Assignment: 9.5

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:

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

    Args:
        text (str): The input string to be reversed.

    Returns:
        str: A new string with characters in reverse order.

    Example:
        >>> reverse_string("hello")
        'olleh'
    """
```

```
    return text[::-1]


# Example usage
if __name__ == "__main__":
    user_input = input("Enter a string: ")
    print("Reversed string:", reverse_string(user_input))
```

Output:

```
Enter a string: Python
Reversed string: nohtyP
```

Justification:

Google-style documentation is best for a password strength checker because it clearly defines inputs, outputs, and behavior. It improves readability, supports security reviews, follows professional standards, works with documentation tools, and makes future updates easier.

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

Code:

```
def check_strength(password):
    """
    Checks whether the given password meets minimum strength criteria.

    A password is considered strong if it contains
    at least 8 characters.

    Args:
        password (str): The password string to evaluate.

    Returns:
        bool: True if password length is >= 8, False otherwise.

    Example:
        >>> check_strength("mypassword")
        True
        >>> check_strength("pass")
        False
    """
    # Check if password length is at least 8 characters
    return len(password) >= 8


# Example usage
if __name__ == "__main__":
    user_password = input("Enter password: ")
    if check_strength(user_password):
```

```
    print("Password is Strong")
else:
    print("Password is Weak")
```

output:

```
Enter password: well
Password is Weak
```

Justification:

Google-style documentation is the most suitable choice for security-related code like a password strength checker. It clearly specifies the input parameter and return type, which reduces confusion and misuse of the function. Since security functions may be reviewed or audited, structured documentation improves readability and understanding.

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.

Code:

```
# Math Utilities Module
import math

def calculate_distance(point1, point2):
```

```
    """Calculate the Euclidean distance between two points."""
    return math.sqrt((point2[0] - point1[0]) ** 2 + (point2[1] - point1[1]) ** 2)

def calculate_angle(point1, point2):
    """Calculate the angle in degrees between two points."""
    return math.degrees(math.atan2(point2[1] - point1[1], point2[0] - point1[0]))

def normalize_vector(vector):
    """Normalize a 2D vector."""
    magnitude = math.sqrt(vector[0] ** 2 + vector[1] ** 2)
    if magnitude == 0:
        return (0, 0)
    return (vector[0] / magnitude, vector[1] / magnitude)

def dot_product(vector1, vector2):
    """Calculate the dot product of two 2D vectors."""
    return vector1[0] * vector2[0] + vector1[1] * vector2[1]

def cross_product(vector1, vector2):
    """Calculate the cross product of two 2D vectors."""
    return vector1[0] * vector2[1] - vector1[1] * vector2[0]

def rotate_point(point, angle):
    """Rotate a point by a given angle in degrees."""
    radians = math.radians(angle)
    cos_angle = math.cos(radians)
    sin_angle = math.sin(radians)
    return (
        point[0] * cos_angle - point[1] * sin_angle,
        point[0] * sin_angle + point[1] * cos_angle
    )

def clamp(value, min_value, max_value):
    """Clamp a value between a minimum and maximum."""
    return max(min_value, min(value, max_value))
```

```
# Example Usage
if __name__ == "__main__":
    print("Distance:", calculate_distance((0, 0), (3, 4)))
    print("Angle:", calculate_angle((0, 0), (1, 1)))
    print("Normalize:", normalize_vector((3, 4)))
    print("Dot Product:", dot_product((1, 2), (3, 4)))
    print("Cross Product:", cross_product((1, 2), (3, 4)))
    print("Rotate Point:", rotate_point((1, 0), 90))
    print("Clamp:", clamp(15, 0, 10))
```

Output:

```
Distance: 5.0
Angle: 45.0
Normalize: (0.6, 0.8)
Dot Product: 11
Cross Product: -2
Rotate Point: (6.123233995736766e-17, 1.0)
Clamp: 10
```

Justification:

Using AI-generated docstrings ensures clear, consistent, and professional documentation. Exporting to HTML provides a structured, readable format suitable for submission, sharing, and future maintenance.

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

Code:

```
#Attendance Management Module
class AttendanceManagement:
    def __init__(self):
        self.attendance_records = {}

    def mark_attendance(self, student_id, date, status):
        if student_id not in self.attendance_records:
            self.attendance_records[student_id] = {}
        self.attendance_records[student_id][date] = status

    def get_attendance(self, student_id):
        return self.attendance_records.get(student_id, {})
    def calculate_attendance_percentage(self, student_id):
        records = self.get_attendance(student_id)
        total_classes = len(records)
        attended_classes = sum(1 for status in records.values() if status
== 'Present')
        if total_classes == 0:
            return 0
        return (attended_classes / total_classes) * 100
    def generate_attendance_report(self, student_id):
        records = self.get_attendance(student_id)
        report = f"Attendance Report for Student ID: {student_id}\n"
        report += "Date\t\tStatus\n"
        for date, status in records.items():
            report += f"{date}\t{status}\n"
        report += f"Attendance Percentage:
{self.calculate_attendance_percentage(student_id):.2f}%\n"
        return report
# Example usage
attendance_manager = AttendanceManagement()
attendance_manager.mark_attendance('S001', '2024-09-01', 'Present')
attendance_manager.mark_attendance('S001', '2024-09-02', 'Absent')
print(attendance_manager.generate_attendance_report('S001'))
```

Output:

```
PS C:\Users\DELL\OneDrive\Documents\New Folder> &
ments/New Folder/ai code"
Attendance Report for Student ID: S001
Date            Status
2024-09-01      Present
2024-09-02      Absent
Attendance Percentage: 50.00%
```

Justification:

Proper docstrings improve clarity, maintainability, and usability. Generating documentation in terminal and HTML format provides professional, structured output suitable for projects and future reference.

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

Code:

```
#File Handling Function

def read_file(file_path):

    try:

        with open(file_path, 'r') as file:

            content = file.read()

            return content

    except FileNotFoundError:

        print(f"Error: The file '{file_path}' was not found.")

    except IOError:

        print(f"Error: An error occurred while reading the file
'{file_path}'.")

def write_file(file_path, content):

    try:

        with open(file_path, 'w') as file:

            file.write(content)

            print(f"Content successfully written to '{file_path}'.")

    except IOError:

        print(f"Error: An error occurred while writing to the file
'{file_path}'.")

#Example Usage

if __name__ == "__main__":

    # Writing to a file

    write_file('example.txt', 'Hello, this is a sample text file.')
```

```
# Reading from a file
content = read_file('example.txt')
if content:
    print("File Content:")
    print(content)
```

output:

```
PS C:\Users\DELL\OneDrive\Documents\New Folder> & C:\Use
ments/New Folder/ai code"
Content successfully written to 'example.txt'.
File Content:
Hello, this is a sample text file.
PS C:\Users\DELL\OneDrive\Documents\New Folder>
```

Justification:

Google-style documentation is best because it clearly separates and defines **Args, Returns, and Raises** sections. This structured format makes exception handling explicit and easy to understand, which is very important in file handling and error-prone operations. It improves readability, supports professional standards, and works well with documentation tools.