AI ASSISTANT CODING

LAB - 9.5

NAME : MD SUBHANI

HTNO : 2303A51207

BATCH : 04

Lab Experiment:

Documentation Generation -Automatic documentation and code comments

Lab Objectives :

1. To understand automatic documentation generation.

2. To generate code comments and docstrings using AI tools.

3. To learn the importance of documentation in software development.

Lab Outcomes :

1. Students will be able to generate documentation automatically for code.

2. Students will be able to add clear comments and docstrings to programs.

3.Students will be able to improve code readability and maintainability using documentation.

Problem 1: String Utilities Function

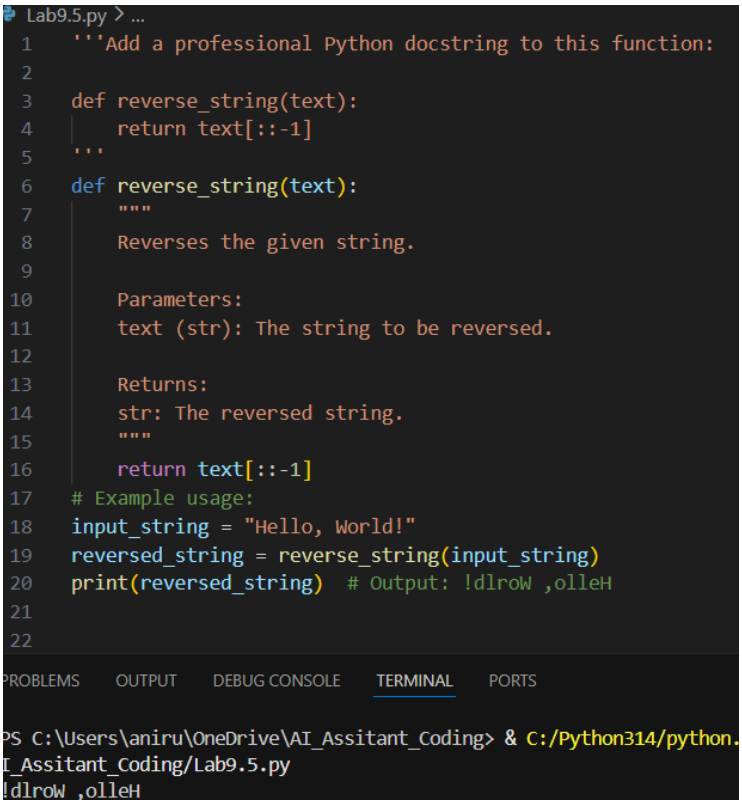
Consider the following Python function:

```
def reverse_string(text):

return text[::-1]
```

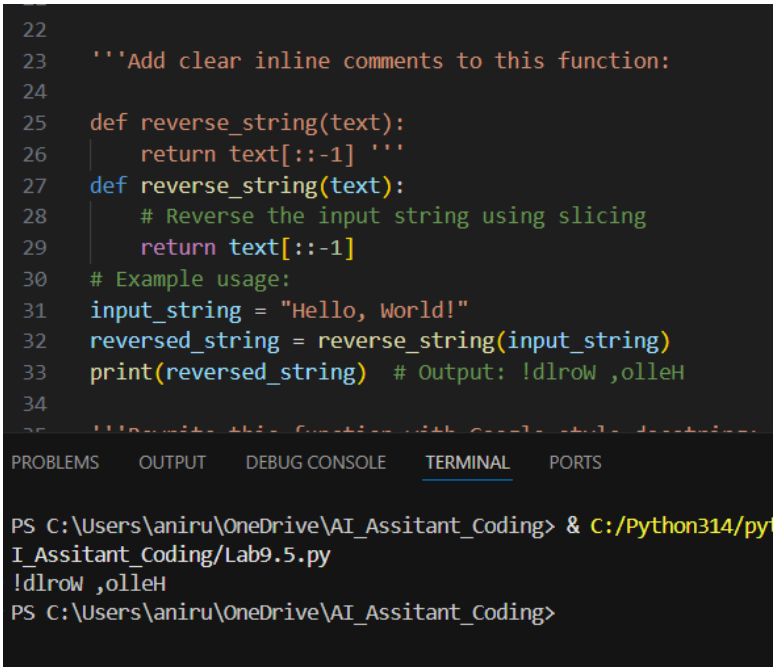
Task:

1. Write documentation in:

(a) Docstring :



(b) Inline comments :



(c) Google-style documentation

```
'''Rewrite this function with Google-style docstring:

def reverse_string(text):
    return text[::-1]'''
def reverse_string(text):
    """Reverses the given string.
    Args:
        text (str): The string to be reversed.
    Returns:
        str: The reversed string.
    """
    return text[::-1]
# Example usage:
input_string = "Hello, World!"
reversed_string = reverse_string(input_string)
print(reversed_string)  # Output: !dlroW ,olleH

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\aniru\OneDrive\AI_Assitant_Coding> & C:/Python314/python.exe c:
I_Assitant_Coding/Lab9.5.py
!dlroW ,olleH
PS C:\Users\aniru\OneDrive\AI_Assitant_Coding>
```

2. Compare the three documentation styles.

```
'''Compare docstring, inline comments, and Google-style documentation.
Explain differences in readability and usefulness.'''

Docstrings, inline comments, and Google-style documentation
serve different purposes in enhancing code readability and usefulness.
```

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

```
'''Which documentation style is best for a utility-based string library?
Justify briefly.'''

For a utility-based string library, Google-style documentation is often the best choice.
This is because it provides a clear and structured format for documenting functions, including
parameters, return values, and any exceptions that may be raised. This level of detail is
particularly useful for a library that may be used by other developers, as it helps them understand
how to use the functions correctly and what to expect from them. Additionally, Google-style documentation
can be easily parsed by tools that generate API documentation, making it more accessible for users of the library.
```

Justification:

Google-style documentation is most suitable for a utility-based string library because it provides a clear, structured, and standardized format. It improves readability, makes functions easier to understand, and works well with automatic documentation tools like pydoc. This style is widely used in professional projects, enhancing maintainability and consistency.

Problem 2: Password Strength Checker

Consider the function:

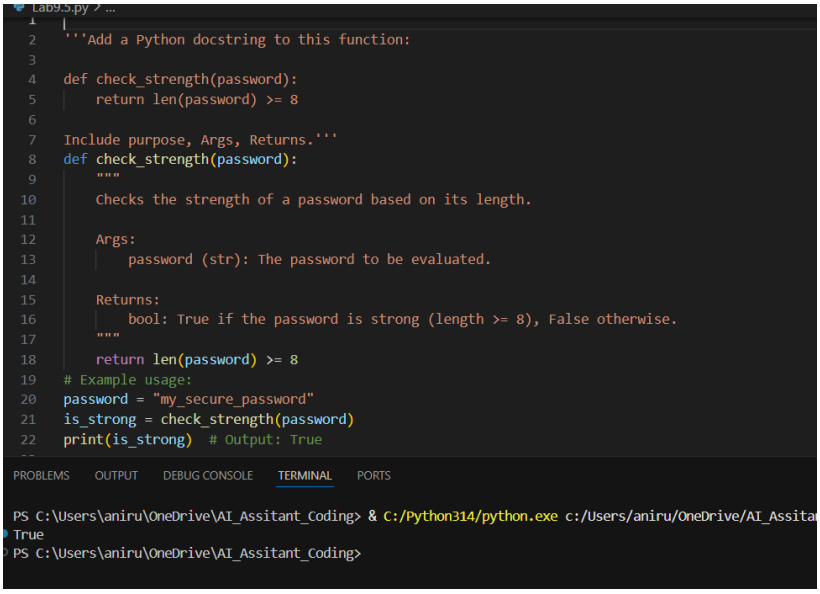
def check_strength(password):

return len(password) >= 8

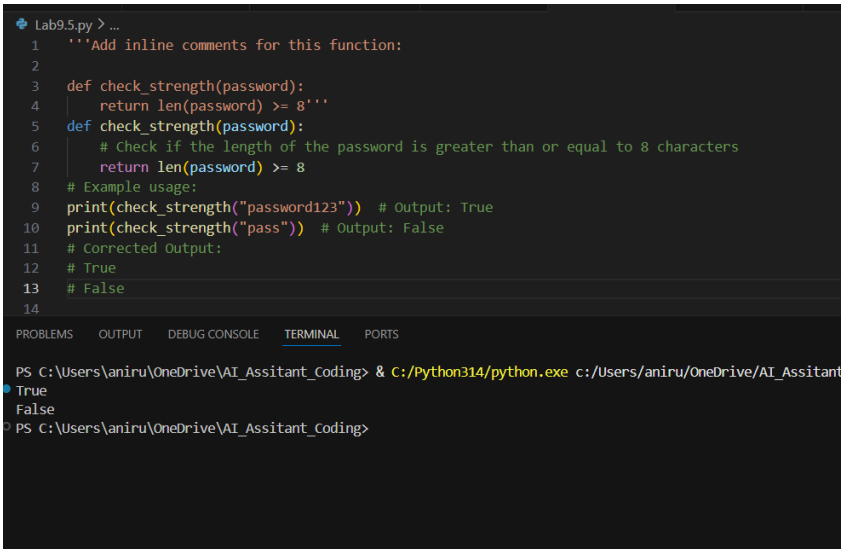
Task:

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

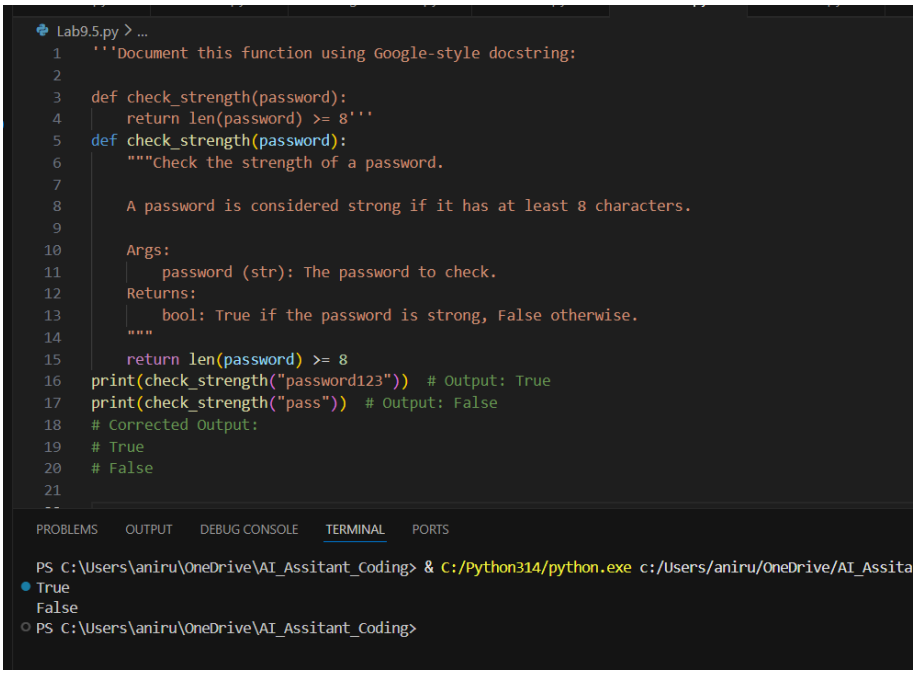
(a) Docstring:



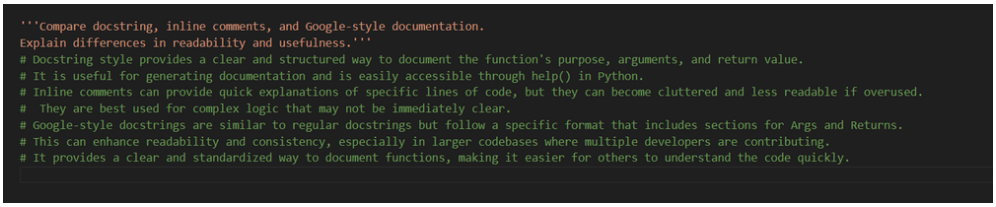
(b) Inline



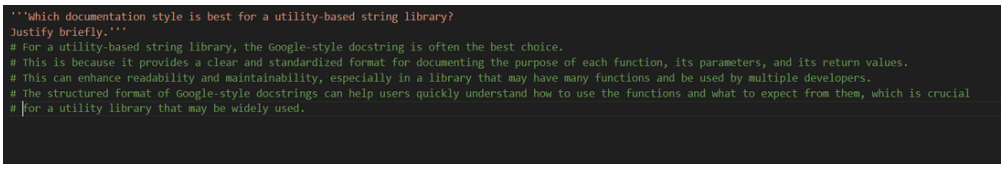
c Google style



2. Compare documentation styles for security-related code.



3. Recommend the most appropriate style.



Justification:

Security-related code requires precise and unambiguous documentation. Google-style docstrings clearly define validation rules, parameters, return values, and potential exceptions. This reduces misuse, helps during code reviews and audits, and ensures developers correctly understand the function's behavior.

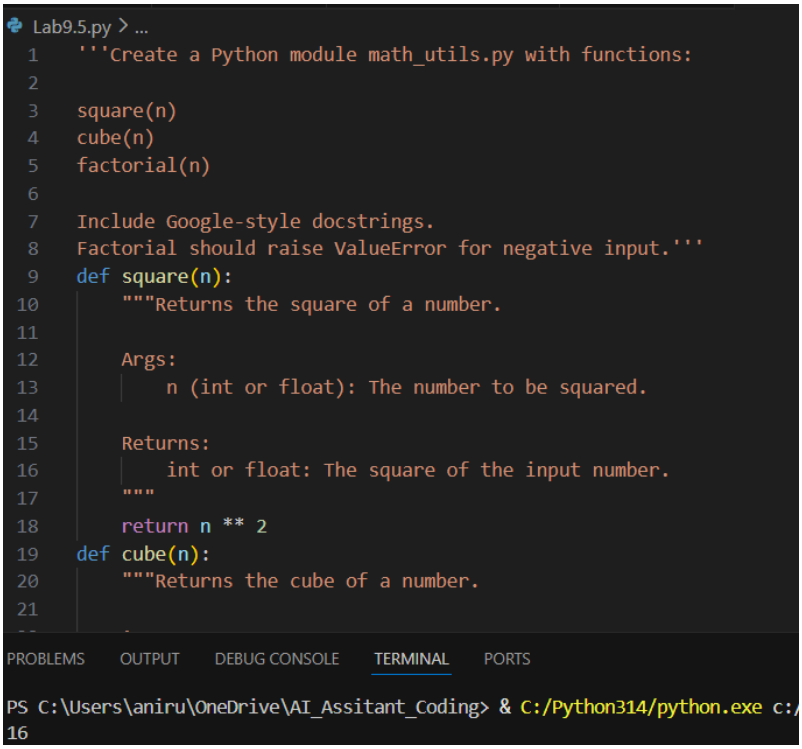
Problem 3: Math Utilities Module

Task:

1. Create a module math_utils.py with functions:

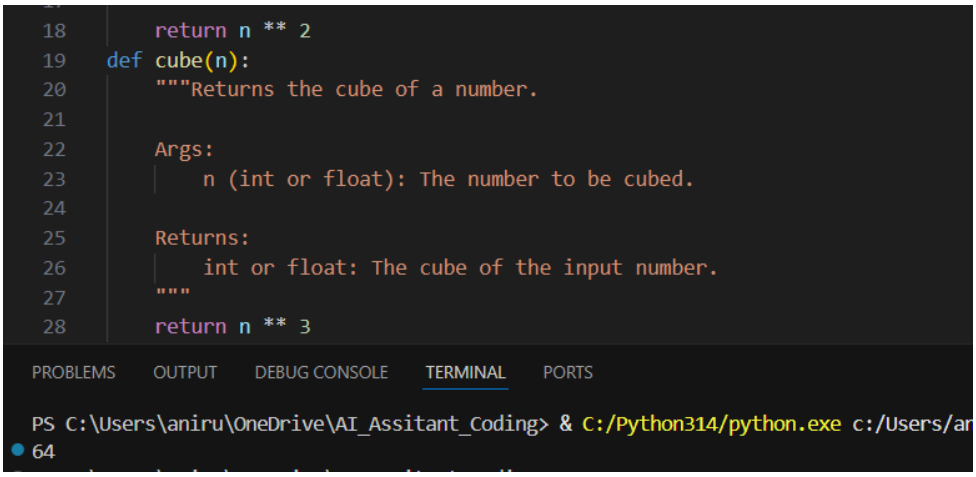
Generate docstrings automatically using AI tools.

o square(n)



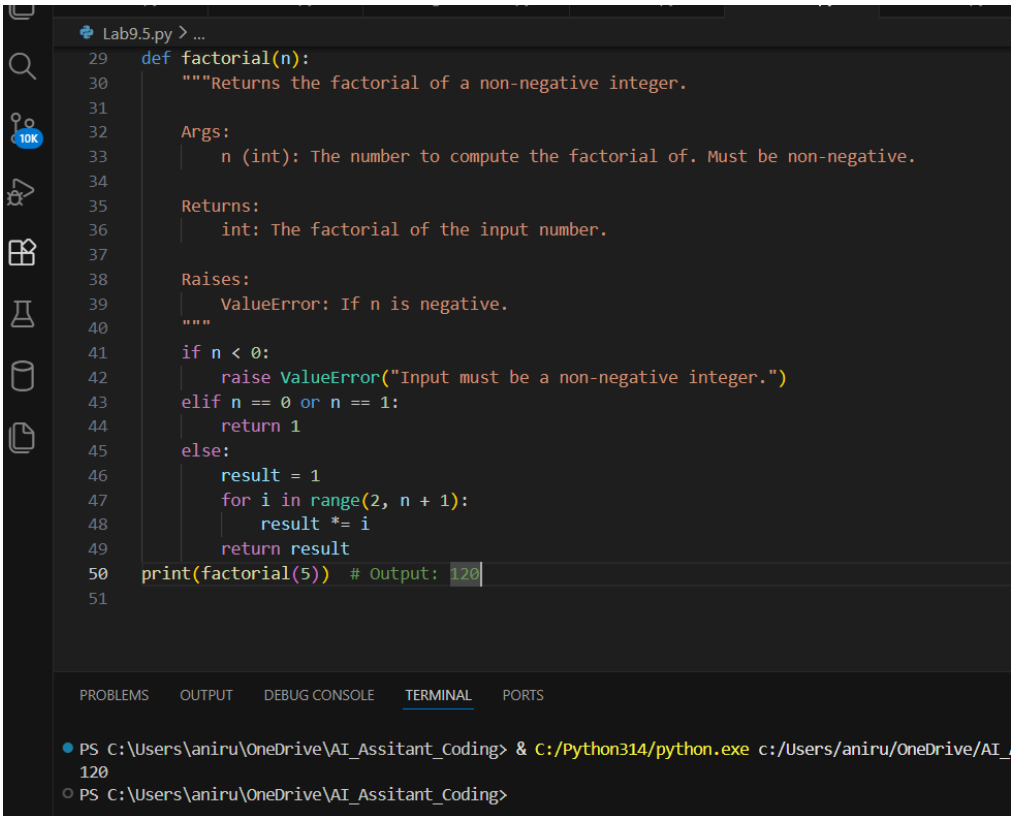
```
Lab9.5.py > ...
'''Create a Python module math_utils.py with functions:

square(n)
cube(n)
factorial(n)

Include Google-style docstrings.
Factorial should raise ValueError for negative input.'''
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 ** 2
def cube(n):
    """Returns the cube of a number.

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\aniru\OneDrive\AI_Assitant_Coding> & C:/Python314/python.exe c:/
16
```

o cube(n)

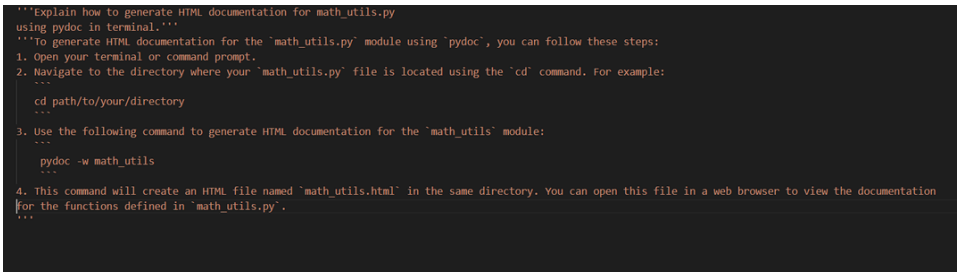


```
    return n ** 2
def cube(n):
    """Returns the cube of a number.

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

    Returns:
        int or float: The cube of the input number.
    """
    return n ** 3

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\aniru\OneDrive\AI_Assitant_Coding> & C:/Python314/python.exe c:/Users/ar
64
```

o factorial(n)



3. Export documentation as an HTML file.



Justification:

Docstrings help describe the purpose, inputs, outputs, and exceptions of each function. They enable automatic documentation generation and make the module easier for other developers to understand and reuse.

Problem 4: Attendance Management Module

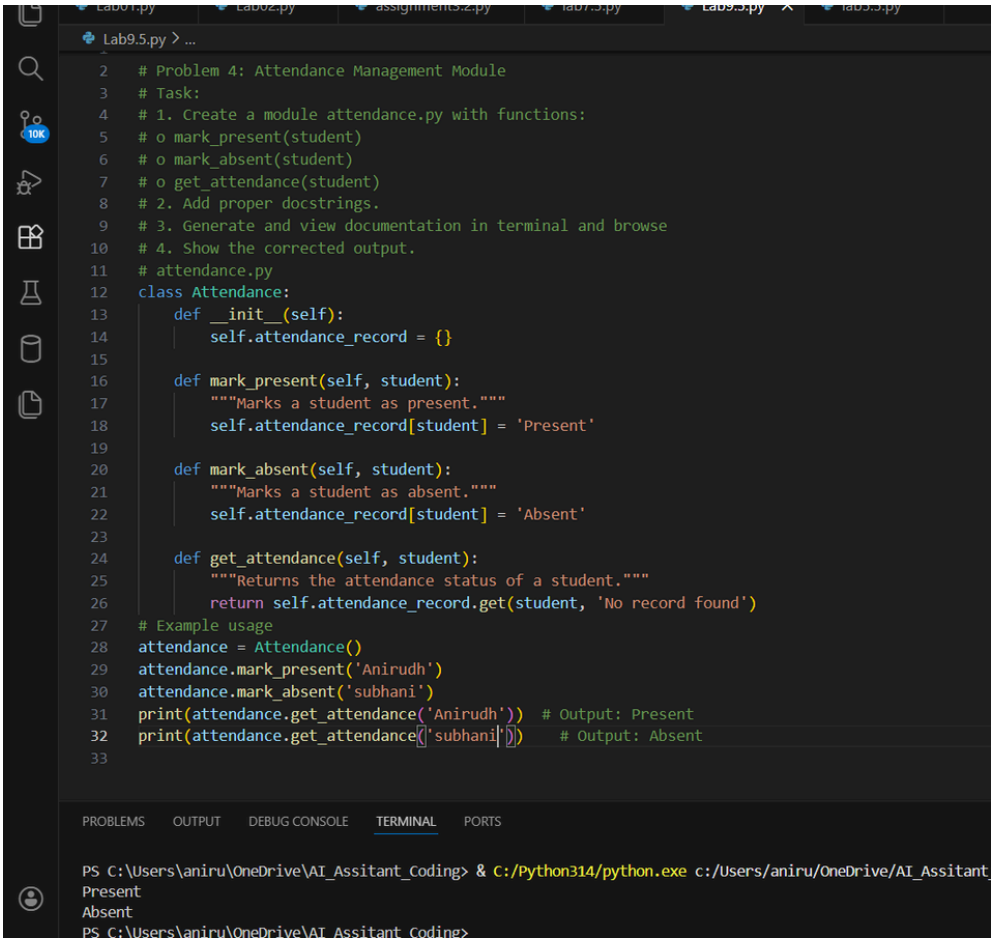
Task:

1. Create a module attendance.py with functions:

o mark_present(student)

o mark_absent(student)

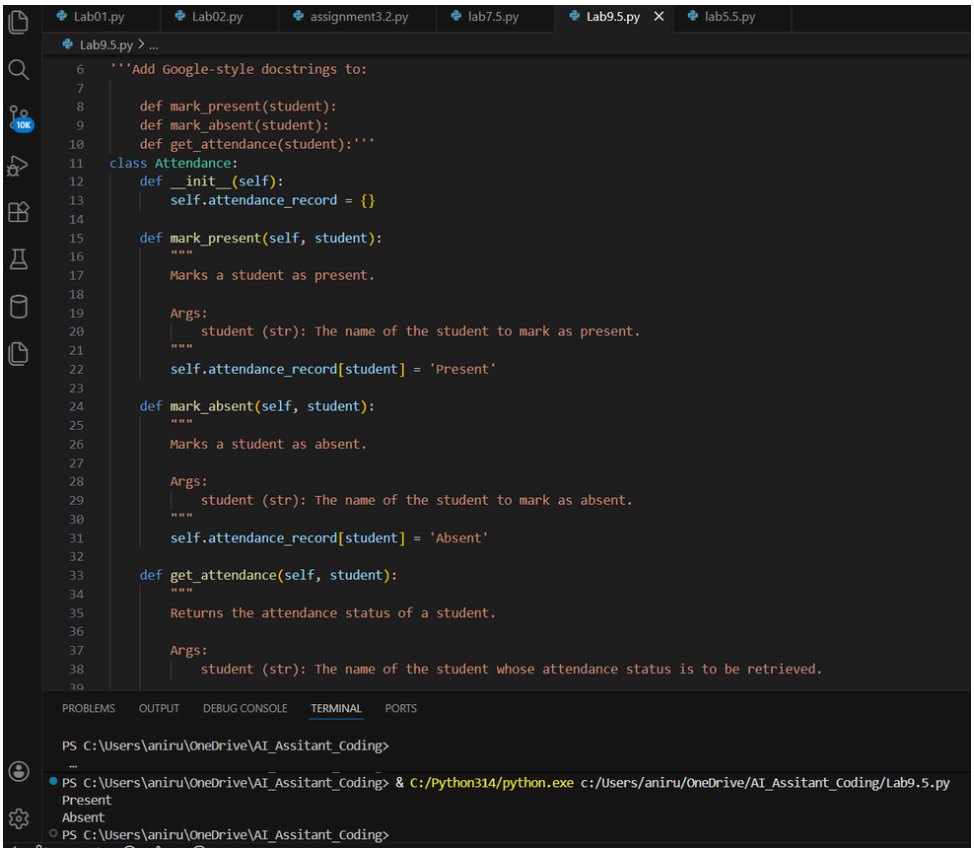
o get_attendance(student)



```python
# 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
# 4. Show the corrected output.
# attendance.py
class Attendance:
    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')
# Example usage
attendance = Attendance()
attendance.mark_present('Anirudh')
attendance.mark_absent('subhani')
print(attendance.get_attendance('Anirudh'))  # Output: Present
print(attendance.get_attendance('subhani'))    # Output: Absent
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\aniru\OneDrive\AI_Assitant_Coding> & C:/Python314/python.exe c:/Users/aniru/OneDrive/AI_Assitant
Present
Absent
PS C:\Users\aniru\OneDrive\AI_Assitant_Coding>
```

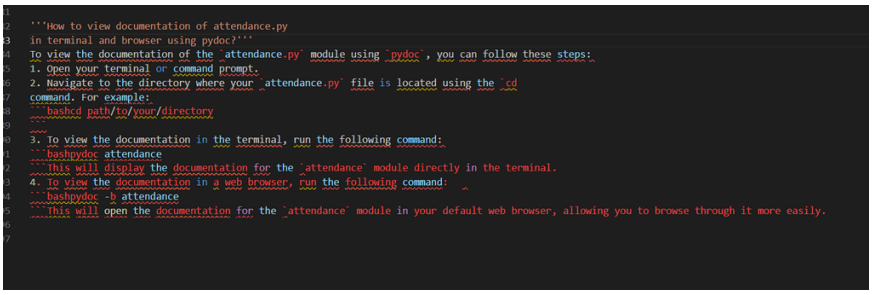
2. Add proper docstrings.



```python
'''Add Google-style docstrings to:

    def mark_present(student):
    def mark_absent(student):
    def get_attendance(student):'''
class Attendance:
    def __init__(self):
        self.attendance_record = {}

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

        Args:
            student (str): The name of the student to mark 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.
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\aniru\OneDrive\AI_Assitant_Coding>
PS C:\Users\aniru\OneDrive\AI_Assitant_Coding> & C:/Python314/python.exe c:/Users/aniru/OneDrive/AI_Assitant_Coding/Lab9.5.py
Present
Absent
PS C:\Users\aniru\OneDrive\AI_Assitant_Coding>
```

3. Generate and view documentation in terminal and browse



Justification :

Documentation ensures developers understand how attendance is stored, updated, and retrieved. It improves collaboration, reduces confusion, and supports maintainability.

Problem 5: File Handling Function

Consider the function:

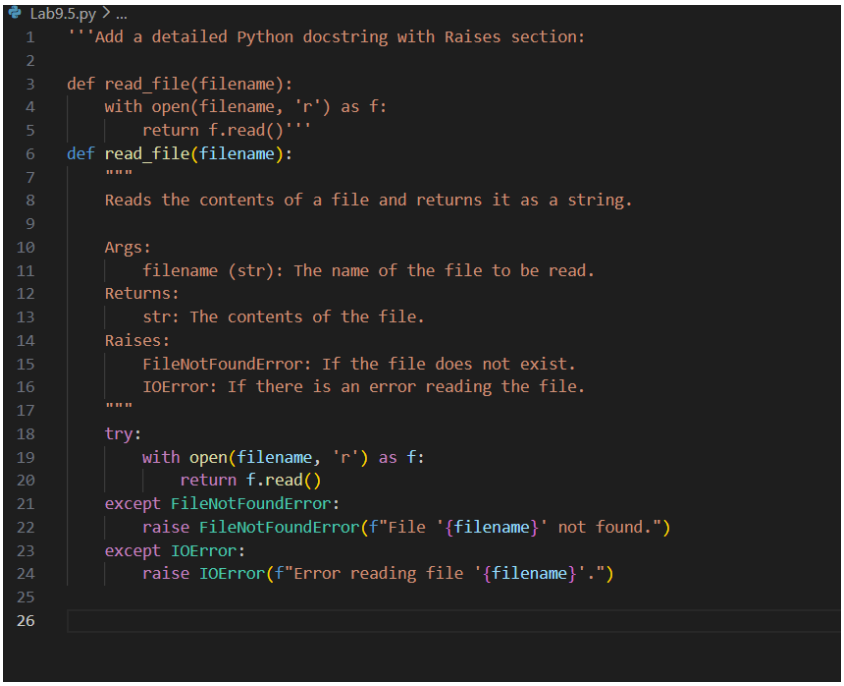
def read_file(filename):

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

return f.read()

Task:

1. Write documentation using all three formats.



```
'''Add inline comments explaining file handling:

def read_file(filename):
    with open(filename, 'r') as f:
        return f.read()'''
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.
    Raises:        FileNotFoundError: If the file does not exist.
        IOError: If there is an error reading the file.
    """
    try:
        # Open the file in read mode ('r')
        with open(filename, 'r') as f:
            # Read the contents of the file and return it
            return f.read()
    except FileNotFoundError:
        # Raise an error if the file is not found
        raise FileNotFoundError(f"File '{filename}' not found.")
    except IOError:
        # Raise an error if there is an issue reading the file
        raise IOError(f"Error reading file '{filename}'.")
```

```
'''Rewrite this function with Google-style docstring
including exception details:

def read_file(filename):
    with open(filename, 'r') as f:
        return f.read()
'''
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.
    Raises:
        FileNotFoundError: If the file does not exist.
        IOError: If there is an error reading the file.
    """
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        raise FileNotFoundError(f"File '{filename}' not found.")
    except IOError:
        raise IOError(f"Error reading file '{filename}'.")
```

2. Identify which style best explains exception handling.

```
'''Which documentation style best explains exception handling?
Compare docstring vs inline vs Google-style.'''

'''The best documentation style for explaining exception handling depends on the context and the audience.
- Docstring: This style is useful for providing a high-level overview of the function, including its purpose, parameters,
    return value, and exceptions. It is typically used in libraries and APIs where users need to understand the function's
behavior without diving into the implementation details.'''
```

3. Justify your recommendation.

```
'''Justify why Google-style documentation is preferred for explaining exceptions.

Google-style documentation is often preferred for explaining exceptions because it provides a clear and structured format
that is easy to read and understand. The "Raises" section in Google-style docstrings explicitly lists the exceptions that
can be raised by the function, along with a brief description of each exception. This makes it easier for developers to quickly
identify potential error conditions and understand how to handle them effectively. Additionally, Google-style documentation is
widely adopted in the Python community, making it a familiar and consistent choice for documenting exceptions.'''
```