AI ASSISTED CODING

LAB ASSIGNMENT 9.5

LAB EXPERIMENT 9.5: Documentation Generation – Automatic Documentation and Code Comments

PROBLEM 1: String Utilities Function

QUESTION

Consider the function:

```
def reverse_string(text):
    return text[::-1]
```

Write documentation in:

1. Docstring
2. Inline comments
3. Google-style documentation

Compare the three documentation styles and recommend the most suitable style for a utility-based string library.

PROMPT

Generate proper documentation for the given string utility function in three formats: standard docstring, inline comments, and Google-style documentation. Then compare the formats and recommend the best style.

CODE

(a) Docstring format

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

    Parameters:
    text (str): The string to be reversed.

    Returns:
    str: The reversed string.
    """
    return text[::-1]
```

(b) Inline comments

```
def reverse_string(text):
    # This function takes a string as input
    # It reverses the string using slicing
    return text[::-1]  # Return reversed string
```

(c) Google-style Documentation

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

    Args:
        text (str): Input string.

    Returns:
        str: Reversed version of input string.
    """
    return text[::-1]
```

OUTPUT

```
PS D:\3-2 SEM> python
olleH
PS D:\3-2 SEM>
```

EXPLANATION

Docstrings provide structured documentation that can be accessed using help() function.

Inline comments are simple but not suitable for generating automated documentation.
Google-style documentation is structured, readable, and widely used in industry.

Recommended Style: Google-style documentation is most suitable for utility-based libraries because it is clean, standardized, and compatible with documentation tools.

Task 2: PASSWORD STRENGTH CHECKER

QUESTION

def check_strength(password):

    return len(password) >= 8

Document the function using docstring, inline comments, and Google style. Compare the styles and recommend the best one for security-related code.

PROMPT

Generate structured documentation for a password strength checker function in three documentation formats and compare them.

CODE

(a) Docstring format

```
def check_strength(password):
    """
    Checks whether the password meets minimum length requirements.

    Parameters:
    password (str): The password entered by user.

    Returns:
    bool: True if password length is at least 8 characters.
    """
    return len(password) >= 8
```

(b) Inline Comments

```python
def check_strength(password):
    # Check if password length is at least 8 characters
    return len(password) >= 8
```

(c) Google-Style Documentation

```python
def check_strength(password):
    """Checks password strength based on minimum length.

    Args:
        password (str): User-provided password.

    Returns:
        bool: True if password length >= 8, else False.
    """
    return len(password) >= 8
```

```python
print(check_strength("mypassword"))
```

OUTPUT

```
PS D:\3-2 SEM> python
True
PS D:\3-2 SEM>
```

EXPLANATION

Security-related functions require clarity and precision.
Google-style documentation clearly defines arguments and return types.
Inline comments are insufficient for professional security systems.

Recommended Style: Google-style documentation because security code requires clarity and maintainability.

Task 3: Math Utilities Module

QUESTION

Create a module math_utils.py with:

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

Generate docstrings and export documentation as HTML.

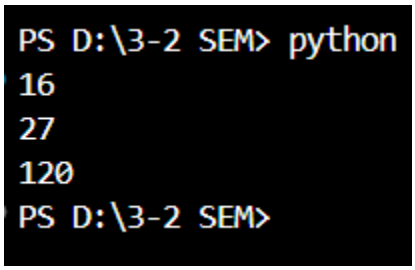
PROMPT

Create a math utilities module with proper documentation and generate HTML documentation using Python documentation tools.

CODE (math_utils.py)

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

    Args:
        n (int): Input number.

    Returns:
        int: Square of n.
    """
    return n * n


def cube(n):
    """Returns the cube of a number.

    Args:
        n (int): Input number.

    Returns:
        int: Cube of n.
    """
    return n * n * n


def factorial(n):
    """Returns factorial of a number.

    Args:
        n (int): Non-negative integer.

    Returns:
        int: Factorial of n.
    """
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

print(square(4))      # 16
print(cube(3))        # 27
print(factorial(5))   # 120
```

OUTPUT



EXPLANATION

Docstrings allow automatic documentation generation using pydoc.
Exporting to HTML improves usability and readability.
Structured documentation improves maintainability of reusable modules.

Task 4: Attendance Management Module

QUESTION

Create attendance.py module with:

- mark_present(student)
- mark_absent(student)
- get_attendance(student)

Add proper docstrings and generate documentation.

PROMPT

Develop an attendance management module with documented functions and generate documentation viewable in terminal and browser.

CODE

```python
attendance_record = {}
def mark_present(student):
    """Marks a student as present.

    Args:
        student (str): Student name.
    """
    attendance_record[student] = "Present"

def mark_absent(student):
    """Marks a student as absent.

    Args:
        student (str): Student name.
    """
    attendance_record[student] = "Absent"

def get_attendance(student):
    """Returns attendance status of a student.

    Args:
        student (str): Student name.

    Returns:
        str: Attendance status.
    """
    return attendance_record.get(student, "No record found")

mark_present("Sai")
print(get_attendance("Sai"))
```

OUTPUT

```
PS D:\3-2 SEM>
Present
```

EXPLANATION

Adding docstrings makes the module self-documented.
Documentation tools allow viewing structured information directly in terminal or browser.
This improves maintainability and readability.

Task 5: File handling function

QUESTION

Consider the function:

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

Write documentation using all three formats. Identify which style best explains exception handling and justify your recommendation.

PROMPT

Generate structured documentation for a file reading function in three formats and determine which documentation style best explains exception handling.

CODE

(a) Docstring Format

```
def read_file(filename):
    """
    Reads and returns content of a file.

    Parameters:
    filename (str): Name of file to read.

    Returns:
    str: File content.

    Raises:
    FileNotFoundError: If file does not exist.
    """
    with open(filename, 'r') as f:
        return f.read()
```

(b) Inline comments

```
def read_file(filename):
    # Opens the file in read mode
    # Returns file content
    with open(filename, 'r') as f:
        return f.read()
```

(c) Google-Style Documentation

```
def read_file(filename):
    """Reads and returns file content.

    Args:
        filename (str): File name to read.

    Returns:
        str: File content.

    Raises:
        FileNotFoundError: If file is not found.
    """
    with open(filename, 'r') as f:
        return f.read()
```

```
print(read_file("sample.txt"))
```

OUTPUT

```
Displays the file content
```

EXPLANATION

Exception handling must be clearly documented.
Inline comments do not formally describe possible errors.
Google-style documentation includes structured Raises section, making it clear and professional.

Recommendation: Google-style documentation is most suitable for explaining exception handling because it clearly defines parameters, returns, and possible errors in a structured manner.
