

WEEK-9.1

2303A51804

Batch-28

TASK-1:

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.

Prompt:

1. "Write documentation for the function using:"

- "(a) A Python docstring"
- "(b) Inline comments"
- "(c) Google-style documentation"


Code:

(a) Docstring

```
def find_max(numbers):  
    """  
    Finds the maximum number in a list of numbers.  
  
    Args:  
        numbers (list): A list of numerical values.  
  
    Returns:  
        int or float: The maximum number in the list.  
  
    Examples:  
    >>> find_max([1, 2, 3, 4, 5])  
    5  
    >>> find_max([-1, -5, -2])  
    -1  
    >>> find_max([10.5, 2.3, 7.8])  
    10.5  
    """  
    return max(numbers)
```

(b) Inline comments:


```
def find_max(numbers):  
    # This function takes a list of numbers as input.  
    # It finds and returns the maximum number from the list.  
    return max(numbers) # The built-in max() function is used to get the largest number.
```

 Add clear and concise inline comments to the following Python function to explain what the code does. Do not use a docstring — only inline comments.

(c) Google-style documentation

```
def find_max(numbers):  
    """Finds the maximum number in a list of numbers.  
  
    Args:  
        numbers (list): A list of numerical values.  
  
    Returns:  
        int or float: The maximum number found in the list.  
    """  
    return max(numbers)
```

Start coding or [generate](#) with AI.

 Write Google-style Python documentation for the following function. The documentation should include a brief description, Args, and Returns sections.

Explanation:

Comparison of Documentation Approaches:

Docstrings provide standard Python documentation and are accessible via `help()` and IDEs, but can lack structure without a defined style.

Inline comments are easy to write and useful for explaining code logic, but they do not describe a function's purpose or interface and cannot be used by documentation tools.

Google-style documentation is well-structured, readable, and scalable, though slightly more verbose.

Recommendation:

Google-style documentation is the most effective for a mathematical utilities library because it clearly defines inputs and outputs, supports auto-generated documentation, and maintains consistency as the library grows

TASK-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.

Prompt:

python

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

Perform the following tasks:

1. "Write documentation for the function using:"
 - "(a) A Python docstring"
 - "(b) Inline comments"
 - "(c) Google-style documentation"

1. Write documentation in all three formats

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

```
def login(user, password, credentials):  
    """Authenticates a user based on provided credentials.  
  
    Args:  
        user (str): The username.  
        password (str): The password.  
        credentials (dict): A dictionary where keys are usernames and values  
        are passwords.  
  
    Returns:  
        bool: True if the username and password match the credentials, False  
        otherwise.  
    """  
    return credentials.get(user) == password
```

Gemini

The code cell `zFR0XoM-nq6G` has been successfully executed, and the `find_max` function is now defined with Google-style documentation.

m Write a proper Python docstring for the following function.

Here's a proper Python docstring for your `login` function:

```
def login(user, password, credentials):  
    # This function attempts to authenticate a user.  
    # It checks if the provided 'user' exists as a key in the 'credentials' dictionary.  
    # It then compares the 'password' provided by the user with the password stored in 'credentials'.  
    # If the user is found and the passwords match, it returns True; otherwise, it returns False.  
    return credentials.get(user) == password
```

Start coding or generate with AI.

`login` function in the environment. This means the function is now available for use.

m Add clear and concise inline comments to the following Python function to explain what each part of the code does. Do not use a docstring — only inline comments.

```
def login(user, password, credentials):  
    """Authenticates a user based on provided credentials.  
  
    Args:  
        user (str): The username attempting to log in.  
        password (str): The password provided by the user.  
        credentials (dict): A dictionary mapping usernames to their corresponding passwords.  
  
    Returns:  
        bool: True if the provided username and password match the credentials, False otherwise.  
    """  
    # This function attempts to authenticate a user.  
    # It checks if the provided 'user' exists as a key in the 'credentials' dictionary.  
    # It then compares the 'password' provided by the user with the password stored in 'credentials'.  
    # If the user is found and the passwords match, it returns True; otherwise, it returns False.  
    return credentials.get(user) == password
```

Write Google-style Python documentation for the following function.

2. Critically compare the approaches.

- Docstrings provide basic documentation and are accessible through IDEs and `help()`, but may lack consistency without a strict format.
- Inline comments help explain specific logic but fail to describe the function's purpose, inputs, and outputs.
- Google-style documentation is structured, readable, and scalable, making it ideal for team environments.

3. Recommend which style would be most helpful for new Developers onboarding a project, and justify your choice.

Google-Style Documentation

Explanation:

Google-style documentation is the most helpful for new developers because it clearly explains what the function does, its parameters, and return values in a standardised format. This reduces confusion, improves readability, and speeds up onboarding in collaborative projects.

TASK-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 `py doc` utility, and open the generated HTML file in a web browser to verify the output.

Prompt:

Prompt:

Design a Python module named `calculator.py` to demonstrate automatic documentation generation.

Code :

```
"""
calculator.py

A simple calculator module that 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."""
```

```

21
22 def divide(a, b):
23     """Returns the quotient of two numbers."""
24     if b == 0:
25         raise ValueError("Division by zero is not allowed")
26     return a / b
27

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\manis\OneDrive\Desktop\PYTHON TRA> python -m pydoc -w calculator
wrote calculator.html
PS C:\Users\manis\OneDrive\Desktop\PYTHON TRA>

```

```

<!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">&nbsp;<strong class="title">calculator</strong></td>
<td class="extra"><a href=".">index</a><br><a href="file:c:\%3A%5Cusers%5Cmanis%5Conedrive%5Cdesktop%5Cpython%20tra%5Ccalcu
    <p><span class="code">calculator.py<br>
&nbsp;<br>
A&nbsp;  simple&nbsp;  calculator&nbsp;  module&nbsp;  that&nbsp;  provides&nbsp;  basic&nbsp;  arithmetic&nbsp;  operations.</span></p>
<p>
<table class="section">
<tr class="decor functions-decor heading-text">
<td class="section-title" colspan=3>&nbsp;<br><strong class="bigsection">Functions</strong></td></tr>

```

```

2  <html lang="en">
6  </head><body>
11 <td class="extra"><a href=".">index</a><br><a href="file:c:\%3A%5Cusers%5Cmanis%5Conedrive%5Cdesktop%5Cpython%20tra%5Ccalcu
12 <p><span class="code">calculator.py<br>
13 &nbsp;<br>
14 A&nbsp;  simple&nbsp;  calculator&nbsp;  module&nbsp;  that&nbsp;  provides&nbsp;  basic&nbsp;  arithmetic&nbsp;  operations.</span></p>
15 <p>
16 <table class="section">
17 <tr class="decor functions-decor heading-text">
18 <td class="section-title" colspan=3>&nbsp;<br><strong class="bigsection">Functions</strong></td></tr>
19
20 <tr><td class="decor functions-decor"><span class="code">&nbsp;&nbsp;&&nbsp;&nbsp;&&nbsp;&nbsp;&&nbsp;&nbsp;&&</span></td><td>&nbsp;</td></tr>
21 <td class="singlecolumn"><dl><dt><a name="-add"><strong>add</strong></a><a>(a, b)</a></dt><dd><span class="code">Returns&nbsp;&nbsp;&nbsp;tl
22 <dl><dt><a name="-divide"><strong>divide</strong></a><a>(a, b)</a></dt><dd><span class="code">Returns&nbsp;&nbsp;&nbsp;the&nbsp;&nbsp;&nbsp;quotient&nbsp;&
23 <dl><dt><a name="-multiply"><strong>multiply</strong></a><a>(a, b)</a></dt><dd><span class="code">Returns&nbsp;&nbsp;&nbsp;the&nbsp;&nbsp;&nbsp;produc
24 <dl><dt><a name="-subtract"><strong>subtract</strong></a><a>(a, b)</a></dt><dd><span class="code">Returns&nbsp;&nbsp;&nbsp;the&nbsp;&nbsp;&nbsp;differ
25 </td></tr></table>
26 </body></html>

```

[index](#)

calculator <c:\users\manis\onedrive\desktop\python tra\calculator.py>

calculator.py

A simple calculator module that 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.

Explanation

When the command `dir` is used in the terminal, it lists all files present in the current working directory. If `calculator.py` does not appear in the output of `dir`, it means that the terminal is not currently pointing to the folder where the file is saved.

Although the folder `PYTHON TRA` exists, the Python file may be:

- Saved in a different location
- Saved with an incorrect file extension such as `calculator.py.txt`
- Opened in VS Code from a different directory than the one shown in the terminal

Because of this mismatch, commands like "`pydoc`" cannot find the module.

TASK-4:

Conversion Utilities Module

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.

Prompt:

Create a Python module named `conversion.py` that demonstrates automatic documentation generation.

```
conversion.py

A utility module for number system conversions.
"""

def decimal_to_binary(n):
    """
    Converts a decimal number to its binary representation.

    Args:
        n (int): A decimal number.

    Returns:
        str: Binary representation of the decimal number.
    """
    return bin(n)[2:]
```

```
version.py > ...
def binary_to_decimal(b):
    """
    Converts a binary number to its decimal representation.

    Args:
        b (str): A binary number as a string.

    Returns:
        int: Decimal representation of the binary number.
    """
    return int(b, 2)

def decimal_to_hexadecimal(n):
    """
    Converts a decimal number to its hexadecimal representation.

    Args:
```

```
conversion.py > ...
33 def decimal_to_hexadecimal(n):
34     """
35     Converts a decimal number to its hexadecimal representation.
36
37     Args:
38         n (int): A decimal number.
39
40     Returns:
41         str: Hexadecimal representation of the decimal number.
42     """
43     return hex(n)[2:]
44
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- PS C:\Users\manis\OneDrive\Desktop\PYTHON TRA> & 'c:\Users\manis\AppData\Local\Programs\Python\Python311\python.exe' -c 'import sys; sys.path.append('sktop\PYTHON TRA\conversion.py'); import conversion; print(decimal_to_hexadecimal(10))'
- PS C:\Users\manis\OneDrive\Desktop\PYTHON TRA> python -m pydoc conversion

Help on module conversion:

NAME

conversion - conversion.py

DESCRIPTION

A utility module for number system conversions.

FUNCTIONS

binary_to_decimal(b)

Converts a binary number to its decimal representation.

Args:

b (str): A binary number as a string.

Returns:

int: Decimal representation of the binary number.

decimal_to_binary(n)

Converts a decimal number to its binary representation.

```
PS C:\Users\manis\OneDrive\Desktop\PYTHON TRA> python -m pydoc -w conversion
wrote conversion.html
PS C:\Users\manis\OneDrive\Desktop\PYTHON TRA>
Indexing completed.
```

conversion.py

A utility module for number system conversions.

Functions

binary_to_decimal(b)

Converts a binary number to its decimal representation.

Args:

b (str): A binary number as a string.

Returns:

int: Decimal representation of the binary number.

decimal_to_binary(n)

Converts a decimal number to its binary representation.

Args:

n (int): A decimal number.

Returns:

str: Binary representation of the decimal number.

decimal_to_hexadecimal(n)

Converts a decimal number to its hexadecimal representation.

Args:

n (int): A decimal number.

Returns:

str: Hexadecimal representation of the decimal number.

Explanation:

- The conversion.py module contains number system conversion functions.
- Docstrings enable automatic documentation generation.
- pydoc is used to view documentation in the terminal.
- pydoc -w exports the documentation as an HTML file for browser viewing.

TASK-5:

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

Prompt:

Design a Python module (for example, `course_manager.py`) to demonstrate automatic documentation generation.

Code:

```
# Dictionary to store course details
_courses = {}

def add_course(course_id, name, credits):
    """
    Adds a course with the given details.

    Args:
        course_id (str): Unique identifier for the course.
        name (str): Name of the course.
        credits (int): Number of credits for the course.

    Returns:
        None
    """
    _courses[course_id] = {
        "name": name,
        "credits": credits
    }

def remove_course(course_id):
    """
    Removes a course using its course ID.
```

```

33     Args:
34         | course_id (str): Unique identifier of the course.
35
36     Returns:
37         | bool: True if the course was removed, False if not found.
38     """
39     return _courses.pop(course_id, None) is not None
40
41
42 def get_course(course_id):
43     """
44     Retrieves course details for a given course ID.
45
46     Args:
47         | course_id (str): Unique identifier of the course.
48
49     Returns:
50         | dict or None: Course details if found, otherwise None.
51     """
52     return _courses.get(course_id)
53

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\manis\OneDrive\Desktop\PYTHON TRA> python -m pydoc -w course
wrote course.html
PS C:\Users\manis\OneDrive\Desktop\PYTHON TRA> 

```

course.py

A simple course management module.

Functions

add_course(course_id, name, credits)

Adds a course with the given details.

Args:

course_id (str): Unique identifier for the course.

name (str): Name of the course.

credits (int): Number of credits for the course.

Returns:

None

get_course(course_id)

Retrieves course details for a given course ID.

Args:

course_id (str): Unique identifier of the course.

Returns:

dict or None: Course details if found, otherwise None.

remove_course(course_id)

Removes a course using its course ID.

Args:

course_id (str): Unique identifier of the course.

Returns:

bool: True if the course was removed, False if not found.

Explanation:

The Course Management module is used to manage course information such as course ID, name, and credits. Docstrings are added to each function to describe its purpose, parameters, and return values. These docstrings enable automatic documentation generation using Python's "pydoc" tool.

The documentation can be viewed directly in the terminal using Python documentation commands. Additionally, the same documentation can be exported into an HTML file, which can be opened in a web browser for easy viewing. This approach improves code readability and maintainability, and helps new developers understand the module quickly.