Assignment- 9.5

Hall Ticket: 2303A51630

Name: K. Nithin Kumar

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.

Code:

(a) Using Docstring

prob1.py > ...

```python
def reverse_string(text):
    """
    Reverses the given string.

    Args:
        text (str): Input string to reverse.

    Returns:
        str: The reversed string.
    """
    return text[::-1]
if __name__ == "__main__":
    print(reverse_string("Hello"))
```

(b) Inline comments

prob1.py > ...

```python
def reverse_string(text):
    # Function to reverse a given string
    # text: input string
    # Returns: reversed string
    return text[::-1]  # Slicing method to reverse string


# Running the function
print(reverse_string("Hello"))
```

(c) Using Google-Style Documentation

```python
def reverse_string(text):
    """
    Reverses the given string.
    Args:
        text (str): Input string to reverse.
    Returns:
        str: The reversed string.
    """
    return text[::-1]
# Running the function
print(reverse_string("Hello"))
```

Output:

```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> python -m pydoc prob1
Help on module prob1:

NAME
    prob1

FUNCTIONS
    reverse_string(text)
        Reverses the given string.

        Args:
            text (str): Input string to reverse.

        Returns:
            str: The reversed string.

FILE
    c:\users\nithin\onedrive\desktop\ai - ass\prob1.py


PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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.

(a) Docstring Style

prob2.py > check_strength

```
1  def check_strength(password):
2      """
3      Checks whether the given password is strong.
4      A password is considered strong if it has
5      at least 8 characters.
6      Parameters:
7      password (str): The password string.
8      Returns:
9      bool: True if strong, False otherwise.
10     """
11     return len(password) >= 8
12 # Running
13 print(check_strength("mypassword"))
14 print(check_strength("pass"))
```

(b) Inline Comments Style

prob2.py > ...

```
1  def check_strength(password):
2      """
3      Checks whether the given password is strong.
4      A password is considered strong if it contains
5      at least 8 characters.
6      Args:
7          password (str): Password provided by the user.
8      Returns:
9          bool: True if password length is >= 8, else False.
10     """
11     return len(password) >= 8
12 # Running
13 print(check_strength("mypassword"))
14 print(check_strength("pass"))
```

prob2.py > ...

```python
def check_strength(password):
    """
    Checks whether the given password is strong.
    A password is considered strong if it contains
    at least 8 characters.
    Args:
        password (str): Password provided by the user.
    Returns:
        bool: True if password length is >= 8, else False.
    """
    return len(password) >= 8
# Running
print(check_strength("mypassword"))
print(check_strength("pass"))
```

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS POSTGRESQL QUERY RESULTS AUGMENT powershell

```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> python -m pydoc prob2
True
False
Help on module prob2:

NAME
    prob2

FUNCTIONS
    check_strength(password)
        Checks whether the given password is strong.
        A password is considered strong if it has
        at least 8 characters.
        Parameters:
        password (str): The password string.
        Returns:
        bool: True if strong, False otherwise.

FILE
    c:\users\nithin\onedrive\desktop\ai - ass\prob2.py
```

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.

prob3.py > factorial

```python
def square(n):
    """
    Returns the square of a number.

    Args:
        n (int or float): Input number.

    Returns:
        int or float: Square of n.
    """
    return n * n
def cube(n):
    """
    Returns the cube of a number.

    Args:
        n (int or float): Input number.

    Returns:
        int or float: Cube of n.
    """
    return n * n * n
def factorial(n):
    """
    Returns the factorial of a non-negative integer.
    Args:
        n (int): Non-negative integer.
    Returns:
        int: Factorial of n.
    Raises:
        ValueError: If n is negative.
    """
    if n < 0:
        raise ValueError("Factorial not defined for negative numbers")
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
if __name__ == "__main__":
    print(square(4))
    print(cube(3))
    print(factorial(5))
```

Outut:

```
PROBLEMS 11  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  POSTGRESQL QUERY RESULTS  AUGMENT

PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> python -m pydoc prob3
Help on module prob3:

NAME
    prob3

FUNCTIONS
    cube(n)
        Returns the cube of a number.

        Args:
            n (int or float): Input number.

        Returns:
            int or float: Cube of n.

    factorial(n)
        Returns the factorial of a non-negative integer.
        Args:
            n (int): Non-negative integer.
        Returns:
            int: Factorial of n.
        Raises:
            ValueError: If n is negative.

    square(n)
        Returns the square of a number.

        Args:
            n (int or float): Input number.

        Returns:
            int or float: Square of n.

FILE
    c:\users\nithin\onedrive\desktop\ai - ass\prob3.py


PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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

prob4.py > ...

```python
attendance_record = {}
def mark_present(student):
    """
    Marks a student as present.
    Args:
        student (str): Name of the student.
    Returns:
        None
    """
    attendance_record[student] = "Present"
def mark_absent(student):
    """
    Marks a student as absent.

    Args:
        student (str): Name of the student.

    Returns:
        None
    """
    attendance_record[student] = "Absent"
def get_attendance(student):
    """
    Retrieves the attendance status of a student.
    Args:
        student (str): Name of the student.
    Returns:
        str: Attendance status ('Present', 'Absent', or 'Not Marked').
    """
    return attendance_record.get(student, "Not Marked")
if __name__ == "__main__":
    mark_present("Raju")
    mark_absent("Ramesh")

    print(get_attendance("Raju"))
    print(get_attendance("Ramesh"))
    print(get_attendance("Suresh"))
```

Output:

```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> python -m pydoc prob4
Help on module prob4:

NAME
    prob4

FUNCTIONS
    get_attendance(student)
        Retrieves the attendance status of a student.
        Args:
            student (str): Name of the student.
        Returns:
            str: Attendance status ('Present', 'Absent', or 'Not Marked').

    mark_absent(student)
        Marks a student as absent.

        Args:
            student (str): Name of the student.

        Returns:
            None

    mark_present(student)
        Marks a student as present.
        Args:
            student (str): Name of the student.
        Returns:
            None

DATA
    attendance_record = {}

FILE
    c:\users\nithin\onedrive\desktop\ai - ass\prob4.py
```

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.

prob4.py > read_file

```python
def read_file(filename):
    """
    Reads and returns the content of a file.
    Args:
        filename (str): Path to the file.
    Returns:
        str: File contents as a string.
    Raises:
        FileNotFoundError: If the file does not exist.
        OSError: If an OS-related error occurs.
    """
    with open(filename, 'r') as f:
        return f.read()

# Running Example
if __name__ == "__main__":
    print(read_file("sample.txt"))
```

prob5.py > ...

```python
def read_file(filename):
    """
    Reads and returns file content.
    Args:
        filename (str): File name.
    Returns:
        str: File data.
    Raises:
        FileNotFoundError: If file not found.
    """

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

if __name__ == "__main__":
    try:
        print(read_file("sample.txt"))
    except Exception as e:
        print("Error:", e)
```

prob4.py > ...

```python
def read_file(filename):
    # Function to read file content
    # filename: name of the file
    # May raise FileNotFoundError if file not found
    with open(filename, 'r') as f:  # open file in read mode
        return f.read()  # return entire content


# Running Example
if __name__ == "__main__":
    print(read_file("sample.txt"))
```