

# ASSIGNMENT – 9.5

NAME: HABEEBA KHANAM

HT.NO: 2303A51474

BT.NO: 29

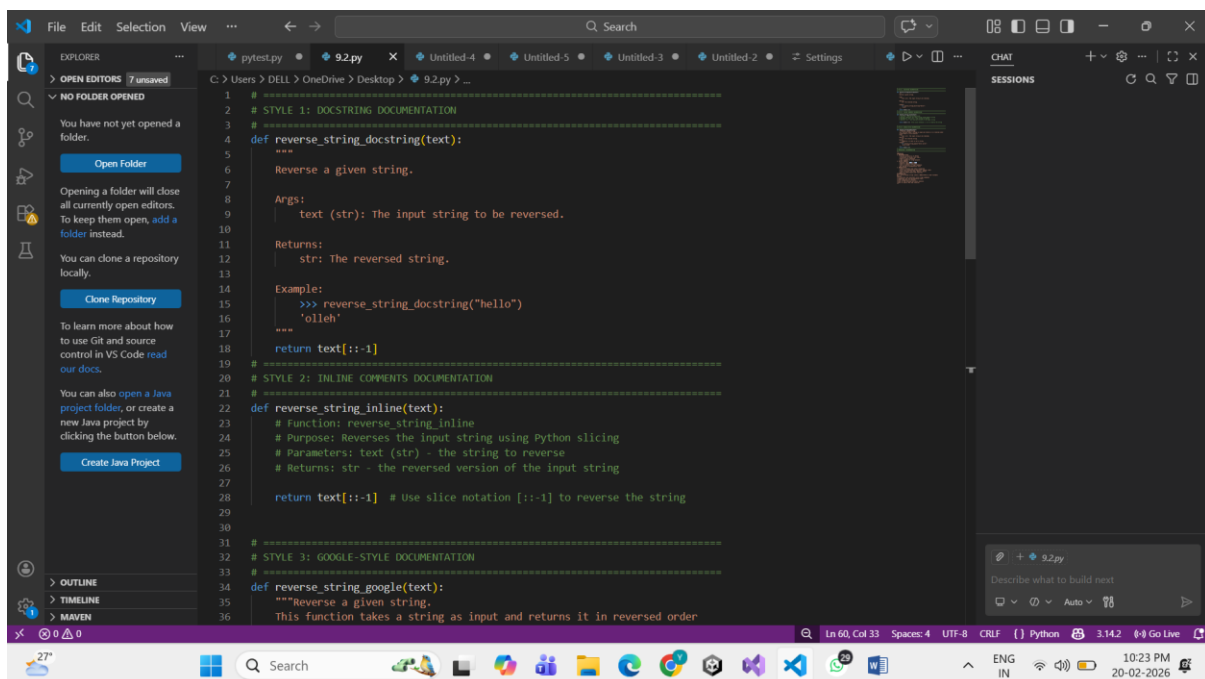
**LAB EXPERIMENT :** Lab Experiment: Documentation Generation – Automatic documentation and code comments.

## PROBLEM -01:

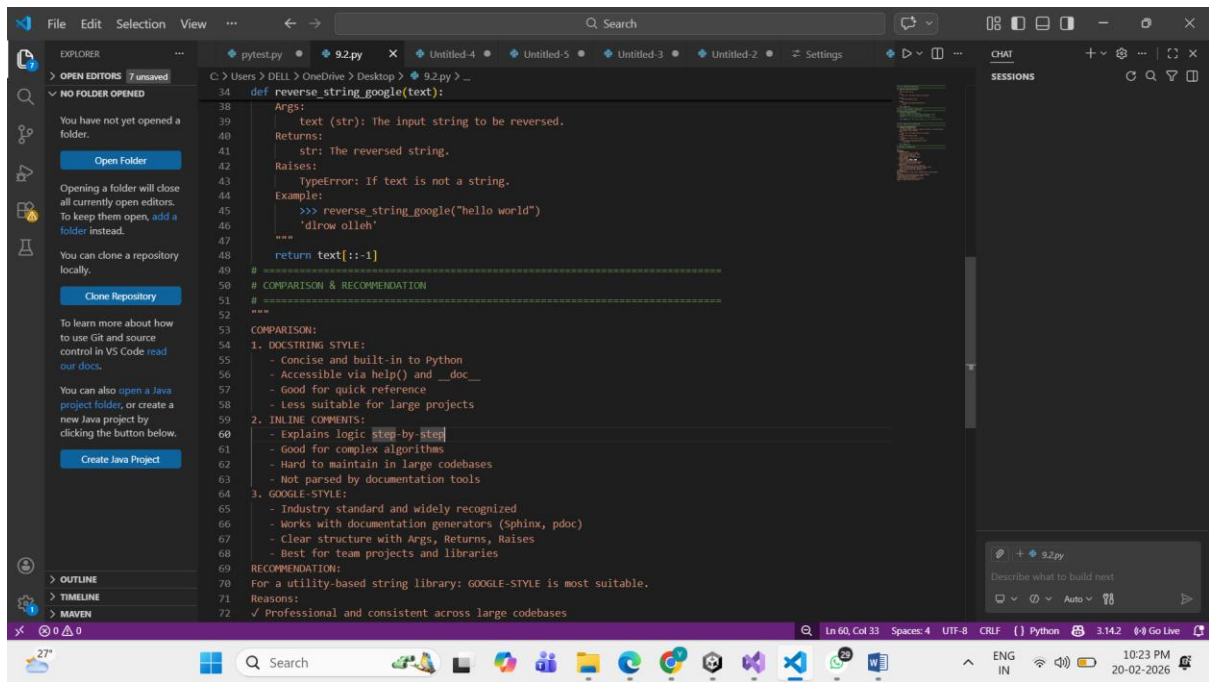
### Task:

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:



```
1 # =====  
2 # STYLE 1: DOCSTRING DOCUMENTATION  
3 # =====  
4 def reverse_string_docstring(text):  
5     """  
6     Reverse a given string.  
7  
8     Args:  
9         text (str): The input string to be reversed.  
10  
11     Returns:  
12         str: The reversed string.  
13  
14     Example:  
15     >>> reverse_string_docstring("hello")  
16     'olleh'  
17     """  
18     return text[::-1]  
19 # =====  
20 # STYLE 2: INLINE COMMENTS DOCUMENTATION  
21 # =====  
22 def reverse_string_inline(text):  
23     # Function: reverse_string_inline  
24     # Purpose: Reverses the input string using Python slicing  
25     # Parameters: text (str) - the string to reverse  
26     # Returns: str - the reversed version of the input string  
27     return text[::-1] # Use slice notation [::-1] to reverse the string  
28  
29  
30 # =====  
31 # STYLE 3: GOOGLE-STYLE DOCUMENTATION  
32 # =====  
33  
34 def reverse_string_google(text):  
35     """Reverse a given string.  
36     This Function takes a string as input and returns it in reversed order
```



## 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:

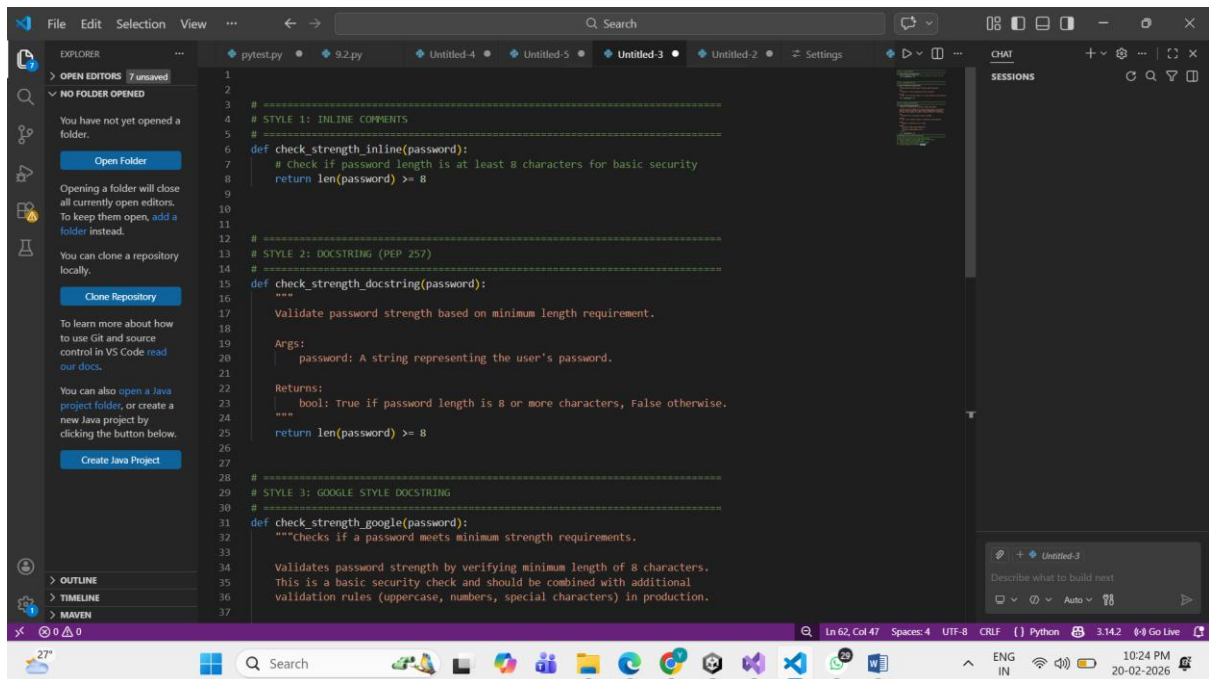
1. Document the function using docstring, inline comments, and

Google style.

2. Compare documentation styles for security-related code.

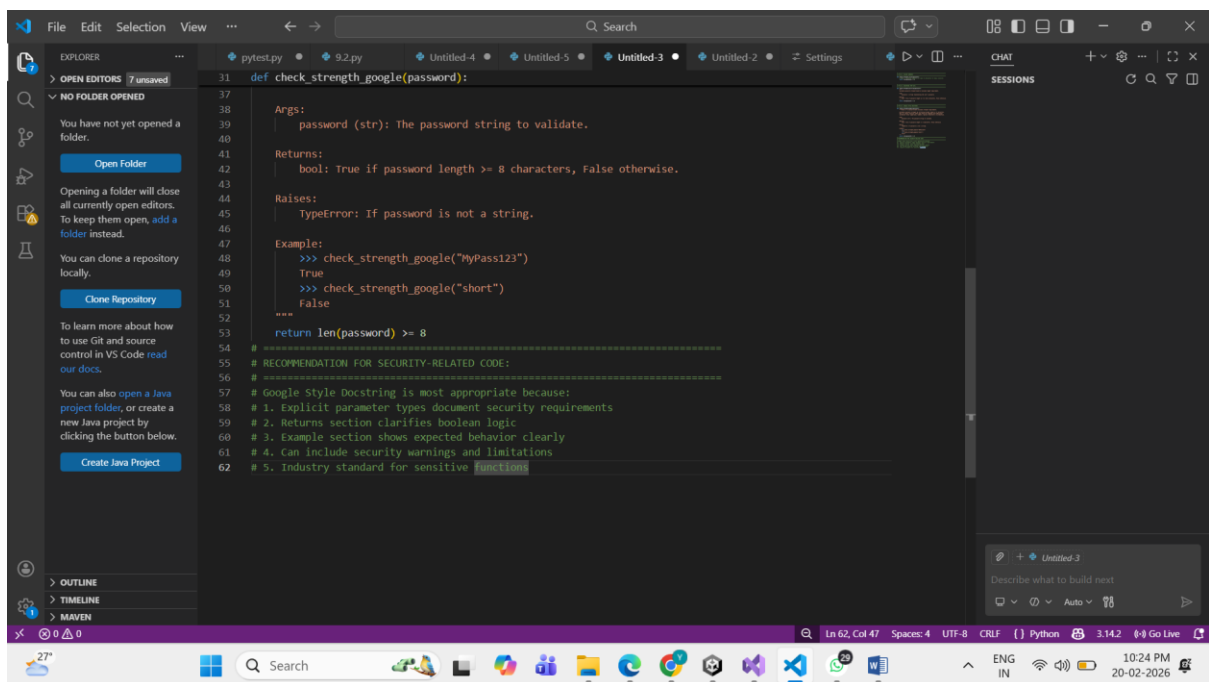
3. Recommend the most appropriate style.

CODE:



The screenshot shows the VS Code editor with a file explorer on the left and a chat panel on the right. The main editor area displays three different documentation styles for a function named `check_strength`. The first style is inline comments, the second is a docstring following PEP 257, and the third is a Google-style docstring. The status bar at the bottom indicates the current file is `pytest.py` and the editor is in `UTF-8` encoding.

```
1 # =====  
2 # STYLE 1: INLINE COMMENTS  
3 # =====  
4  
5 def check_strength_inline(password):  
6     # check if password length is at least 8 characters for basic security  
7     return len(password) >= 8  
8  
9  
10  
11  
12 # =====  
13 # STYLE 2: DOCSTRING (PEP 257)  
14 # =====  
15  
16 def check_strength_docstring(password):  
17     """  
18     Validate password strength based on minimum length requirement.  
19  
20     Args:  
21         password: A string representing the user's password.  
22  
23     Returns:  
24         bool: True if password length is 8 or more characters, False otherwise.  
25     """  
26     return len(password) >= 8  
27  
28 # =====  
29 # STYLE 3: GOOGLE STYLE DOCSTRING  
30 # =====  
31  
32 def check_strength_google(password):  
33     """Checks if a password meets minimum strength requirements.  
34  
35     Validates password strength by verifying minimum length of 8 characters.  
36     This is a basic security check and should be combined with additional  
37     validation rules (uppercase, numbers, special characters) in production.  
38  
39     Args:  
40         password (str): The password string to validate.  
41  
42     Returns:  
43         bool: True if password length >= 8 characters, False otherwise.  
44  
45     Raises:  
46         TypeError: If password is not a string.  
47  
48     Example:  
49     >>> check_strength_google("MyPass123")  
50     True  
51     >>> check_strength_google("short")  
52     False  
53     """  
54     return len(password) >= 8  
55  
56 # =====  
57 # RECOMMENDATION FOR SECURITY-RELATED CODE:  
58 # =====  
59 # Google Style Docstring is most appropriate because:  
60 # 1. Explicit parameter types document security requirements  
61 # 2. Returns section clarifies boolean logic  
62 # 3. Example section shows expected behavior clearly  
63 # 4. Can include security warnings and limitations  
64 # 5. Industry standard for sensitive functions
```



The screenshot shows the VS Code editor with a file explorer on the left and a chat panel on the right. The main editor area displays the Google-style docstring for the function `check_strength_google`. The status bar at the bottom indicates the current file is `pytest.py` and the editor is in `UTF-8` encoding.

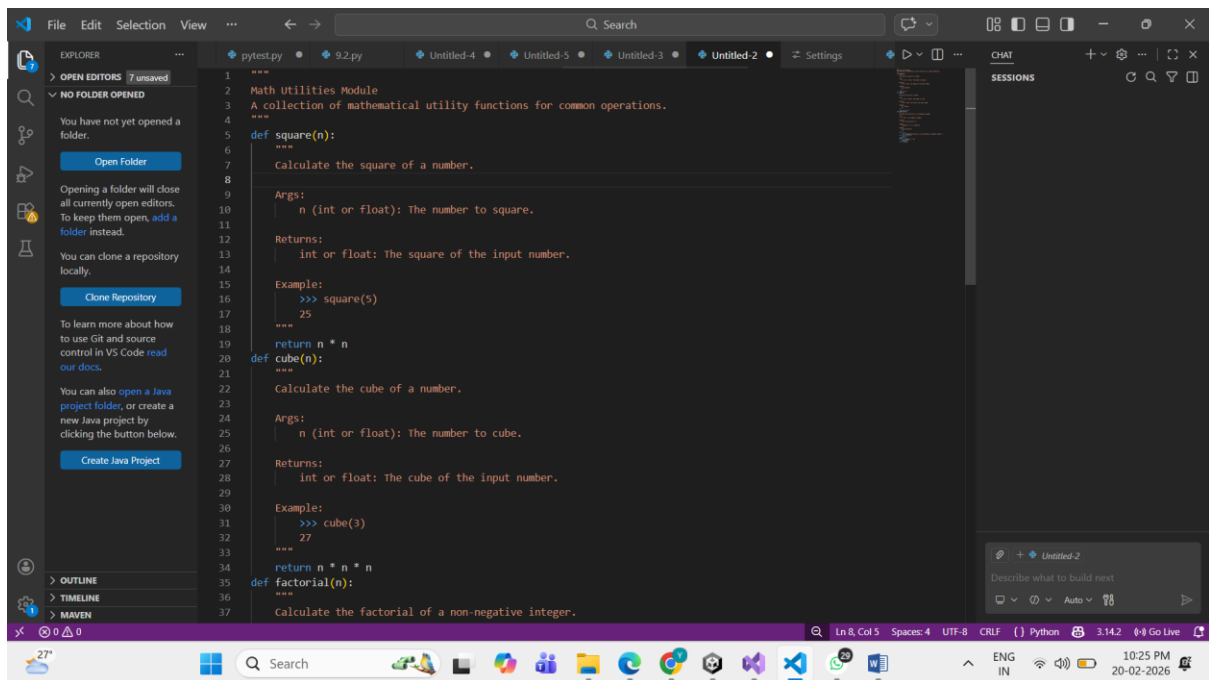
```
31 def check_strength_google(password):  
32  
33     """Checks if a password meets minimum strength requirements.  
34  
35     Validates password strength by verifying minimum length of 8 characters.  
36     This is a basic security check and should be combined with additional  
37     validation rules (uppercase, numbers, special characters) in production.  
38  
39     Args:  
40         password (str): The password string to validate.  
41  
42     Returns:  
43         bool: True if password length >= 8 characters, False otherwise.  
44  
45     Raises:  
46         TypeError: If password is not a string.  
47  
48     Example:  
49     >>> check_strength_google("MyPass123")  
50     True  
51     >>> check_strength_google("short")  
52     False  
53     """  
54     return len(password) >= 8  
55  
56 # =====  
57 # RECOMMENDATION FOR SECURITY-RELATED CODE:  
58 # =====  
59 # Google Style Docstring is most appropriate because:  
60 # 1. Explicit parameter types document security requirements  
61 # 2. Returns section clarifies boolean logic  
62 # 3. Example section shows expected behavior clearly  
63 # 4. Can include security warnings and limitations  
64 # 5. Industry standard for sensitive functions
```

## Problem 3: Math Utilities Module

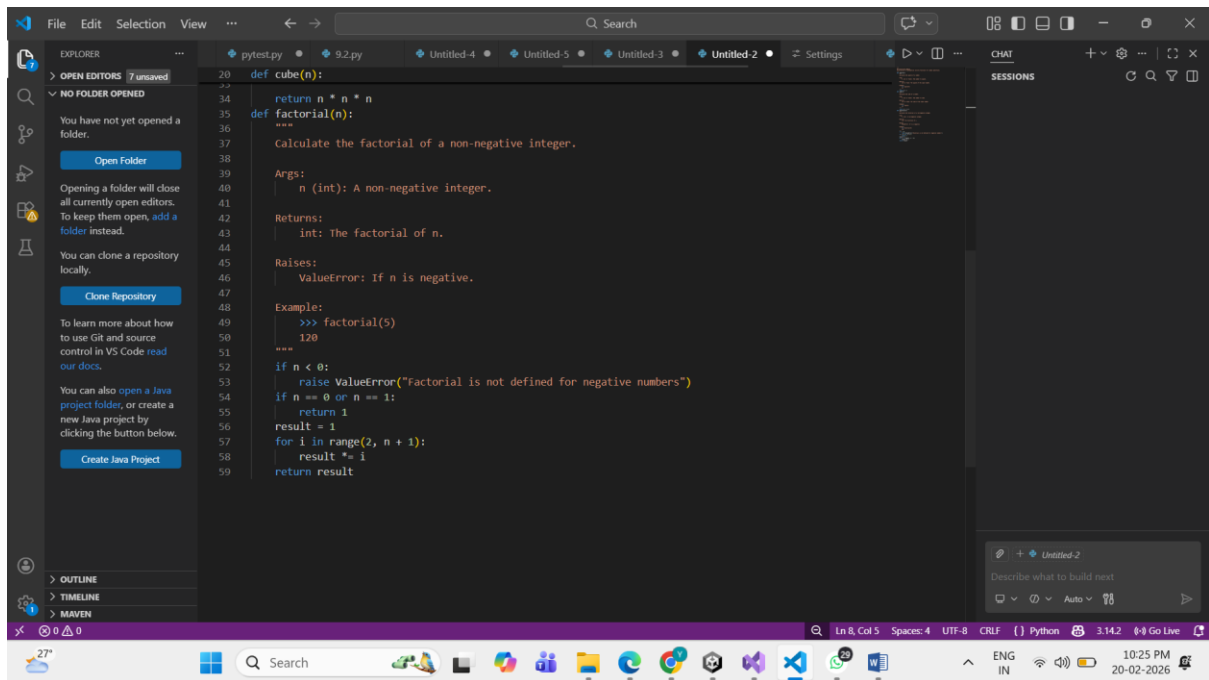
### Task:

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:



```
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
21 def cube(n):
22     """
23     Calculate the cube of a number.
24
25     Args:
26         n (int or float): The number to cube.
27
28     Returns:
29         int or float: The cube of the input number.
30
31     Example:
32     >>> cube(3)
33     27
34     """
35     return n * n * n
36
37 def factorial(n):
38     """
39     Calculate the factorial of a non-negative integer.
40
41     Args:
42         n (int): A non-negative integer.
43
44     Returns:
45         int: The factorial of the input number.
46
47     Example:
48     >>> factorial(5)
49     120
50     """
51     # Implementation of factorial function
52     result = 1
53     for i in range(1, n + 1):
54         result *= i
55     return result
```



## 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 editor interface. The Explorer panel on the left shows a project with files 'pytest.py' and '9.2.py'. The main editor window displays the code for the 'mark\_present' function. The function is a docstring that describes its purpose, arguments, and returns. It includes an example of how to use the function. The status bar at the bottom indicates the current line and column (Ln 65, Col 47) and the file encoding (UTF-8).

```
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
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, [])
```

The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left shows a project with files 'pytest.py' and '9.2.py'. The main editor window displays the code for the 'get\_attendance' function. The function is a docstring that describes its purpose, arguments, and returns. It includes an example of how to use the function. The status bar at the bottom indicates the current line and column (Ln 65, Col 47) and the file encoding (UTF-8).

```
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):
```

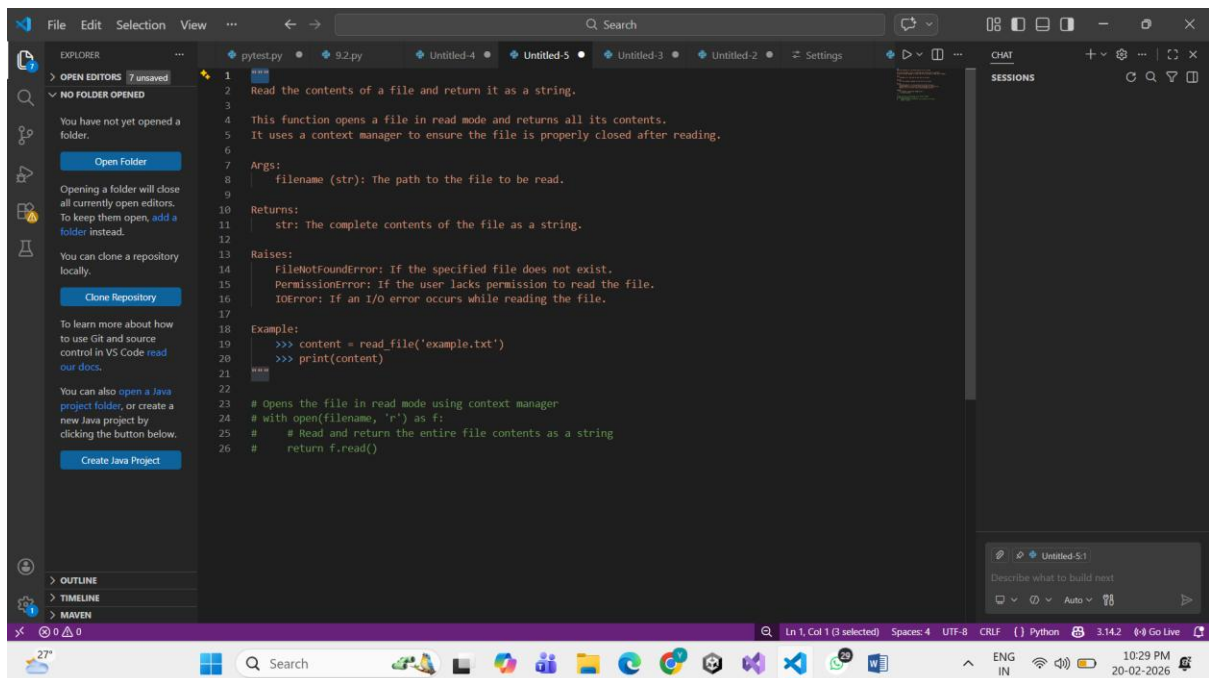
```
    with open(filename, 'r') as f:
```

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

Task:

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

## CODE:



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