ASSIGNMENT-9.1

HT NO:2303A51585

Batch No:2303A51585

Course:AI Assisted Coding

Problem 1:

Consider the following Python function:

def find_max(numbers):

return max(numbers)

Prompt:

Generate documentation for the given Python function using all three formats:

1. Python docstring

2. Inline comments

3. Google-style documentation

After writing the documentation, critically compare the three approaches by discussing their advantages, disadvantages, and suitable use cases.

Finally, recommend the most effective documentation style for a mathematical utilities library and justify the choice.

Given Python Code:

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

Generate Python Docstring:

```
def find_max(numbers):
    """
    Finds the maximum number in a list of numbers.

    Args:
        numbers (list or iterable): A collection of numbers.

    Returns:
        int or float: The largest number from the input list.
    """
    return max(numbers)
```

Generate Inline Comments:

```
def find_max(numbers):
    """
    Finds the maximum number in a list of numbers.

    Args:
        numbers (list or iterable): A collection of numbers.

    Returns:
        int or float: The largest number from the input list.
    """
    return max(numbers)  # Use the built-in max() function to find the largest number
```

Generate Google-style Documentation:

```
def find_max(numbers):
    """Finds the maximum number in a list of numbers.

    Args:
        numbers (list or iterable): A collection of numbers to search through.

    Returns:
        int or float: The largest number found in the input collection.
    """
    return max(numbers)  # Use the built-in max() function to find the largest number
```

Justification:

1. Google-style docstrings provide clear and well-structured documentation for mathematical functions.

2. They clearly define inputs, outputs, and exceptions using *Args*, *Returns*, and *Raises* sections.

3. This format improves readability, especially for functions involving formulas and edge cases.

4. Developers can quickly understand function usage without reading the implementation.

5. Google-style docstrings are easier to write and maintain than reStructuredText.

6. They support automated API documentation using tools like Sphinx with Napoleon.

7. Consistency across functions improves collaboration in team projects.

8. Inline comments alone cannot describe complete function behavior effectively.

9. Google-style docstrings balance professionalism with simplicity.

10. Hence, they are ideal for a mathematical utilities library.

Problem 2:

Consider the following Python function:

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

return credentials.get(user) == password
```

Prompt:

Generate documentation for the given Python login function using:

1. Python docstring

2. Inline comments

3. Google-style documentation

Then compare the three documentation approaches and recommend which style is most helpful for new developers onboarding a project, with proper justification.

Given Python Code:

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

Generate Python Docstring:

```
def login(user, password, credentials):
    """Authenticates a user against a provided set of credentials.

    Args:
        user (str): The username attempting to log in.
        password (str): The password provided by the user.
        credentials (dict): A dictionary where keys are usernames (str)
                            and values are their corresponding passwords (str).

    Returns:
        bool: True if the user's password matches the stored credentials,
              False otherwise.
    """
    return credentials.get(user) == password
```

Generate Inline Comments:

```
def login(user, password, credentials):
    """Authenticates a user against a provided set of credentials.

    Args:
        user (str): The username attempting to log in.
        password (str): The password provided by the user.
        credentials (dict): A dictionary where keys are usernames (str)
                            and values are their corresponding passwords (str).

    Returns:
        bool: True if the user's password matches the stored credentials,
              False otherwise.
    """
    return credentials.get(user) == password  # Compare provided password with stored password using .get() to handle non-existent users gracefully.
```

Generate Google-style Documentation:

```python
def login(user, password, credentials):
    """Authenticates a user against a provided set of credentials.

    Args:
        user (str): The username attempting to log in.
        password (str): The password provided by the user.
        credentials (dict): A dictionary where keys are usernames (str)
            and values are their corresponding passwords (str).

    Returns:
        bool: True if the user's password matches the stored credentials,
            False otherwise.
    """
    return credentials.get(user) == password  # Compare provided password with stored password using .get() to handle non-existent users gracefully.
```

Justification:

1. Google-style docstrings are easy to read and understand due to their simple, structured format.

2. Clear sections like *Args*, *Returns*, and *Raises* help new developers quickly grasp function usage.

3. The learning curve is low compared to reStructuredText-based Python docstrings.

4. Consistent formatting across the codebase improves navigation and predictability.

5. New developers can understand APIs without deeply reading implementation details.

6. Google-style docstrings balance conciseness with sufficient detail.

7. They reduce confusion during onboarding and speed up developer productivity.

8. Automated documentation can be generated using Sphinx with the Napoleon extension.

9. Inline comments alone cannot provide a complete overview of function behavior.

10. Therefore, Google-style docstrings are ideal for onboarding new developers effectively.

Problem 3:

Calculator (Automatic Documentation Generation)

Task: Design a Python module named calculator.py and

demonstrate automatic documentation generation.

Prompt:

Design a Python module named calculator.py.

Tasks:

1. Create functions add(a, b), subtract(a, b), multiply(a, b), and divide(a, b).

2. Add appropriate Python docstrings to each function for automatic documentation generation.

3. Show how to display the module documentation in the terminal using Python documentation tools.

4. Generate and export the module documentation in HTML format using the pydoc utility and verify the output in a web browser.

Python Code:

```
"""
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
```

Python Code Output:

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python -m pydoc calculator
>>
Help on module calculator:

NAME
    calculator

NAME
    calculator

DESCRIPTION
    Calculator Module
    Provides basic arithmetic operations.

FUNCTIONS
    Calculator Module
    Provides basic arithmetic operations.

FUNCTIONS
    Provides basic arithmetic operations.

FUNCTIONS
FUNCTIONS
    add(a, b)
        Returns the sum of two numbers.
-- More  --
```

Creating HTML File from Python file

```
PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_C
oding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python -m pydoc -w calculator
>>
wrote calculator.html
```

HTML Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Python: module calculator</title>
</head><body>

<table class="heading">
<tr class="heading-text decor">
<td class="title"> <br><strong class="title">calculator</strong></td>
<td class="extra"><a href=".">index</a><br><a href="file:c%3A%5Cusers%5Cdell%5Conedrive%5Cdesktop%5Clearning%20courses%5Cai_assisant_coding%5Ccalculator.py">c:\users\de
    <p><span class="code">Calculator Module<br>
Provides basic arithmetic operations.</span></p>
<p>
<table class="section">
<tr class="decor functions-decor heading-text">
<td class="section-title" colspan=3> <br><strong class="bigsection">Functions</strong></td></tr>

<tr><td class="decor functions-decor"><span class="code">      </span></td><td> </td>
<td class="singlecolumn"><dl><dt><a name="-add"><strong>add</strong></a>(a, b)</dt><dd><span class="code">Returns the sum of two numbers.</spar
 <dl><dt><a name="-divide"><strong>divide</strong></a>(a, b)</dt><dd><span class="code">Returns the quotient of two numbers.</span></dd></dl>
 <dl><dt><a name="-multiply"><strong>multiply</strong></a>(a, b)</dt><dd><span class="code">Returns the product of two numbers.</span></dd></dl
 <dl><dt><a name="-subtract"><strong>subtract</strong></a>(a, b)</dt><dd><span class="code">Returns the difference of two numbers.</span></dd><
</td></tr></table>
</body></html>
```

HTML Code Output:

---

index
calculator c:\users\dell\onedrive\desktop\learning courses\ai_assisant_coding\calculator.py

Calculator Module
Provides basic arithmetic operations.

Functions

add(a, b)
Returns the sum of two numbers.

divide(a, b)
Returns the quotient of two numbers.

multiply(a, b)
Returns the product of two numbers.

subtract(a, b)
Returns the difference of two numbers.

Justification:

1. Task demonstrates automatic documentation generation using Python docstrings.
2. Docstrings help explain the purpose and functionality of each calculator function.
3. Using pydoc reduces the need for manual documentation writing.
4. Terminal-based documentation allows quick access during development.
5. HTML documentation provides a clear and user-friendly presentation.
6. Automatic documentation ensures consistency across the module.
7. Well-documented code improves readability and maintainability.
8. It helps new developers understand the module easily.
9. The approach follows standard Python documentation practices.
10. This task highlights the importance of documentation in professional software development.

Problem 4: Conversion Utilities Module

Task:

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

→ decimal_to_binary(n)

→binary_to_decimal(b)

→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.

Prompt:

Create a Python module named conversion.py with number conversion

functions.

Use an AI tool to auto-generate clear Python docstrings for each function.

Generate the module documentation in the terminal and export the

documentation in HTML format using pydoc.

Python Code:

```python
"""
Conversion Utilities Module
"""

def decimal_to_binary(n):

    """Converts a decimal number to binary."""
    """Converts a decimal number to its binary representation as a string."""
    return bin(n)[2:]

def binary_to_decimal(b):
    """Converts a binary number to decimal."""
    """Converts a binary number (given as a string) to its decimal representation as an integer."""
    return int(b, 2)

def decimal_to_hexadecimal(n):
    """Converts a decimal number to hexadecimal."""
    """Converts a decimal number to its hexadecimal representation as a string."""
    return hex(n)[2:]
def hexadecimal_to_decimal(h):
    """Converts a hexadecimal number to decimal."""
    """Converts a hexadecimal number (given as a string) to its decimal representation as an integer."""
    return int(h, 16)
```

Python Code Output:

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python -m pydoc conversion
Help on module conversion:

NAME
    conversion

FILE
    c:\users\dell\onedrive\desktop\learning courses\ai_assisant_coding\conversion.py


(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

Creating HTML File from Python file

```
PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python -m pydoc -w conversion
>>
wrote conversion.html
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

HTML Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Python: module conversion</title>
</head><body>

<table class="heading">
<tr class="heading-text decor">
<td class="title"> <br><strong class="title">conversion</strong></td>
<td class="extra"><a href=".">index</a><br><a href="file:c%3A%5Cusers%5Cdell%5Conedrive%5Cdesktop%5Clearning%20courses%5Cai_assisant_coding%5Cconversion.py">c:\users\dell
</a></td></tr></table>
    <p></p>

</body></html>
```

HTML Code Output:

conversion index
c:\users\dell\onedrive\desktop\learning courses\ai_assisant_coding\conversion.py

Justification:

1.Task demonstrates the use of automatic documentation generation for utility functions.

2. Using Copilot helps in generating accurate and consistent docstrings quickly.

3. Docstrings clearly explain the purpose and functionality of each conversion function.

4. Automatic documentation reduces manual documentation effort.

5.Terminal-based documentation allows quick reference during development.

6. HTML documentation provides a clear and readable presentation of the module.

7. Well-documented code improves readability and maintainability.

8. It helps new developers understand the conversion logic easily.

9. Using pydoc follows standard Python documentation practices.

10. This task highlights the importance of documentation in real-world software development.

Problem 5 – Course Management Module

Task:

1. Create a module course.py with functions:

→add_course(course_id, name, credits)

→ remove_course(course_id)

→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.

Prompt:

Create a Python module named course.py for managing course information.

Add meaningful Python docstrings using an AI tool.

Generate the module documentation in the terminal and export it in HTML

format using the pydoc utility.

Python Code:

```python
"""
Course Management Module
"""

courses = {}

def add_course(course_id, name, credits):
    """Adds a new course to the course list."""
    #docstring suggestion: "Adds a new course to the course list with the given course ID, name, and credits."
    courses[course_id] = {
        "name": name,
        "credits": credits
    }

def remove_course(course_id):
    """Removes a course using course ID."""
    #docstring suggestion: "Removes a course from the course list using the given course ID."
    courses.pop(course_id, None)

def get_course(course_id):
    """Retrieves details of a course."""
    #docstring suggestion: "Retrieves details of a course using the given course ID. Returns None if the course does not exist."
    return courses.get(course_id)
```

Python Code Output:

```
PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python -m pydoc co rse
Help on module course:

NAME
    course

FILE
    c:\users\dell\onedrive\desktop\learning courses\ai_assisant_coding\course.py


(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

Creating HTML File from python file

```
PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING CO
s1"
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python -m pydoc -w co rse
>>
wrote course.html
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

HTML Code:

course.html > html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Python: module course</title>
</head><body>

<table class="heading">
<tr class="heading-text decor">
<td class="title"> <br><strong class="title">course</strong></td>
<td class="extra"><a href=".">index</a><br><a href="file:c%3A%5Cusers%5Cdell%5Conedrive%5Cdesktop%5Clearning%20courses%5Cai_assisant_coding%5Ccourse.py">c:\users\dell\one
    <p></p>

</body></html>
```

HTML Code Output:

index

course c:\users\dell\onedrive\desktop\learning courses\ai_assisant_coding\course.py

Justification:

1. Task demonstrates automatic documentation generation for a real-world module.
2. Using Copilot helps generate consistent and accurate docstrings efficiently.
3. Docstrings clearly explain the purpose of each course management function.
4. Automatic documentation reduces manual documentation effort.
5. Terminal documentation provides quick reference for developers.
6. HTML documentation offers a structured and readable view of the module.
7. Well-documented code improves readability and maintainability.
8. It helps new developers understand the course workflow easily.
9. Using pydoc follows standard Python documentation practices.
10. This task highlights the importance of documentation in software development.