

ASSIGNMENT – 9.5

NAME: HABEEBA KHANAM

HT.NO: 2303A51474

BT.NO: 29

LAB EXPERIMENT : Lab Experiment: Documentation Generation – Automatic documentation and code comments.

PROBLEM -01:

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:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "pytests" containing files like "pytests.py", "9.2.py", and several "Untitled" files.
- Search Bar:** Contains the word "Search".
- Code Editor:** Displays three examples of Python docstring styles:
 - Style 1: Docstring Documentation:** A multi-line docstring with sections for `Args:`, `Returns:`, and `Example:`.
 - Style 2: Inline Comments Documentation:** A multi-line docstring using inline comments (`#`) to describe the function's purpose and parameters.
 - Style 3: Google-Style Documentation:** A single-line docstring with a detailed description of the function's behavior.
- Sessions:** Shows a session titled "9.2.py" with a note to "Describe what to build next".
- Bottom Status Bar:** Shows file path ("C:\Users\DELL\OneDrive\Desktop\9.2.py"), line number (Line 60), column (Col 33), spaces (Spaces: 4), and encoding (UTF-8). It also includes tabs for "Python", "C", "C++", "C#", "Java", "JavaScript", "TypeScript", "Go", and "Live Server".
- Bottom Icons:** Includes icons for file operations, search, and various extensions like Git, GitHub, and others.

The screenshot shows the Visual Studio Code interface with a Python file named '9.2.py' open. The code defines a function 'reverse_string_google' that takes a string and returns its reverse. A docstring is provided at the top of the function. Below the code, there is a comparison section with three parts: Docstring Style, Inline Comments, and Google-Style. The Google-Style section is highlighted. The status bar at the bottom shows the file is 27 lines long, has 60 columns, 33 rows, and is in Python mode.

```

34 def reverse_string_google(text):
35     """
36         Args:
37             text (str): The input string to be reversed.
38         Returns:
39             str: The reversed string.
40         Raises:
41             TypeError: If text is not a string.
42         Example:
43             >>> reverse_string_google("Hello world")
44             'dlrow olleh'
45     """
46
47     return text[::-1]
48
49 # COMPARISON & RECOMMENDATION
50 #
51 # -----
52 #
53 # COMPARISON:
54 # 1. DOCSTRING STYLE:
55 #   - Concise and built-in to Python
56 #   - Accessible via help() and __doc__
57 #   - Good for quick reference
58 #   - Less suitable for large projects
59 # 2. INLINE COMMENTS:
60 #   - Explains logic step-by-step
61 #   - Good for complex algorithms
62 #   - Hard to maintain in large codebases
63 #   - Not parsed by documentation tools
64 # 3. GOOGLE-STYLE:
65 #   - Industry standard and widely recognized
66 #   - Works with documentation generators (Sphinx, pdoc)
67 #   - Clear structure with Args, Returns, Raises
68 #   - Best for team projects and libraries
69 # RECOMMENDATION:
70 # For a utility-based string library: GOOGLE-STYLE is most suitable.
71 # Reasons:
72 # ✓ Professional and consistent across large codebases

```

COMPARISON OF DOCUMENTATION STYLES :

1. Docstring Style
Concise and built into Python. Accessible using `help()` and `__doc__`. Good for small projects, but less structured for large systems.
2. Inline Comments
Explain code logic step-by-step. Useful for complex algorithms, but difficult to maintain in large codebases and not supported by documentation tools.
3. Google-Style Documentation
Structured and industry standard. Includes sections like Args and Returns. Works well with documentation tools and team projects.

RECOMMENDATION

For a utility-based string library, Google-style documentation is best because it is professional, structured, tool-friendly, and suitable for large or public projects.

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:

```
File Edit Selection View ... Search Chat Sessions
EXPLORER OPEN EDITORS 7 unsaved NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Opening a folder will close all currently open editors. To keep them open, add a folder instead.
Clone Repository
You can clone a repository locally.
Create Java Project
TO LEARN ABOUT HOW TO USE GIT AND SOURCE CONTROL IN VS CODE READ OUR DOCS.
You can also open a Java project folder, or create a new Java project by clicking the button below.
Create Java Project
OUTLINE TIMELINE MAVEN
pytest.py 9.2.py Untitled-4 Untitled-5 Untitled-3 Untitled-2 Settings
1 # =====
2 # STYLE 1: INLINE COMMENTS
3 #
4 # def check_strength_inline(password):
5 #     # Check if password length is at least 8 characters for basic security
6 #     return len(password) >= 8
7 #
8 # =====
9 #
10 # =====
11 # STYLE 2: DOCSTRING (PEP 257)
12 #
13 # def check_strength_docstring(password):
14 #     """
15 #         Validate password strength based on minimum length requirement.
16 #
17 #     Args:
18 #         password: A string representing the user's password.
19 #
20 #     Returns:
21 #         bool: True if password length is 8 or more characters, False otherwise.
22 #
23 #         return len(password) >= 8
24 #
25 # =====
26 # STYLE 3: GOOGLE STYLE DOCSTRING
27 #
28 # def check_strength_google(password):
29 #     """Checks if a password meets minimum strength requirements.
30 #
31 #     Validates password strength by verifying minimum length of 8 characters.
32 #     This is a basic security check and should be combined with additional validation rules (uppercase, numbers, special characters) in production.
33 #
34 #     Args:
35 #         password: str - The password string to validate.
36 #
37 #     Returns:
38 #         bool: True if password length >= 8 characters, False otherwise.
39 #
40 #     Raises:
41 #         TypeError: If password is not a string.
42 #
43 #     Example:
44 #         >>> check_strength_google("MyPass123")
45 #             True
46 #         >>> check_strength_google("short")
47 #             False
48 #
49 #     return len(password) >= 8
50 #
51 # =====
52 # RECOMMENDATION FOR SECURITY-RELATED CODE:
53 #
54 # Google Style Docstring is most appropriate because:
55 # 1. Explicit parameter types document security requirements
56 # 2. Returns section clarifies boolean logic
57 # 3. Example section shows expected behavior clearly
58 # 4. Can include security warnings and limitations
59 # 5. Industry standard for sensitive functions
60 #
61 #
62 # =====
```

This screenshot shows the Visual Studio Code interface with several tabs open. The main editor contains Python code for password strength checking, demonstrating three different documentation styles: inline comments, PEP 257 docstrings, and Google-style docstrings. The code includes examples and a recommendation for security-related code.

```
File Edit Selection View ... Search Chat Sessions
EXPLORER OPEN EDITORS 7 unsaved NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Opening a folder will close all currently open editors. To keep them open, add a folder instead.
Clone Repository
You can clone a repository locally.
Create Java Project
TO LEARN ABOUT HOW TO USE GIT AND SOURCE CONTROL IN VS CODE READ OUR DOCS.
You can also open a Java project folder, or create a new Java project by clicking the button below.
Create Java Project
OUTLINE TIMELINE MAVEN
pytest.py 9.2.py Untitled-4 Untitled-5 Untitled-3 Untitled-2 Settings
31 def check_strength_google(password):
32     """
33     Args:
34         password: str - The password string to validate.
35     Returns:
36         bool: True if password length >= 8 characters, False otherwise.
37     Raises:
38         TypeError: If password is not a string.
39     Example:
40         >>> check_strength_google("MyPass123")
41         True
42         >>> check_strength_google("short")
43         False
44     return len(password) >= 8
45 #
46 # =====
47 # RECOMMENDATION FOR SECURITY-RELATED CODE:
48 #
49 # Google Style Docstring is most appropriate because:
50 # 1. Explicit parameter types document security requirements
51 # 2. Returns section clarifies boolean logic
52 # 3. Example section shows expected behavior clearly
53 # 4. Can include security warnings and limitations
54 # 5. Industry standard for sensitive functions
55 #
56 # =====
```

This screenshot shows the Visual Studio Code interface with several tabs open. The main editor contains Python code for password strength checking, specifically using Google-style docstrings. The code includes detailed annotations for parameters, returns, raises, examples, and a recommendation for security-related code.

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:

The screenshot shows the VS Code interface with the following code in the editor:

```
1 """
2 Math Utilities Module
3 A collection of mathematical utility functions for common operations.
4 """
5 def square(n):
6     """
7         Calculate the square of a number.
8
9     Args:
10        n (int or float): The number to square.
11
12    Returns:
13        int or float: The square of the input number.
14
15    Example:
16        >>> square(5)
17        25
18
19    return n * n
20 def cube(n):
21     """
22         Calculate the cube of a number.
23
24     Args:
25        n (int or float): The number to cube.
26
27    Returns:
28        int or float: The cube of the input number.
29
30    Example:
31        >>> cube(3)
32        27
33
34    return n * n * n
35 def factorial(n):
36     """
37         Calculate the factorial of a non-negative integer.
```

The code includes three functions: `square`, `cube`, and `factorial`. Each function has a docstring with examples. The code is saved in a file named `math_utils.py`.

The screenshot shows the Visual Studio Code (VS Code) interface. The main area displays a Python script named `cube.py` with the following content:

```
def cube(n):
    """Calculate the factorial of a non-negative integer.

    Args:
        n (int): A non-negative integer.

    Returns:
        int: The factorial of n.

    Raises:
        ValueError: If n is negative.

    Example:
        >>> factorial(5)
        120
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    if n == 0 or n == 1:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
```

The left sidebar shows the Explorer view with "7 unsaved" files. The status bar at the bottom indicates the file is in Python mode, has 3:14.2 lines, and was last modified at 10:25 PM on 20-02-2026.

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:

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** Search
- Editor:** pytest.py (1-37 lines), Untitled-4 (27-37 lines), Untitled-5, Untitled-3, Untitled-2, Settings
- Left Sidebar (EXPLORER):**
 - Open Editors: 7 unsaved
 - No Folder Opened
 - Open Folder (button)
 - Clone Repository (button)
 - Create Java Project (button)
- Bottom Status Bar:** Ln 65, Col 47, Spaces:4, UTF-8, CRLF, Python 3.14.2, Go Live, ENG IN, 10:27 PM, 20-02-2026

The code in `pytest.py` is as follows:

```
1 """
2 Attendance Management Module
3
4 This module provides functions to manage student attendance records.
5 """
6
7 # Dictionary to store attendance records
8 attendance_records = {}
9
10
11 def mark_present(student):
12     """
13         Mark a student as present.
14
15     Args:
16         student (str): The name of the student to mark as present.
17
18     Returns:
19         str: Confirmation message that the student is marked present.
20
21     Example:
22     >>> mark_present("Alice")
23     'Alice marked present'
24
25     if student not in attendance_records:
26         attendance_records[student] = []
27     attendance_records[student].append("Present")
28     return f"{student} marked present"
29
30
31 def mark_absent(student):
32     """
33         Mark a student as absent.
34
35     Args:
36         student (str): The name of the student to mark as absent.
37
```

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** Search
- Editor:** pytest.py (31-65 lines), Untitled-4 (27-37 lines), Untitled-5, Untitled-3, Untitled-2, Settings
- Left Sidebar (EXPLORER):**
 - Open Editors: 7 unsaved
 - No Folder Opened
 - Open Folder (button)
 - Clone Repository (button)
 - Create Java Project (button)
- Bottom Status Bar:** Ln 65, Col 47, Spaces:4, UTF-8, CRLF, Python 3.14.2, Go Live, ENG IN, 10:28 PM, 20-02-2026

The code in `pytest.py` is as follows:

```
31 def mark_absent(student):
32     """
33         Mark a student as absent.
34
35     Args:
36         student (str): The name of the student to mark as absent.
37
38     Returns:
39         str: Confirmation message that the student is marked absent.
40
41     Example:
42     >>> mark_absent("Bob")
43     'Bob marked absent'
44
45     if student not in attendance_records:
46         attendance_records[student] = []
47     attendance_records[student].append("Absent")
48     return f"{student} marked absent"
49
50
51 def get_attendance(student):
52     """
53         Retrieve attendance records for a student.
54
55     Args:
56         student (str): The name of the student.
57
58     Returns:
59         list: A list of attendance records for the student, or empty list if no records exist.
60
61     Example:
62     >>> get_attendance("Alice")
63     ['Present', 'Present', 'Absent']
64
65     return attendance_records.get(student, [])
```

Problem 5: File Handling Function

Consider the function:

```
def read_file(filename):
```

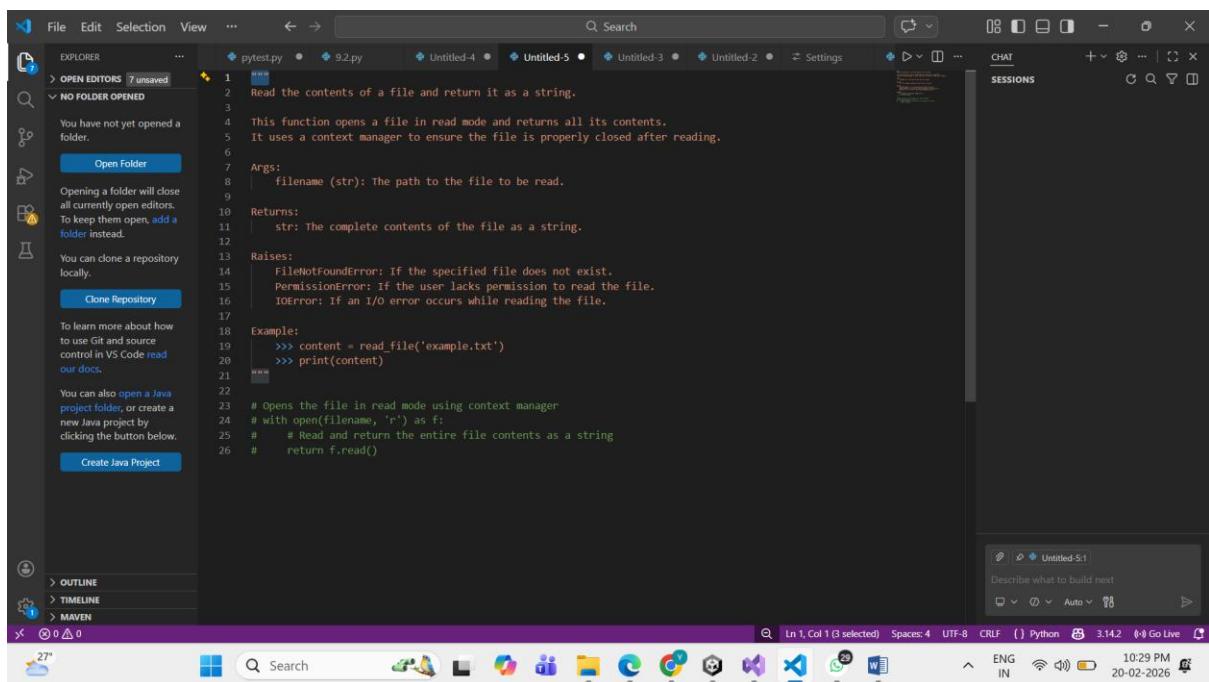
with open(filename, 'r') as f:

```
    return f.read()
```

Task:

1. Write documentation using all three formats.
2. Identify which style best explains exception handling.
3. Justify your recommendation.

CODE:



The screenshot shows the Visual Studio Code (VS Code) interface. The main area displays a Python script named `read_file.py` with the following content:

```

1 """ Read the contents of a file and return it as a string.
2
3 This function opens a file in read mode and returns all its contents.
4 It uses a context manager to ensure the file is properly closed after reading.
5
6 Args:
7     filename (str): The path to the file to be read.
8
9 Returns:
10    str: The complete contents of the file as a string.
11
12 Raises:
13     FileNotFoundError: If the specified file does not exist.
14     PermissionError: If the user lacks permission to read the file.
15     IOError: If an I/O error occurs while reading the file.
16
17 Example:
18     >>> content = read_file('example.txt')
19     >>> print(content)
20
21
22 # Opens the file in read mode using context manager
23 # with open(filename, 'r') as f:
24 #     # Read and return the entire file contents as a string
25 #         return f.read()
26

```

The left sidebar shows the Explorer, Outline, Timeline, and Maven sections. The bottom status bar includes the file path (`Untitled-5:1`), line and column information (`In 1, Col 1 (3 selected)`), and various system and application icons.