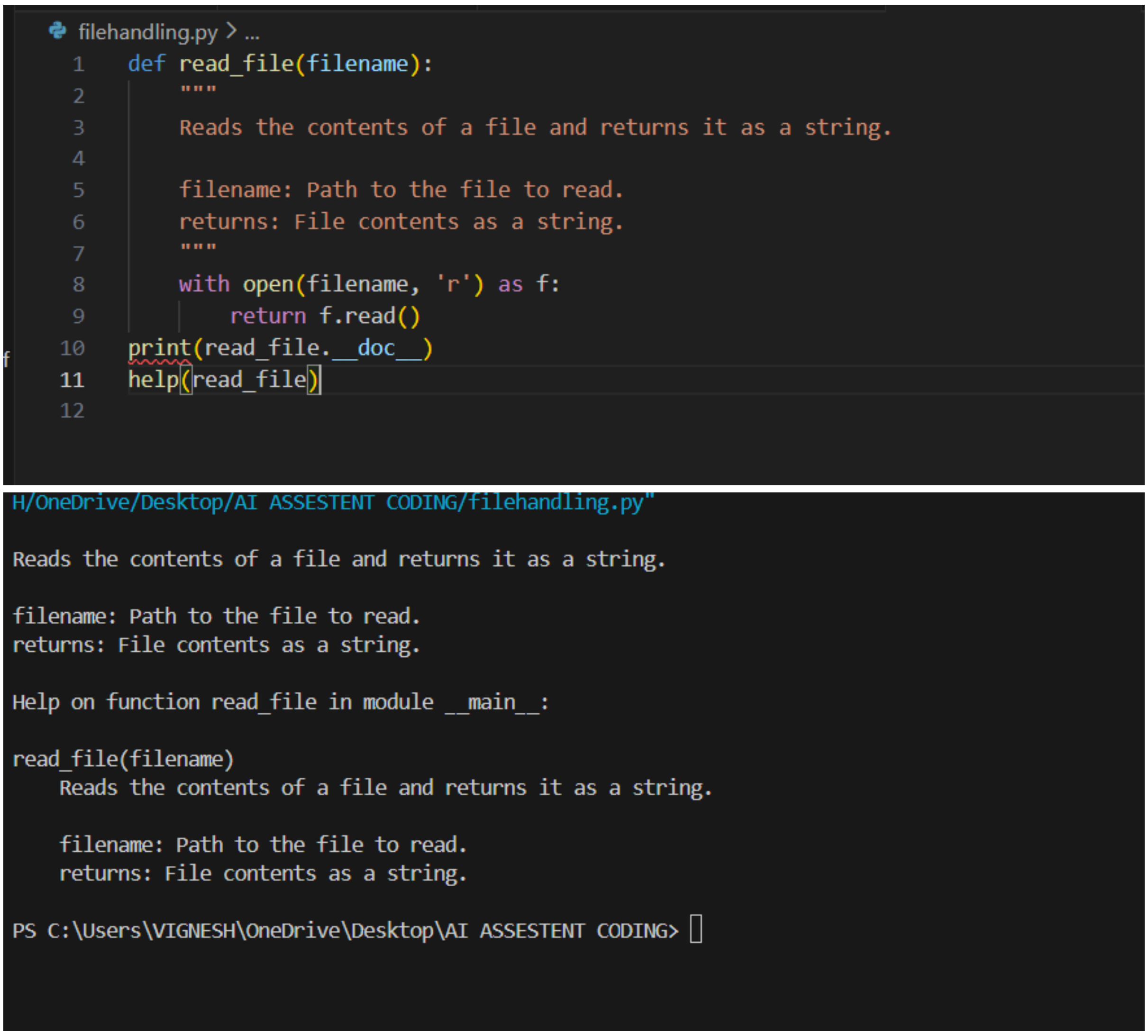
3. Justify your recommendation.

Inline:



```
filehandling.py > ...
def read_file(filename):
    # Open the file in read mode
    with open(filename, 'r') as f:
        # Read the entire contents of the file
        return f.read()  # Return the contents as a string

```

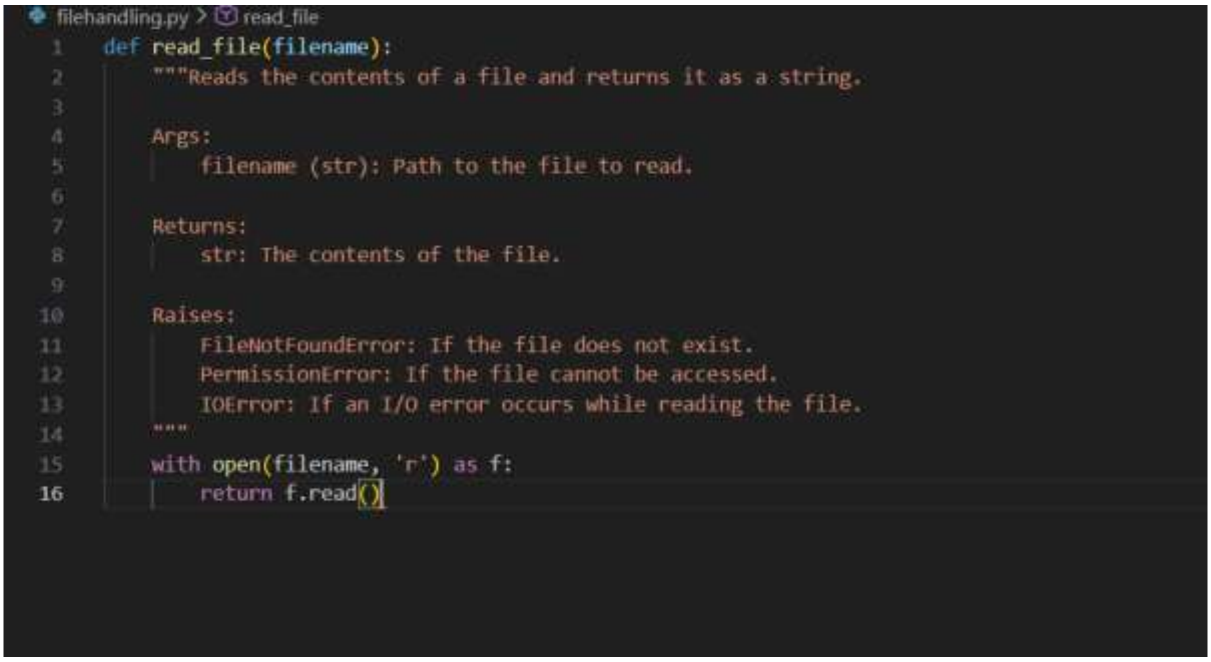
DocString:



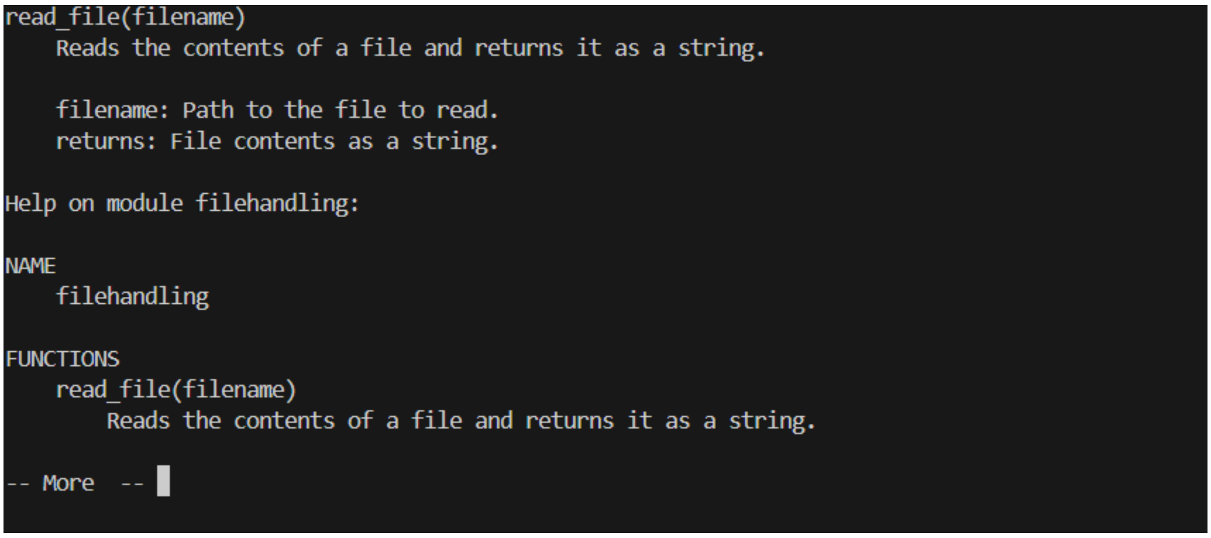
```
filehandling.py > ...
def read_file(filename):
    """
    Reads the contents of a file and returns it as a string.

    filename: Path to the file to read.
    returns: File contents as a string.
    """
    with open(filename, 'r') as f:
        return f.read()
print(read_file.__doc__)
help(read_file)

```

```
H/OneDrive/Desktop/AI ASSESTENT CODING/filehandling.py"

Reads the contents of a file and returns it as a string.

filename: Path to the file to read.
returns: File contents as a string.

Help on function read_file in module __main__:

read_file(filename)
    Reads the contents of a file and returns it as a string.

    filename: Path to the file to read.
    returns: File contents as a string.

PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING>
```

Google style:


filehandling.py > read_file

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

    Args:
        filename (str): Path to the file to read.

    Returns:
        str: The contents of the file.

    Raises:
        FileNotFoundError: If the file does not exist.
        PermissionError: If the file cannot be accessed.
        IOError: If an I/O error occurs while reading the file.
    """
    with open(filename, 'r') as f:
        return f.read()
```


```
read_file(filename)
    Reads the contents of a file and returns it as a string.

    filename: Path to the file to read.
    returns: File contents as a string.

Help on module filehandling:

NAME
    filehandling

FUNCTIONS
    read_file(filename)
        Reads the contents of a file and returns it as a string.

-- More  --
```

Justification :

The style that best explains exception handling among inline comments, docstrings, and Google style documentation is the Google style documentation.

This is because Google style documentation provides a clear and structured way to document exceptions, including the type of exception, the conditions under which it may be raised, and any relevant information about the exception.

In contrast, inline comments may not provide enough context or detail about the exceptions, and

docstrings may not always follow a consistent format for documenting exceptions.

Google style documentation is widely used and recognized in the Python community,

making it easier for developers to understand and follow the documentation when it comes to exception handling.

Therefore, I recommend using Google style documentation for documenting exceptions in Python code.