

AI Assisted Coding

ASSIGNMENT 9.5

Name: Niharshith

HT No: 2303A52044

Batch: 31

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) Docstring

```
1 # 1. Docstring Type Documentation
2 def reverse_string(text):
3     """
4         Reverses the given string.
5         Parameters:
6             text: The string to be reversed.
7         Returns:
8             The reversed string.
9         """
10        return text[::-1]
11 print(reverse_string.__doc__)
```

```
Reverses the given string.
Parameters:
text (str): The string to be reversed.
Returns:
str: The reversed string.
Example:
>>> reverse_string("hello")
'olleh'
```

(b) Inline comments

```
1 # Inline Type Documentation
2 def reverse_string(text):
3     return text[::-1] # Reverses the given string
4 #-----
5 # Reverse the string "Hello, World!" and print the result
6 #-----
```

(c) Google-style documentation

```

1  # Google Style Documentation
2  def reverse_string(text: str) -> str:
3      """Reverses the given string.
4
5          Args:
6              text (str): The string to be reversed.
7
8          Returns:
9              str: The reversed string.
10
11         """
12
13     return text[::-1]
14 input_text = input("Enter a string to reverse: ")
15 if not isinstance(input_text, str):
16     raise ValueError("Input must be a string.")
17 print(reverse_string(input_text))
18 print(reverse_string.__doc__)

```

Enter a string to reverse: hello
olleh
Reverses the given string.

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

Returns:
str: The reversed string.

2. Compare the three documentation styles.

| Aspect | Docstring | Inline Comments | Google-Style |
|-----------|-----------|-----------------|--------------|
| Scope | Function | Line | Function |
| Structure | Basic | None | Structured |

| | | | |
|------------------|----------|--------|-----------|
| Readability | Good | Simple | Clear |
| Detail Level | Medium | Low | High |
| Tool Support | Yes | No | Yes |
| Professional Use | Moderate | Low | High |
| Best For | Scripts | Logic | Libraries |

3. Recommend the most suitable style for a utility-based string library.

Google-style documentation is most suitable for a utility-based string library because it reflects professional coding standards and industry practices.

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.

```
1  # Inline Comment  
2  # This function checks if the password is strong enough by ensuring it has at least 8 characters.  
3  def check_strength(password: str) -> bool:  
4      # Doctype Comment  
5      """  
6      Parameters:  
7          | password (str): The password to be checked for strength.  
8      Returns:  
9          | bool: True if the password is strong (at least 8 characters), False otherwise.  
10     """  
11    # Google Docstring Comment  
12    """  
13    A strong password is defined as one that has at least 8 characters.  
14    This function checks the length of the password and returns True  
15    if it meets the requirement, otherwise it returns False.  
16    Args:  
17        | password (str): The password to be evaluated for strength.  
18    Returns:  
19        | bool: True if the password is strong, False if it is weak.  
20    Example:  
21    >>> check_strength("password123")  
22    True  
23    """  
24    return len(password) >= 8 # This is the minimum length requirement for a strong password.  
25  print(check_strength.__doc__)  
26  print(check_strength("password123")) # This should return True since the password is strong.
```

Parameters:
password (str): The password to be checked for strength.
Returns:
bool: True if the password is strong (at least 8 characters), False otherwise.

True

2. Compare documentation styles for security-related code.

| Aspect | Docstring | Inline Comments | Google-Style |
|----------------------|------------|-----------------|--------------|
| Clarity | Good | Basic | Very Clear |
| Structure | Simple | None | Structured |
| Security Explanation | Limited | Minimal | Detailed |
| Professional Use | Moderate | Low | High |
| Maintainability | Medium | Low | High |
| Industry Preference | Acceptable | Rare | Strong |

3. Recommend the most appropriate style.

For security-related code like password validation, Google-style documentation is most appropriate because it ensures clarity, professionalism, and maintainability.

Problem 3: Math Utilities Module

Task:

1. Create a module math_utils.py with functions:

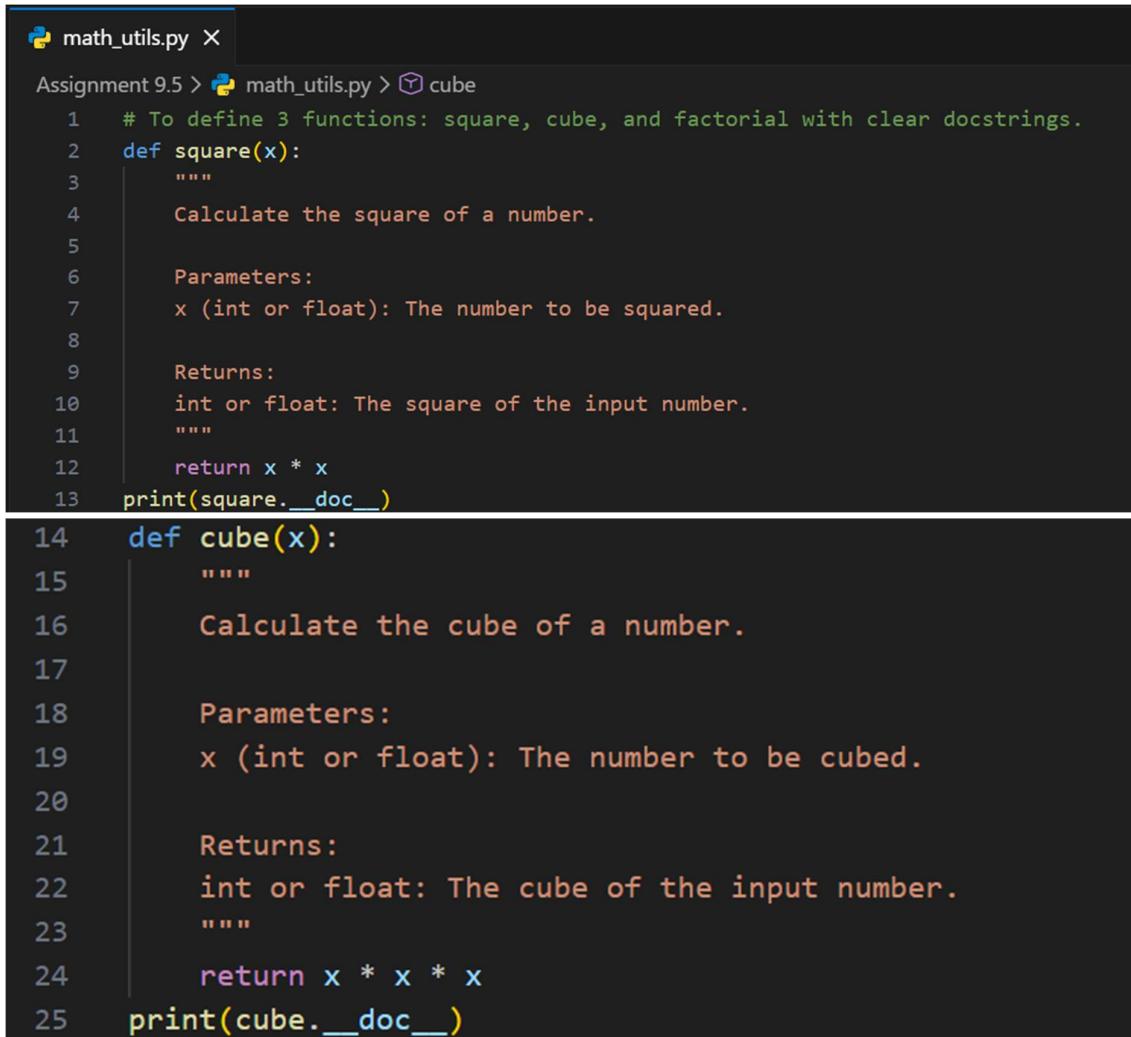
- o square(n)

- o cube(n)

- o factorial(n)

2. Generate docstrings automatically using AI tools.

3. Export documentation as an HTML file.



```
math_utils.py X
Assignment 9.5 > math_utils.py > cube
1  # To define 3 functions: square, cube, and factorial with clear docstrings.
2  def square(x):
3      """
4          Calculate the square of a number.
5
6      Parameters:
7          x (int or float): The number to be squared.
8
9      Returns:
10         int or float: The square of the input number.
11         """
12         return x * x
13     print(square.__doc__)
14
15  def cube(x):
16      """
17          Calculate the cube of a number.
18
19      Parameters:
20          x (int or float): The number to be cubed.
21
22      Returns:
23          int or float: The cube of the input number.
24          """
25         return x * x * x
26     print(cube.__doc__)
```

```

26 def factorial(n):
27     """
28     Calculate the factorial of a non-negative integer.
29     Parameters:
30     n (int): The non-negative integer for which to calculate the factorial.
31     Returns:
32     int: The factorial of the input number.
33     Raises:
34     ValueError: If n is negative.
35     """
36     if n < 0:
37         raise ValueError("Factorial is not defined for negative numbers.")
38     elif n == 0 or n == 1:
39         return 1
40     else:
41         result = 1
42         for i in range(2, n + 1):
43             result *= i
44     return result
45 print(factorial.__doc__)

```

3. Export documentation as an HTML file.

● PS Z:\AIAC\Assignment 9.5> `python -m pydoc -w .\math_utils.py`

Calculate the square of a number.

Parameters:

x (int or float): The number to be squared.

Returns:

int or float: The square of the input number.

Calculate the cube of a number.

Parameters:

x (int or float): The number to be cubed.

Returns:

int or float: The cube of the input number.

Calculate the factorial of a non-negative integer.

Parameters:

n (int): The non-negative integer for which to calculate the factorial.

Returns:

int: The factorial of the input number.

Raises:

ValueError: If n is negative.

wrote `math_utils.html`

The screenshot shows a browser window with the title "Pydoc: module math_utils". The address bar indicates the URL is "localhost:8080/math_utils.html". The page content is the generated documentation for the "math_utils" module, specifically the "Functions" section. It includes three functions: `cube(x)`, `factorial(n)`, and `square(x)`. Each function has a detailed docstring describing its purpose, parameters, returns, and any exceptions it might raise.

```
# To define 3 functions: square, cube, and factorial with clear docstrings.

Functions
cube(x)
    Calculate the cube of a number.

    Parameters:
        x (int or float): The number to be cubed.

    Returns:
        int or float: The cube of the input number.

factorial(n)
    Calculate the factorial of a non-negative integer.

    Parameters:
        n (int): The non-negative integer for which to calculate the factorial.

    Returns:
        int: The factorial of the input number.

    Raises:
        ValueError: If n is negative.

square(x)
    Calculate the square of a number.

    Parameters:
        x (int or float): The number to be squared.

    Returns:
        int or float: The square of the input number.
```

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

```
attendance.py ●
Assignment 9.5 > attendance.py > ...
1  # To Create 3 Functions mark_present(student), mark_absent(student) and get_attendance() with
2  # Proper Docstrings
3  def mark_present(student: str) -> None:
4      """
5          Marks a student as present.
6
7          Args:
8              student (str): The name of the student to mark as present.
9              """
10         print(f"{student} is marked as present.")
11
12 def mark_absent(student: str) -> None:
13     """
14         Marks a student as absent.
15
16         Args:
17             student (str): The name of the student to mark as absent.
18             """
19         print(f"{student} is marked as absent.")
20
21 def get_attendance(student: str) -> None:
22     """
23         Gets the attendance status of a student.
24
25         Args:
26             student (str): The name of the student whose attendance is to be retrieved.
27             """
28         print(f"Attendance status for {student} is being retrieved.")
29         print(mark_present.__doc__, mark_absent.__doc__, get_attendance.__doc__)
```

```
○ PS Z:\AIAC\Assignment 9.5> python -m pydoc -p 8080
Server ready at http://localhost:8080/
Server commands: [b]rowser, [q]uit
server> b
server>
Marks a student as present.

Args:
    student (str): The name of the student to mark as present.

Marks a student as absent.

Args:
    student (str): The name of the student to mark as absent.

Gets the attendance status of a student.

Args:
    student (str): The name of the student whose attendance is to be retrieved.
```

```

Marks a student as present.

Args:
    student (str): The name of the student to mark as present.

Marks a student as absent.

Args:
    student (str): The name of the student to mark as absent.

Gets the attendance status of a student.

Args:
    student (str): The name of the student whose attendance is to be retrieved.

```

Pydoc: module attendance

localhost:8080/attendance.html

Python 3.13.5 [tags/v3.13.5:6cb20a2, MSC v.1943 64 bit (AMD64)]

Module Index : Topics : Keywords

Get Search

attendance

To Create 3 Functions `mark_present(student)`, `mark_absent(student)` and `get_attendance()` with
Proper Docstrings

Functions

```

get_attendance(student: str) -> None
    Gets the attendance status of a student.

    Args:
        student (str): The name of the student whose attendance is to be retrieved.

mark_absent(student: str) -> None
    Marks a student as absent.

    Args:
        student (str): The name of the student to mark as absent.

mark_present(student: str) -> None
    Marks a student as present.

    Args:
        student (str): The name of the student to mark as present.

```

index
z:\aiac\assignment 9.5\attendance.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.

```
File Handling.py X
Assignment 9.5 > File Handling.py > ...
1  # In Line Documentation for File Handling
2  def read_file_inline(filename):
3      with open(filename, 'r') as f: # Open the file in read mode
4          return f.read() # Return the contents of the file as a string
5
6  # Docstring Documentation for File Handling
7  def read_file_docstring(filename):
8      """
9          Reads the contents of a file and returns it as a string.
10
11         Parameters:
12             filename (str): The name of the file to be read.
13
14         Returns:
15             str: The contents of the file.
16
17         Raises:
18             FileNotFoundError: If the file does not exist.
19             IOError: If there is an error reading the file.
20         """
21         with open(filename, 'r') as f:
22             return f.read()
23
24  # Google Style Docstring Documentation for File Handling
25  def read_file_google_style(filename: str) -> str:
26      """
27          Reads the contents of a file and returns it as a string.
28
29         Args:
30             filename (str): The name of the file to be read.
31         Returns:
32             str: The contents of the file.
33         Raises:
34             FileNotFoundError: If the file does not exist.
35             IOError: If there is an error reading the file.
36         """
37         with open(filename, 'r') as f:
38             return f.read()
39
40
41  print(read_file_inline.__doc__)
42  print(read_file_docstring.__doc__)
43  print(read_file_google_style.__doc__)
```

None

Reads the contents of a file and returns it as a string.

Parameters:

filename (str): The name of the file to be read.

Returns:

str: The contents of the file.

Raises:

FileNotFoundException: If the file does not exist.

IOError: If there is an error reading the file.

Reads the contents of a file and returns it as a string.

Args:

filename (str): The name of the file to be read.

Returns:

str: The contents of the file.

Raises:

FileNotFoundException: If the file does not exist.

IOError: If there is an error reading the file.

Identify which style best explains exception handling, Justify your recommendation.

Google Style Docstring

It clearly separates exceptions under a dedicated "Raises" section, making error handling easy to understand and professionally structured.