

Assignment - 9.1

A.Nikhil

2303A51934 B - 27

Lab Experiment: Automatic Documentation Generation & Code Comments

Problem 1:

Consider the following Python function:

```
def find_max(numbers): return  
max(numbers)
```

- Task:
- Write documentation for the function in all three formats:
 - (a) Docstring
 - (b) Inline comments
 - (c) Google-style documentation
 - Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.
 - Recommend which documentation style is most effective for a mathematical utilities library and justify your answer.

Problem 1: Documentation Styles

Given Function : `def`
`find_max(numbers):`
`return max(numbers)` **(a)**

Docstring Documentation:

```
def find_max(numbers):  
    """  
    Returns the maximum value from a list of numbers.
```

```

Parameters:
    numbers (list): A list of numeric values.

Returns:
    int or float: The maximum value in the list.
"""
return max(numbers)

```

(b) Inline Comments :

```

def find_max(numbers):
    # Use built-in max() function to find the largest
    number
    return max(numbers)

```

(c) Google-Style Documentation :

```

def find_max(numbers):
    """
    Finds the maximum value in a list.

```

```

Args:
    numbers (list): List of numeric values.

```

```

Returns:
    int or float: Maximum value from the list.
"""
return max(numbers)

```

Critical Comparison :

Style	Advantages	Disadvantages	Use Case
Docstring	Standard, works with pydoc	Less structured	Small functions

Inline Comments	Simple, easy to read	No auto-doc support	Explaining logic
Google Style	Structured, readable, professional	Slightly verbose	Libraries & APIs

Problem 2: Consider the following Python function:

```
def login(user, password, credentials): return
credentials.get(user) == password
```

- Task:
1. Write documentation in all three formats.
 2. Critically compare the approaches.
 3. Recommend which style would be most helpful for new developers onboarding a project, and justify your choice.

Problem 2: Login Function Documentation

Given Function :

```
def login(user, password, credentials):
return credentials.get(user) == password (a)
```

Docstring

```
def login(user, password, credentials):
    """
    Checks whether the provided user credentials are
    valid.
    """
    return credentials.get(user) == password
```

(b) Inline Comments

```
def login(user, password, credentials):
```

```
# Compare stored password with input password
return credentials.get(user) == password
```

(c) Google-Style Documentation

```
def login(user, password, credentials):
    """
    Authenticates a user using provided credentials.

    Args:
        user (str): Username.
        password (str): Password entered by the user.
        credentials (dict): Dictionary of user-
        password pairs.

    Returns:
        bool: True if login is successful, False
        otherwise.
    """
    return credentials.get(user) == password
```

Comparison

- Inline comments: minimal help
- Simple docstring: lacks details
- Google-style: clear, onboarding-friendly

Problem 3: Calculator (Automatic Documentation Generation)

Task: Design a Python module named `calculator.py` and demonstrate automatic documentation generation.

Instructions:

1. Create a Python module `calculator.py` that includes the following functions, each written with appropriate docstrings:
 - o `add(a, b)` – returns the sum of two numbers
 - o `subtract(a, b)` – returns the difference of two numbers
 - o `multiply(a, b)` – returns the product of two numbers
 - o `divide(a, b)` – returns the quotient of two numbers
2. Display the module documentation in the terminal using Python's documentation tools.
3. Generate and export the module documentation in HTML format using the `pydoc` utility, and open the generated HTML file in a web browser to verify the output.

Problem 3: Calculator Module

calculator.py:

```
"""
Calculator Module
Provides basic arithmetic operations.
"""

def add(a, b):
    """Returns the sum of two numbers."""
    return a + b

    def subtract(a,
b):
    """Returns the difference of two numbers."""
    return a - b

    def multiply(a,
b):
    """Returns the product of two numbers."""
    return a * b

    def divide(a,
b):
```

```

    """Returns the quotient of two numbers."""
    if b == 0:
        return "Division by zero
error"
    return a / b

```

Problem 4: Conversion Utilities Module Task:

1. Write a module named `conversion.py` with functions:

o `decimal_to_binary(n)` o

`binary_to_decimal(b)` o

`decimal_to_hexadecimal(n)`

2. Use Copilot for auto-generating docstrings.

3. Generate documentation in the terminal.

4. Export the documentation in HTML format and open it in a browser.

Problem 4: Conversion Utilities Module

`conversion.py`

```

"""
Conversion Utilities Module
"""
def
decimal_to_binary(n):
    """Converts a decimal number to binary."""
    return bin(n)[2:]
    def
binary_to_decimal(b):
    """Converts a binary number to decimal."""
    return int(b, 2)
    def
decimal_to_hexadecimal(n):

```

```
    """Converts a decimal number to hexadecimal."""  
    return hex(n)[2:]
```

Problem 5 – Course Management Module Task:

1. Create a module `course.py` with functions:
 - o `add_course(course_id, name, credits)`
 - o `remove_course(course_id)`
 - o `get_course(course_id)`
2. Add docstrings with Copilot.
3. Generate documentation in the terminal.
4. Export the documentation in HTML format and open it in a browser.

Problem 5: Course Management Module

`course.py`

```
"""  
Course Management Module  
"""  
courses = {}  
def  
add_course(course_id, name, credits):  
    """Adds a new course to the system."""  
    courses[course_id] = {  
        "name":  
name,  
        "credits": credits  
    }  
def  
remove_course(course_id):  
    """Removes a course using course ID."""  
    courses.pop(course_id, None)
```

```
def
get_course(course_id):
    """Returns details of a course."""
return courses.get(course_id)
```