AI ASSISTED CODING

ASSIGNMENT-9.5

Name: G. Harshitha

H.T.No:2303A51102

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

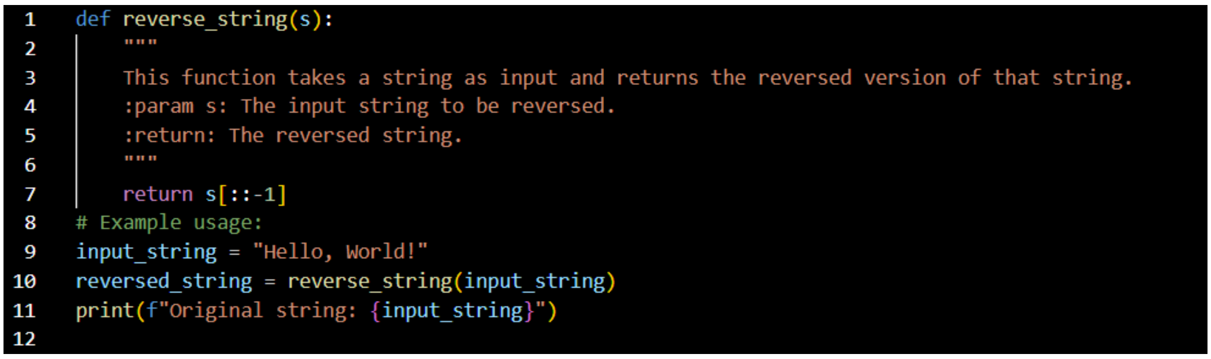
o (b) Inline comments

o (c) Google-style documentation

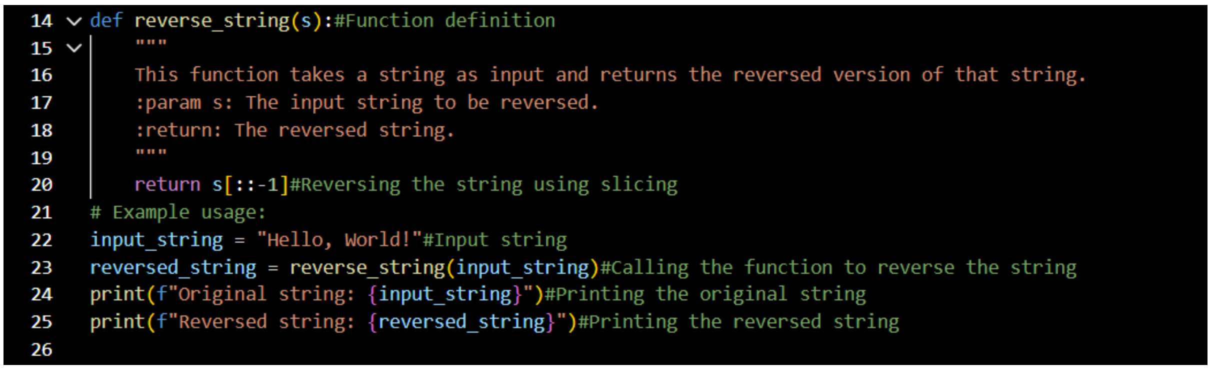
2. Compare the three documentation styles.

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

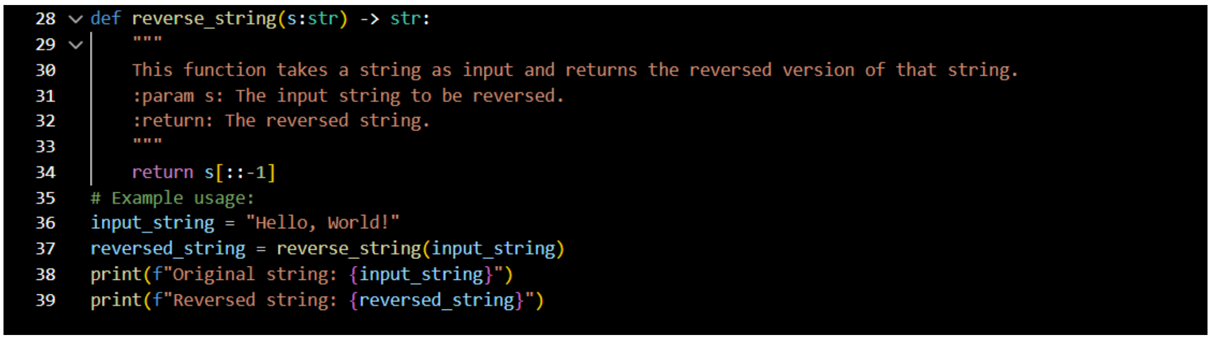
library.



```python
def reverse_string(s):
    """
    This function takes a string as input and returns the reversed version of that string.
    :param s: The input string to be reversed.
    :return: The reversed string.
    """
    return s[::-1]
# Example usage:
input_string = "Hello, World!"
reversed_string = reverse_string(input_string)
print(f"Original string: {input_string}")

```



```python
def reverse_string(s):#Function definition
    """
    This function takes a string as input and returns the reversed version of that string.
    :param s: The input string to be reversed.
    :return: The reversed string.
    """
    return s[::-1]#Reversing the string using slicing
# Example usage:
input_string = "Hello, World!"#Input string
reversed_string = reverse_string(input_string)#Calling the function to reverse the string
print(f"Original string: {input_string}")#Printing the original string
print(f"Reversed string: {reversed_string}")#Printing the reversed string

```



```python
def reverse_string(s:str) -> str:
    """
    This function takes a string as input and returns the reversed version of that string.
    :param s: The input string to be reversed.
    :return: The reversed string.
    """
    return s[::-1]
# Example usage:
input_string = "Hello, World!"
reversed_string = reverse_string(input_string)
print(f"Original string: {input_string}")
print(f"Reversed string: {reversed_string}")
```

Inline comments are written using # to explain specific lines or logic inside the code.
Google style docstrings use triple quotes with structured sections like Args and Returns for clarity.
Standard docstrings give a general description of what a function or class does.
Inline comments help with quick understanding of complex steps.
Google style docstrings are recommended because they are clear, structured and best.

Problem 2: Password Strength Checker

Consider the function:

def check_strength(password):

return len(password) >= 8

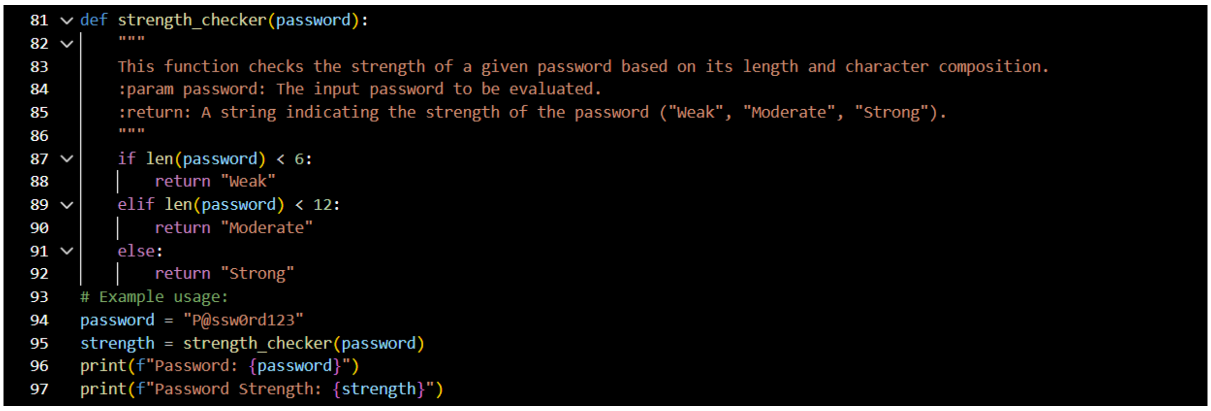
Task:

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

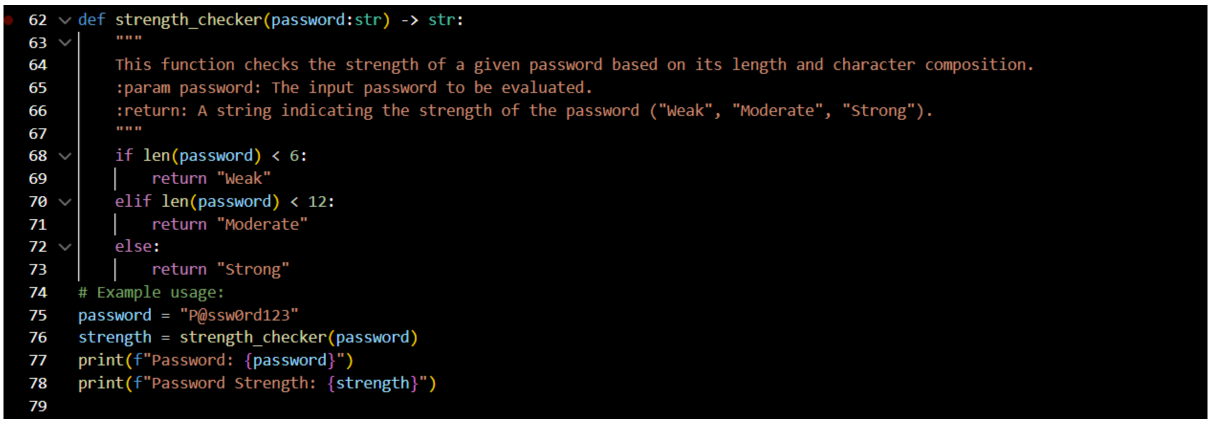
Google style.

2. Compare documentation styles for security-related code.

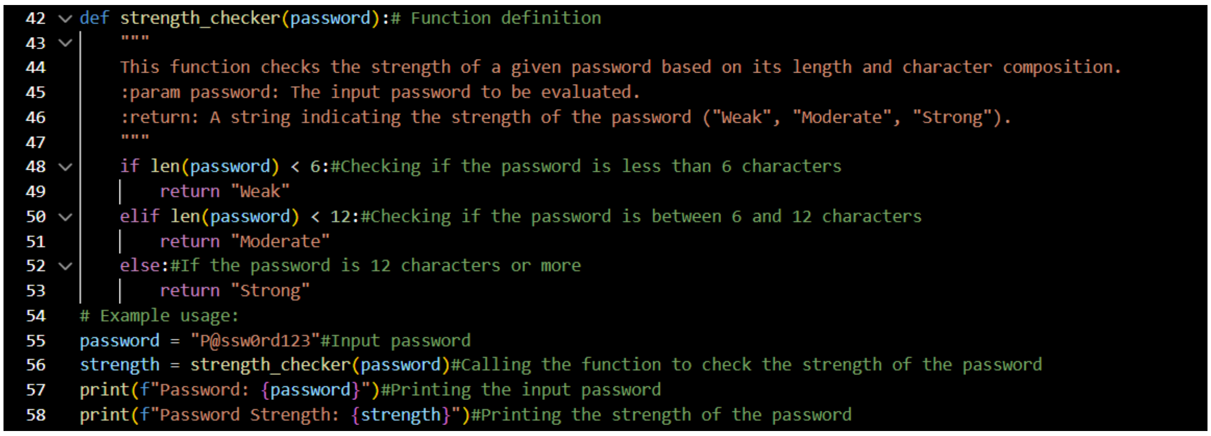
3. Recommend the most appropriate style.



```python
def strength_checker(password):
    """
    This function checks the strength of a given password based on its length and character composition.
    :param password: The input password to be evaluated.
    :return: A string indicating the strength of the password ("Weak", "Moderate", "Strong").
    """
    if len(password) < 6:
        return "Weak"
    elif len(password) < 12:
        return "Moderate"
    else:
        return "Strong"
# Example usage:
password = "P@ssw0rd123"
strength = strength_checker(password)
print(f"Password: {password}")
print(f"Password Strength: {strength}")
```



```python
def strength_checker(password:str) -> str:
    """
    This function checks the strength of a given password based on its length and character composition.
    :param password: The input password to be evaluated.
    :return: A string indicating the strength of the password ("Weak", "Moderate", "Strong").
    """
    if len(password) < 6:
        return "Weak"
    elif len(password) < 12:
        return "Moderate"
    else:
        return "Strong"
# Example usage:
password = "P@ssw0rd123"
strength = strength_checker(password)
print(f"Password: {password}")
print(f"Password Strength: {strength}")
```



```python
def strength_checker(password):# Function definition
    """
    This function checks the strength of a given password based on its length and character composition.
    :param password: The input password to be evaluated.
    :return: A string indicating the strength of the password ("Weak", "Moderate", "Strong").
    """
    if len(password) < 6:#Checking if the password is less than 6 characters
        return "Weak"
    elif len(password) < 12:#Checking if the password is between 6 and 12 characters
        return "Moderate"
    else:#If the password is 12 characters or more
        return "Strong"
# Example usage:
password = "P@ssw0rd123"#Input password
strength = strength_checker(password)#Calling the function to check the strength of the password
print(f"Password: {password}")#Printing the input password
print(f"Password Strength: {strength}")#Printing the strength of the password
```

Inline comments are written using # to explain specific lines or logic inside the code.
Google style docstrings use triple quotes with structured sections like Args and Returns for clarity.
Standard docstrings give a general description of what a function or class does.
Inline comments help with quick understanding of complex steps.
Google style docstrings are recommended because they are clear, structured and best.

Problem 3: Math Utilities Module

Task:

1. Create a module math_utils.py with functions:

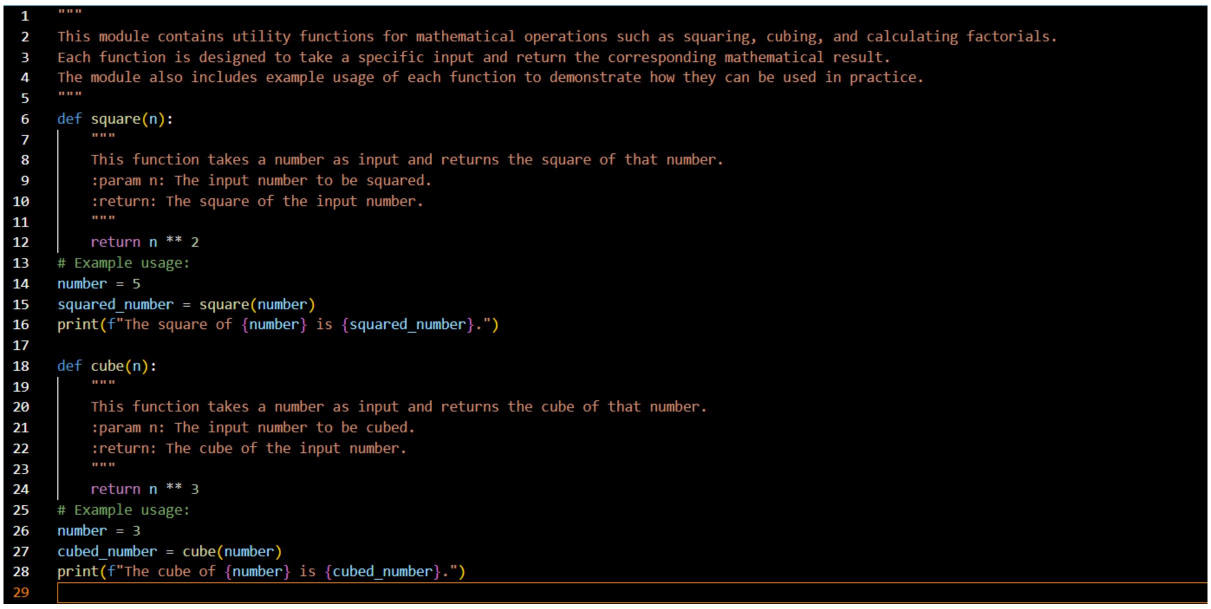
o square(n)

o cube(n)

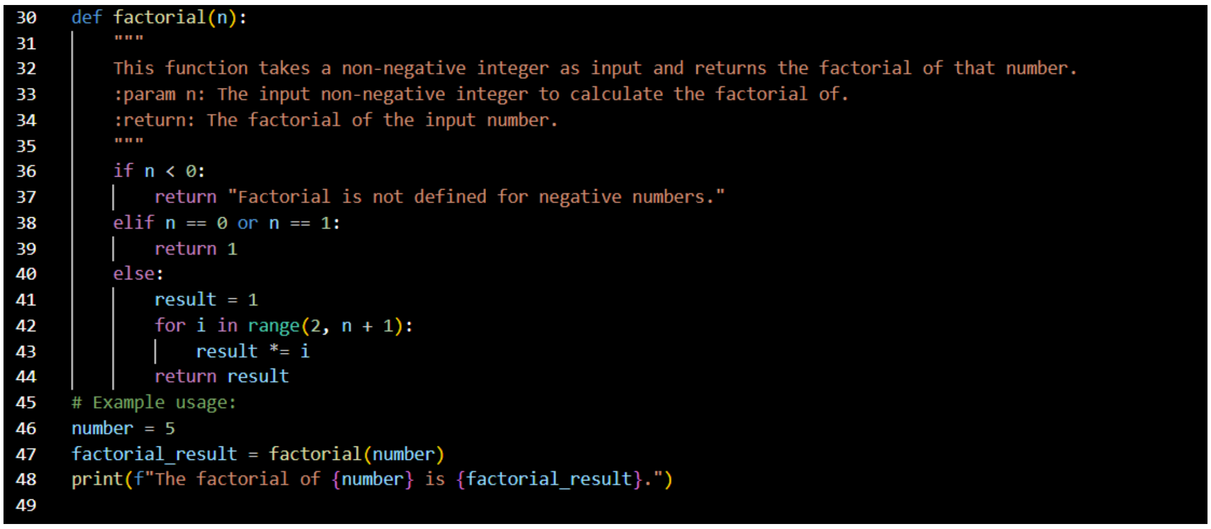
o factorial(n)

2. Generate docstrings automatically using AI tools.

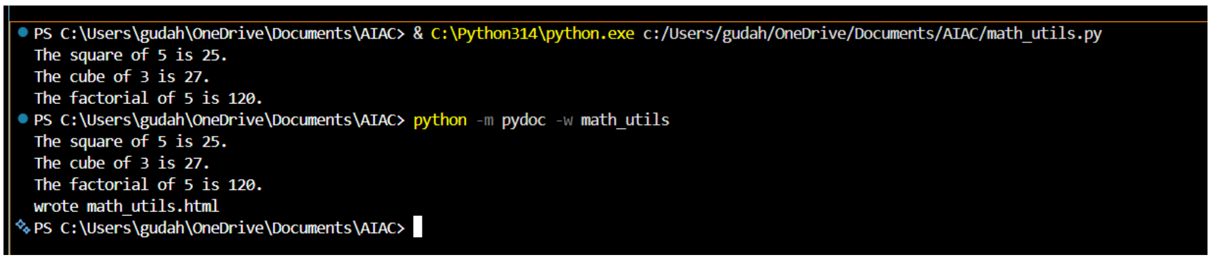
3. Export documentation as an HTML file.



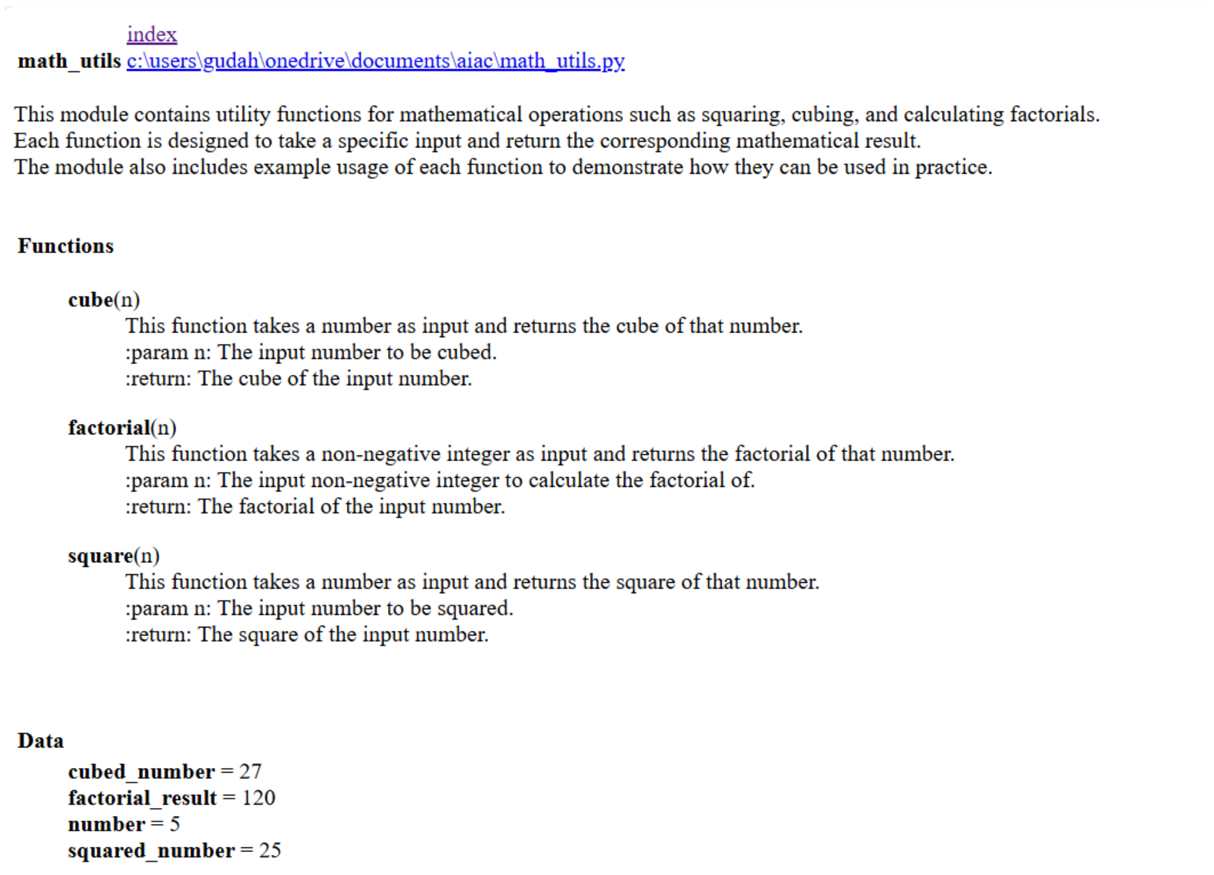
```python
"""
This module contains utility functions for mathematical operations such as squaring, cubing, and calculating factorials.
Each function is designed to take a specific input and return the corresponding mathematical result.
The module also includes example usage of each function to demonstrate how they can be used in practice.
"""
def square(n):
    """
    This function takes a number as input and returns the square of that number.
    :param n: The input number to be squared.
    :return: The square of the input number.
    """
    return n ** 2
# Example usage:
number = 5
squared_number = square(number)
print(f"The square of {number} is {squared_number}.")

def cube(n):
    """
    This function takes a number as input and returns the cube of that number.
    :param n: The input number to be cubed.
    :return: The cube of the input number.
    """
    return n ** 3
# Example usage:
number = 3
cubed_number = cube(number)
print(f"The cube of {number} is {cubed_number}.")

```



```python
def factorial(n):
    """
    This function takes a non-negative integer as input and returns the factorial of that number.
    :param n: The input non-negative integer to calculate the factorial of.
    :return: The factorial of the input number.
    """
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# Example usage:
number = 5
factorial_result = factorial(number)
print(f"The factorial of {number} is {factorial_result}.")

```



```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/math_utils.py
The square of 5 is 25.
The cube of 3 is 27.
The factorial of 5 is 120.
PS C:\Users\gudah\OneDrive\Documents\AIAC> python -m pydoc -w math_utils
The square of 5 is 25.
The cube of 3 is 27.
The factorial of 5 is 120.
wrote math_utils.html
PS C:\Users\gudah\OneDrive\Documents\AIAC> 
```



index
math_utils c:\users\gudah\onedrive\documents\aiac\math_utils.py

This module contains utility functions for mathematical operations such as squaring, cubing, and calculating factorials.
Each function is designed to take a specific input and return the corresponding mathematical result.
The module also includes example usage of each function to demonstrate how they can be used in practice.

Functions

cube(n)
This function takes a number as input and returns the cube of that number.
:param n: The input number to be cubed.
:return: The cube of the input number.

factorial(n)
This function takes a non-negative integer as input and returns the factorial of that number.
:param n: The input non-negative integer to calculate the factorial of.
:return: The factorial of the input number.

square(n)
This function takes a number as input and returns the square of that number.
:param n: The input number to be squared.
:return: The square of the input number.

Data

cubed_number = 27
factorial_result = 120
number = 5
squared_number = 25

Problem 4: Attendance Management Module

Task:

1. Create a module attendance.py with functions:

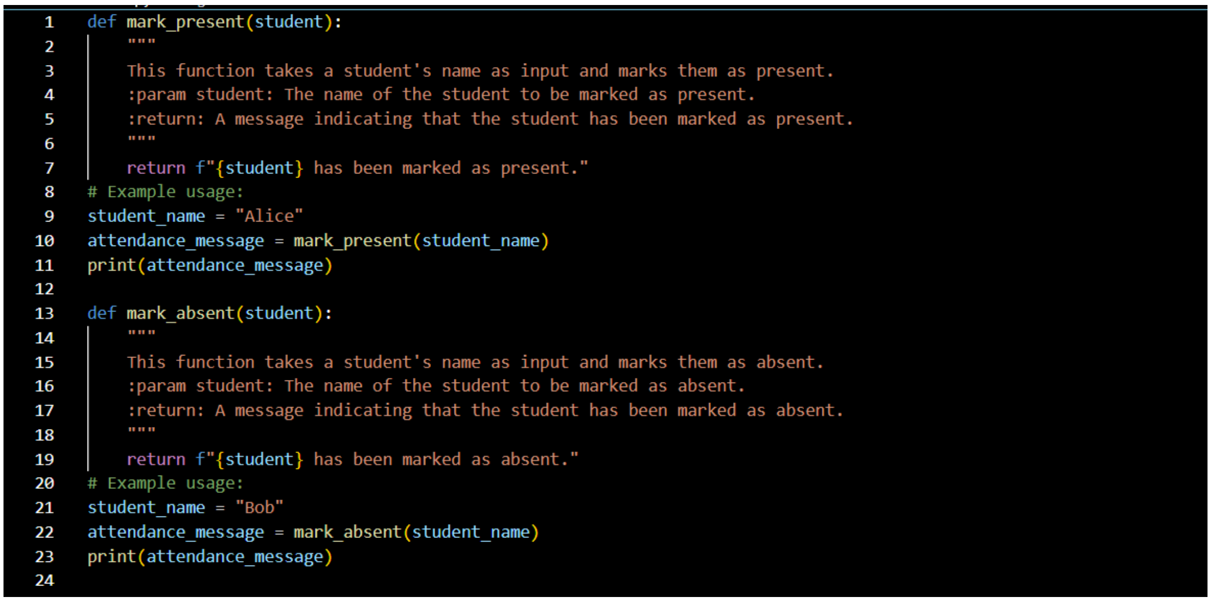
o mark_present(student)

o mark_absent(student)

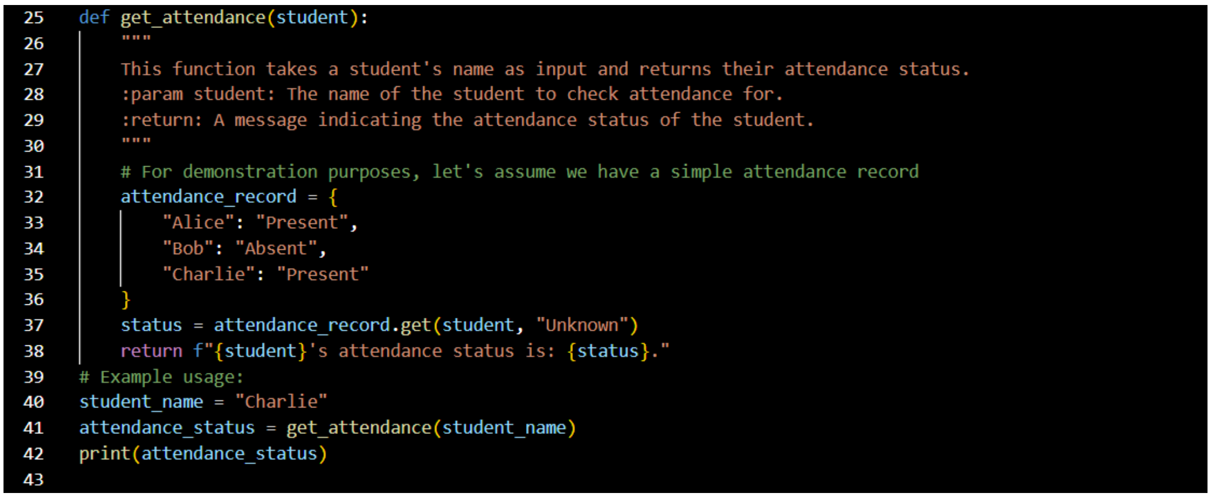
o get_attendance(student)

2. Add proper docstrings.

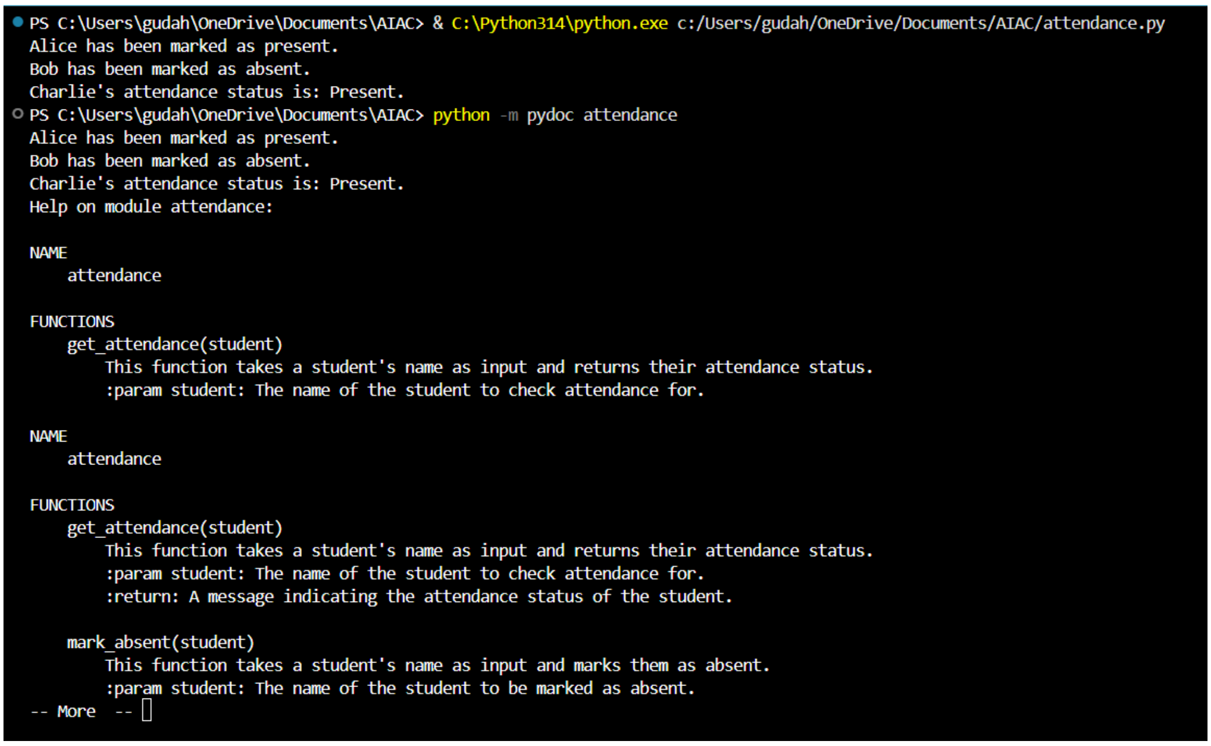
3. Generate and view documentation in terminal and browser



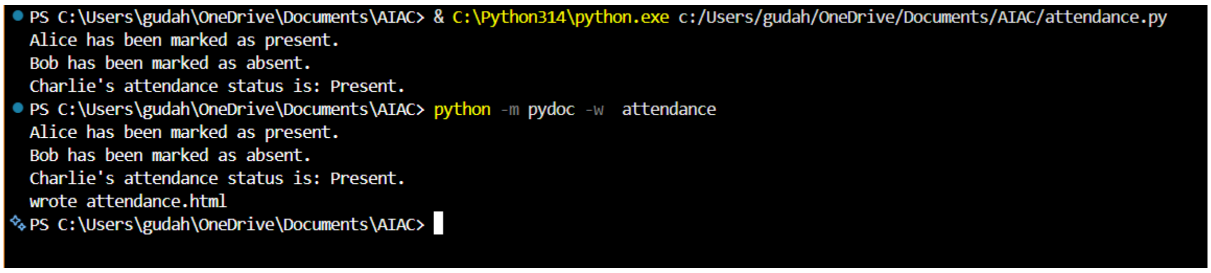
```python
def mark_present(student):
    """
    This function takes a student's name as input and marks them as present.
    :param student: The name of the student to be marked as present.
    :return: A message indicating that the student has been marked as present.
    """
    return f"{student} has been marked as present."
# Example usage:
student_name = "Alice"
attendance_message = mark_present(student_name)
print(attendance_message)

def mark_absent(student):
    """
    This function takes a student's name as input and marks them as absent.
    :param student: The name of the student to be marked as absent.
    :return: A message indicating that the student has been marked as absent.
    """
    return f"{student} has been marked as absent."
# Example usage:
student_name = "Bob"
attendance_message = mark_absent(student_name)
print(attendance_message)

```



```python
def get_attendance(student):
    """
    This function takes a student's name as input and returns their attendance status.
    :param student: The name of the student to check attendance for.
    :return: A message indicating the attendance status of the student.
    """
    # For demonstration purposes, let's assume we have a simple attendance record
    attendance_record = {
        "Alice": "Present",
        "Bob": "Absent",
        "Charlie": "Present"
    }
    status = attendance_record.get(student, "Unknown")
    return f"{student}'s attendance status is: {status}."
# Example usage:
student_name = "Charlie"
attendance_status = get_attendance(student_name)
print(attendance_status)

```



```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/attendance.py
Alice has been marked as present.
Bob has been marked as absent.
Charlie's attendance status is: Present.
PS C:\Users\gudah\OneDrive\Documents\AIAC> python -m pydoc attendance
Alice has been marked as present.
Bob has been marked as absent.
Charlie's attendance status is: Present.
Help on module attendance:

NAME
    attendance

FUNCTIONS
    get_attendance(student)
        This function takes a student's name as input and returns their attendance status.
        :param student: The name of the student to check attendance for.

NAME
    attendance

FUNCTIONS
    get_attendance(student)
        This function takes a student's name as input and returns their attendance status.
        :param student: The name of the student to check attendance for.
        :return: A message indicating the attendance status of the student.

    mark_absent(student)
        This function takes a student's name as input and marks them as absent.
        :param student: The name of the student to be marked as absent.
-- More  --
```



```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/attendance.py
Alice has been marked as present.
Bob has been marked as absent.
Charlie's attendance status is: Present.
PS C:\Users\gudah\OneDrive\Documents\AIAC> python -m pydoc -w  attendance
Alice has been marked as present.
Bob has been marked as absent.
Charlie's attendance status is: Present.
wrote attendance.html
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

index
attendance c:\users\gudah\onedrive\documents\aiac\attendance.py

Functions

get_attendance(student)
This function takes a student's name as input and returns their attendance status.
:param student: The name of the student to check attendance for.
:return: A message indicating the attendance status of the student.

mark_absent(student)
This function takes a student's name as input and marks them as absent.
:param student: The name of the student to be marked as absent.
:return: A message indicating that the student has been marked as absent.

mark_present(student)
This function takes a student's name as input and marks them as present.
:param student: The name of the student to be marked as present.
:return: A message indicating that the student has been marked as present.

Data

attendance_message = 'Bob has been marked as absent.'
attendance_status = "Charlie's attendance status is: Present."
student_name = 'Charlie'

Problem 5: File Handling Function

Consider the function:

def read_file(filename):

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

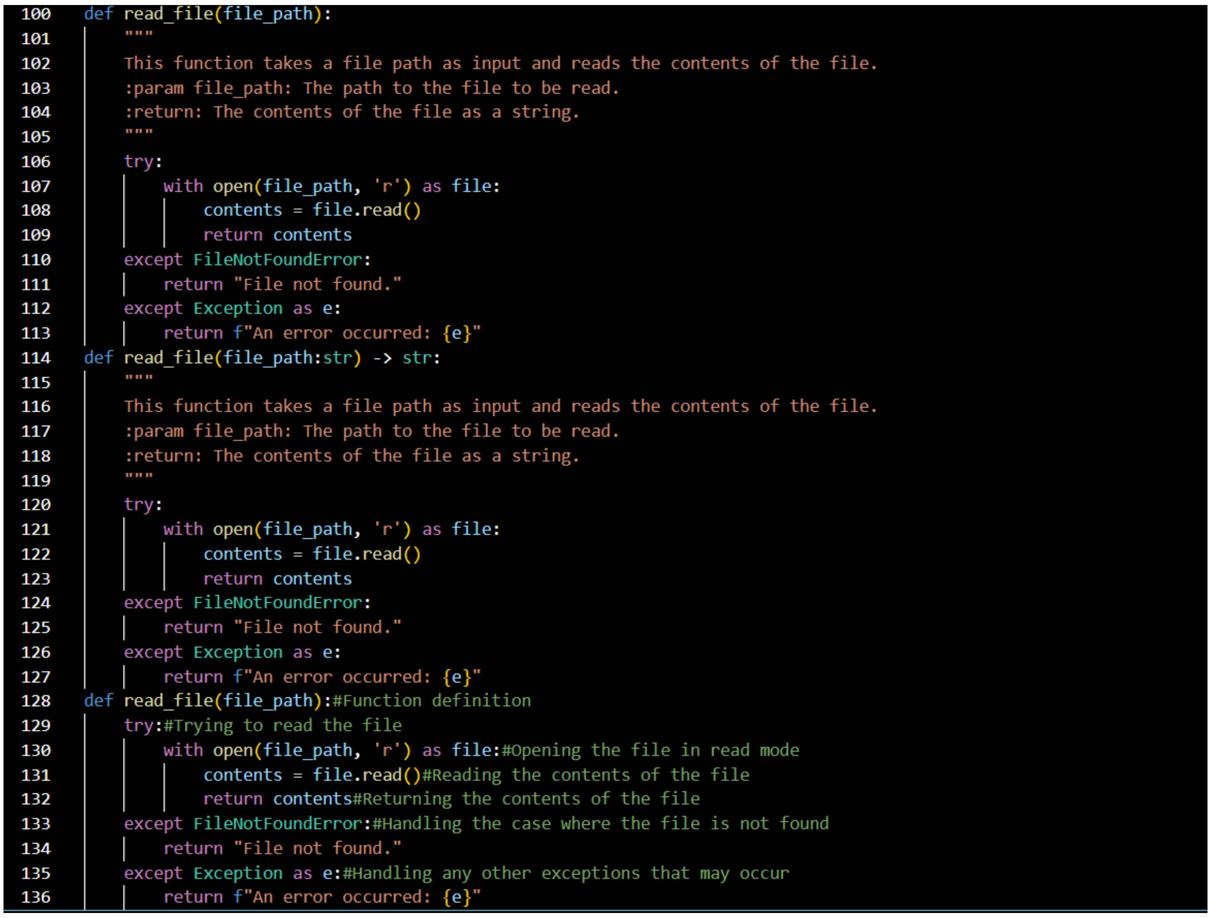
return f.read()

Task:

1. Write documentation using all three formats.

2. Identify which style best explains exception handling.

3. Justify your recommendation.



```python
def read_file(file_path):
    """
    This function takes a file path as input and reads the contents of the file.
    :param file_path: The path to the file to be read.
    :return: The contents of the file as a string.
    """
    try:
        with open(file_path, 'r') as file:
            contents = file.read()
            return contents
    except FileNotFoundError:
        return "File not found."
    except Exception as e:
        return f"An error occurred: {e}"
def read_file(file_path:str) -> str:
    """
    This function takes a file path as input and reads the contents of the file.
    :param file_path: The path to the file to be read.
    :return: The contents of the file as a string.
    """
    try:
        with open(file_path, 'r') as file:
            contents = file.read()
            return contents
    except FileNotFoundError:
        return "File not found."
    except Exception as e:
        return f"An error occurred: {e}"
def read_file(file_path):#Function definition
    try:#Trying to read the file
        with open(file_path, 'r') as file:#Opening the file in read mode
            contents = file.read()#Reading the contents of the file
            return contents#Returning the contents of the file
    except FileNotFoundError:#Handling the case where the file is not found
        return "File not found."
    except Exception as e:#Handling any other exceptions that may occur
        return f"An error occurred: {e}"
```

Inline comments explain specific file operations (like open, read, write) directly in the code.
Standard docstrings describe what the entire file-handling function does.
Google style docstrings clearly document parameters (file name, mode) and return values.
For file handling, clear input/output explanation is very important.
Google style docstring is best because it neatly explains file paths, modes, exceptions, and return values.