

Name: Achi Vaishnavi

Hall Ticket:2303A51554

Batch:29

Assessment:9.5

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 & Output:

```
#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
# # 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 str
```

Significance:

This task helps understand how simple utility functions can be documented using different styles. It shows how docstrings, inline comments, and Google-style documentation

improve code readability and usability. Since string utilities are reused frequently, proper documentation ensures that developers can easily understand the function purpose, parameters, and outputs. It also teaches comparison of documentation methods to select the most effective one for library development.

Task 2: Password Strength Checker

Prompt:

```
def check_strength(password):
```

```
    return len(password) >= 8
```

Write documentation in docstring, inline comments, and Google style.

Compare the three documentation styles for security purposes and recommend appropriate documentation style for this function.

Code & Output:

```
# (a) Docstring
from fileinput import filename
from math import factorial

def check_strength(password):
    """
    Checks the strength of a password.
    Args:
        password (str): The password to be checked.

    Returns:
        bool: True if the password is strong, False otherwise.
    """
    return len(password) >= 8

# (b) Inline comments
# def check_strength(password):
#     # Check if the password length is at least 8 characters
#     return len(password) >= 8

# (c) Google-style documentation
# def check_strength(password):
#     """
#     Checks the strength of a password.
#
#     Args:
#         password (str): The password to be checked.
#
#     Returns:
#         bool: True if the password is strong, False otherwise.
#     """
#     return len(password) >= 8

# Comparison:
# 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.
# The Google-style documentation is similar to the docstring but follows a specific format 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.
# For security purposes, the Google-style documentation is recommended for this function.
```

Significance:

This task emphasizes the importance of clear documentation in security-related code. Password validation functions must be precise and well explained to avoid vulnerabilities. By documenting using multiple styles, students learn how structured documentation improves safety, maintainability, and developer awareness. It highlights that Google-style documentation is most suitable for security functions because it clearly specifies inputs, outputs, and rules.

Task 3: Math Utilities Module

Prompt:

Create a module `math.utils.py` with functions: `square(n)`, `cube(n)`, `factorial(n)` generate docstrings and export documentation as an HTML file

Code&Output:

```
#Create a module math.utils.py with functions:square(n),cube(n),factorial(n)
# generate docstrings and export documentation as an HTML file
```

```
def square(n):
    """
    Returns the square of a number.
    Args:
        n (int): The number to be squared.

    Returns:
        int: The square of the number.
    """
    return n ** 2

def cube(n):
    """
    Returns the cube of a number.
    Args:
        n (int): The number to be cubed.
    Returns:
        int: The cube of the number.
    """
    return n ** 3
def factorial(n):
    """
    Returns the factorial of a number.
    Args:
        n (int): The number to calculate the factorial for.

    Returns:
        int: The factorial of the number.
    """
    return factorial(n)

"""# Export documentation as an HTML file
if __name__ == '__main__':
    import pydoc
    pydoc.writedoc(__name__)
"""
```

Significance:

This task introduces modular programming and automated documentation generation. Creating `math_utils.py` demonstrates how reusable mathematical functions can be organized professionally. Generating AI-based docstrings and exporting HTML documentation teaches real-world API documentation practices used in industry, improving software maintainability and presentation.

Task 4: Attendance Management Module

Prompt:

Create a module `attendance.py` with functions `mark_present(student)`, `mark_absent(student)`, `get_attendance(student)`, add proper docstrings and generate and view documentation in terminal and browse

Code & Output:

```
class Attendance:
    def __init__(self):
        self.attendance = {}
    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[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[student] = 'Absent'
    def get_attendance(self, student):
        """
        Gets the attendance status of a student.
        Args:
            student (str): The name of the student to get the attendance status for.

        Returns:
            str: The attendance status of the student ('Present' or 'Absent').
        """
        return self.attendance.get(student, 'No record')
# Generate and view documentation in terminal
if __name__ == '__main__':
    import pydoc
    print(pydoc.render_doc(Attendance))
```

Significance:

This task simulates a real-world application module. Writing docstrings for attendance functions improves clarity and future scalability. Generating terminal and browser documentation shows how backend modules can be documented and shared. It builds experience in creating structured educational or enterprise systems with proper documentation standards.

Task 5: File Handling Function

Prompt:

Consider the file handling functions `def read_file(filename):` with `open(filename, 'r') as f: return f.read()` write documentation using all three formats and identify which style best explains exception handling and justify your recommendation.

Code & Output:

```
# (a) Docstring
def read_file(filename):
    """
    Reads the contents of a file.
    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.
    """
    with open(filename, 'r') as f:
        return f.read()

# (b) Inline comments
def read_file(filename):
    # Try to open the file and read its contents
    try:
        with open(filename, 'r') as f:
            return f.read()
    # Handle the case where the file does not exist
    except FileNotFoundError:
        print(f"Error: The file '{filename}' was not found.")
    # Handle any other I/O errors that may occur
    except IOError as e:
        print(f"Error reading the file '{filename}': {e}")

# (c) Google-style documentation
def read_file(filename):
    """
    Reads the contents of a file.

    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.
    """
    with open(filename, 'r') as f:
        return f.read()

# The Google-style documentation best explains exception handling as it clearly outlines the exceptions that may be raised by the function, along with a description of each exception.
# This format allows developers to quickly understand the potential errors that may occur and how to handle them, making it easier to write robust code.
```

Significance:

This task highlights the importance of documenting I/O operations and exception behavior. File handling is error-prone, so documentation must explain possible failures. Comparing

documentation styles shows that Google-style best explains exceptions clearly. This teaches defensive programming and prepares students to write robust production-level code.