Assignment-9.1

Name: Arjun Manoj

Batch:44

H.No:2303A52134

Problem 1:

Consider the following Python function:

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

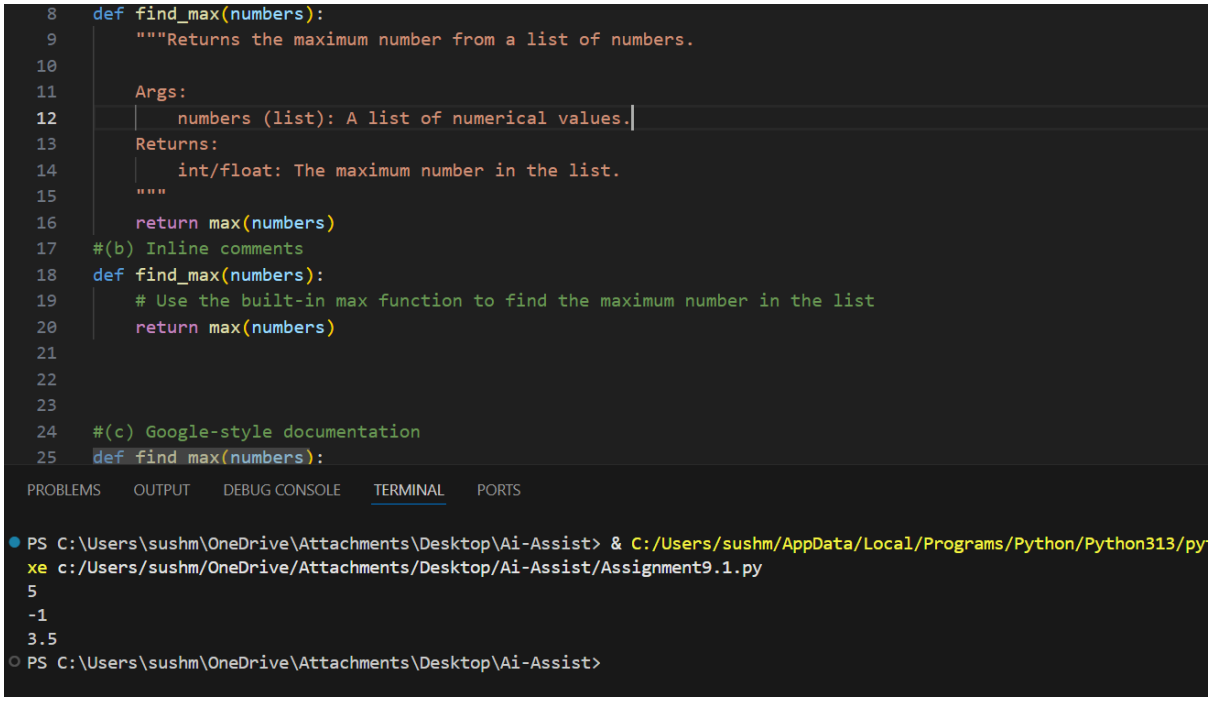
Task:

• Write documentation for the function in all three formats:

Prompt:

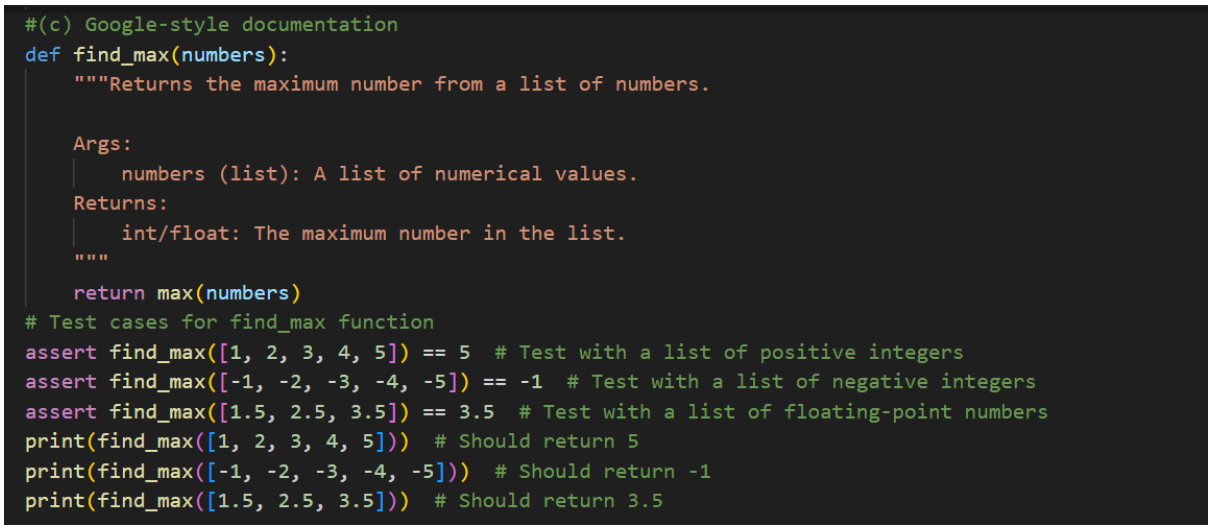
Create a Python function find_max(numbers) to return the maximum value from a list and document it using a docstring, inline comments, and Google-style documentation. Include test cases.

Code:



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

    Args:
        numbers (list): A list of numerical values.
    Returns:
        int/float: The maximum number in the list.
    """
    return max(numbers)
#(b) Inline comments
def find_max(numbers):
    # Use the built-in max function to find the maximum number in the list
    return max(numbers)



#(c) Google-style documentation
def find_max(numbers):
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/python.exe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment9.1.py
5
-1
3.5
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist>
```



```
#(c) Google-style documentation
def find_max(numbers):
    """Returns the maximum number from a list of numbers.

    Args:
        numbers (list): A list of numerical values.
    Returns:
        int/float: The maximum number in the list.
    """
    return max(numbers)
# Test cases for find_max function
assert find_max([1, 2, 3, 4, 5]) == 5  # Test with a list of positive integers
assert find_max([-1, -2, -3, -4, -5]) == -1  # Test with a list of negative integers
assert find_max([1.5, 2.5, 3.5]) == 3.5  # Test with a list of floating-point numbers
print(find_max([1, 2, 3, 4, 5]))  # Should return 5
print(find_max([-1, -2, -3, -4, -5]))  # Should return -1
print(find_max([1.5, 2.5, 3.5]))  # Should return 3.5
```

Advantages, disadvantages, and suitable use cases of each Style.

- The three documentation styles—docstring, inline comments, and Google-style documentation—each have their own advantages and disadvantages.

- Docstrings are the most commonly used and recommended approach in Python. They are integrated into the Python documentation system and are accessible via `help()` or `__doc__`. They are ideal for documenting functions, classes, and modules in a standardized way.

- Inline comments are useful for explaining specific lines of code that may not be self-explanatory. However, they are not as accessible or standardized as docstrings and can clutter the code if overused.

- Google-style documentation is a popular alternative that provides more structured formatting for documenting parameters, return values, and exceptions. It is especially useful in large projects where consistency in documentation is important.

Mathematical utilities library and justifying answer.

For a mathematical utilities library, docstrings are most effective because they provide clear, accessible documentation that integrates well with Python's ecosystem. They support standard practices for documenting functions and modules in a way that is both readable and maintainable. Additionally, docstrings can be easily accessed by users of the library, making it easier for them to understand how to use the functions without needing to read through the code.

Observation:

The find_max function was implemented using the built-in max() function. All test cases passed successfully, and documentation was provided in three different formats.

Problem2: Consider the following Python function:

def login(user, password, credentials):

return credentials.get(user) == password

Task:

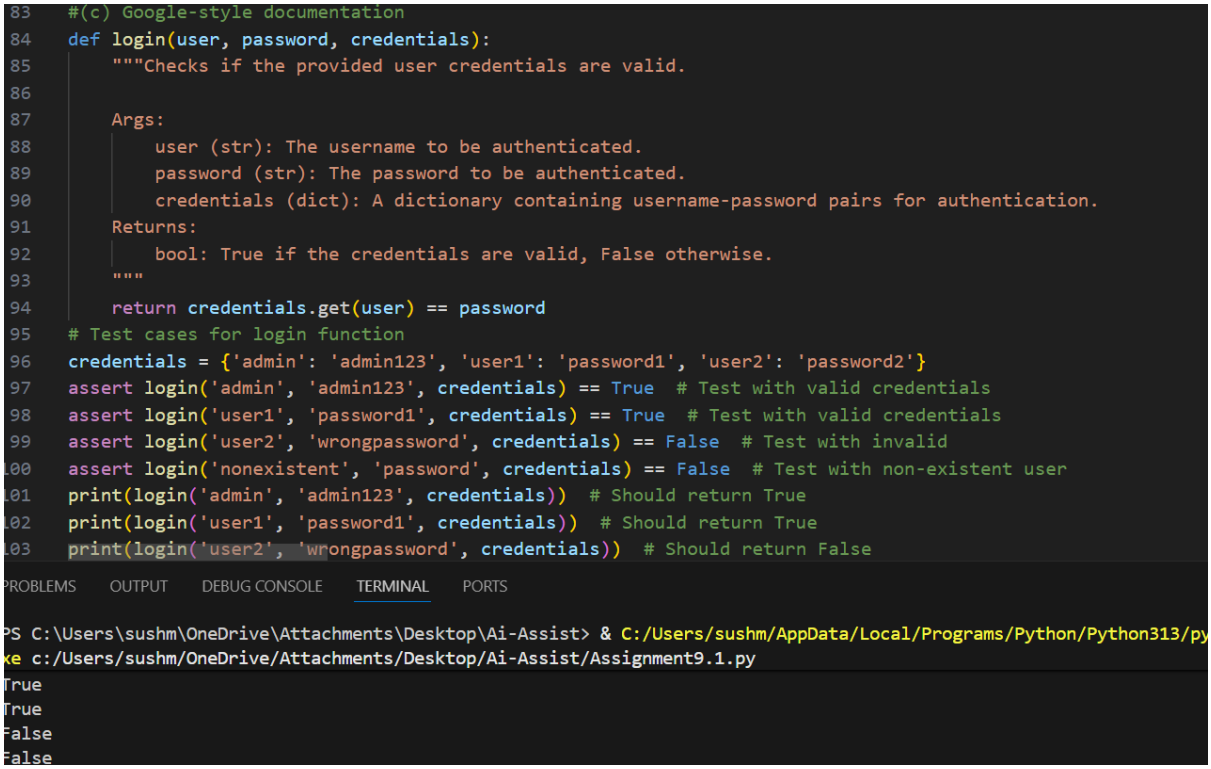
1. Write documentation in all three formats.

Prompt:

Create a Python function login(user, password, credentials) to validate user authentication using a dictionary of stored credentials. Provide documentation in three formats: docstring, inline comments, and Google-style documentation. Include test cases to verify functionality.

Code:

```
#(a) Docstring
def login(user, password, credentials):
    """Checks if the provided user credentials are valid.

    Args:
        user (str): The username to be authenticated.
        password (str): The password to be authenticated.
        credentials (dict): A dictionary containing username-password pairs for authentication.
    Returns:
        bool: True if the credentials are valid, False otherwise.
    """
    return credentials.get(user) == password
#(b) Inline comments
def login(user, password, credentials):
    # Check if the provided user exists in the credentials dictionary and if the password matches
    return credentials.get(user) == password

#(c) Google-style documentation
def login(user, password, credentials):
```



```python
#(c) Google-style documentation
def login(user, password, credentials):
    """Checks if the provided user credentials are valid.

    Args:
        user (str): The username to be authenticated.
        password (str): The password to be authenticated.
        credentials (dict): A dictionary containing username-password pairs for authentication.
    Returns:
        bool: True if the credentials are valid, False otherwise.
    """
    return credentials.get(user) == password
# Test cases for login function
credentials = {'admin': 'admin123', 'user1': 'password1', 'user2': 'password2'}
assert login('admin', 'admin123', credentials) == True  # Test with valid credentials
assert login('user1', 'password1', credentials) == True  # Test with valid credentials
assert login('user2', 'wrongpassword', credentials) == False  # Test with invalid
assert login('nonexistent', 'password', credentials) == False  # Test with non-existent user
print(login('admin', 'admin123', credentials))  # Should return True
print(login('user1', 'password1', credentials))  # Should return True
print(login('user2', 'wrongpassword', credentials))  # Should return False
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/py
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment9.1.py
True
True
False
False
```

Comparison of approaches.

- The three documentation styles—docstring, inline comments, and Google-style documentation—each have their own advantages and disadvantages.

- Docstrings are the most commonly used and recommended approach in Python. They are integrated into the Python documentation system and are accessible via `help()` or `__doc__`. They are ideal for documenting functions, classes, and modules in a standardized way.

- Inline comments are useful for explaining specific lines of code that may not be self-explanatory. However, they are not as accessible or standardized as docstrings and can clutter the code if overused.

- Google-style documentation is a popular alternative that provides more structured formatting for documenting

parameters, return values and exceptions. It is especially useful in large projects where consistency in documentation is important.

Recommend which style would be most helpful for new developers onboarding a project.

For new developers onboarding a project, Google-style documentation would be most helpful. This is because it provides a clear and structured format for documenting functions, parameters, return values, and exceptions.

The consistency in formatting makes it easier for new developers to quickly understand the purpose and usage of each function without having to read through the code

Observation:

The login function was successfully implemented to authenticate users by comparing input credentials with stored values in a dictionary. Documentation was written in three formats, and all test cases passed, confirming correct behavior for valid, invalid, and non-existent users.

Problem 3: Calculator (Automatic Documentation Generation)

Task: Design a Python module named calculator.py and

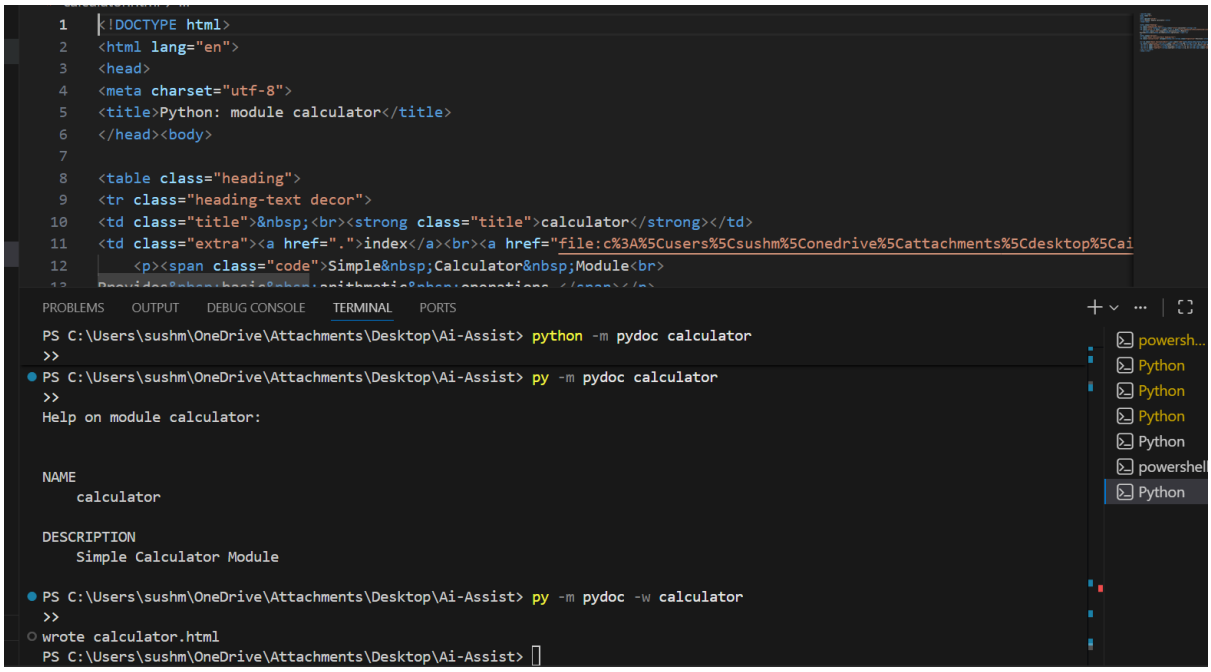
demonstrate automatic documentation generation.

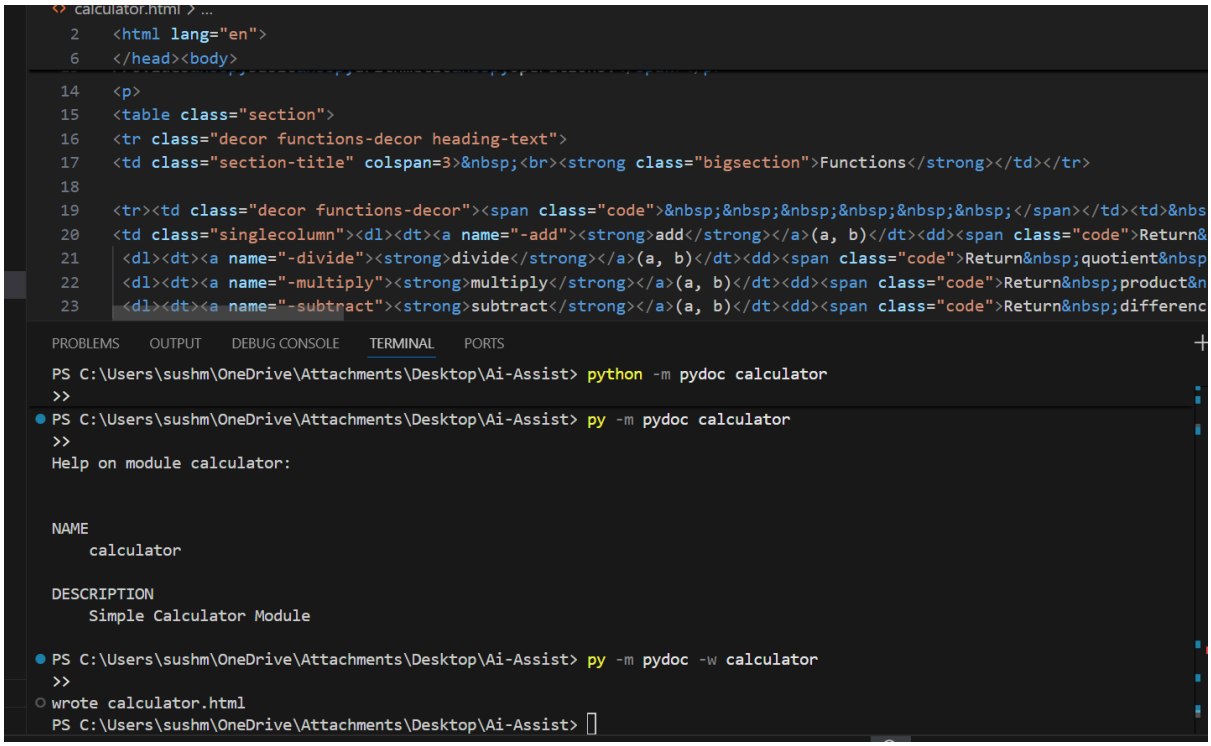
Instructions

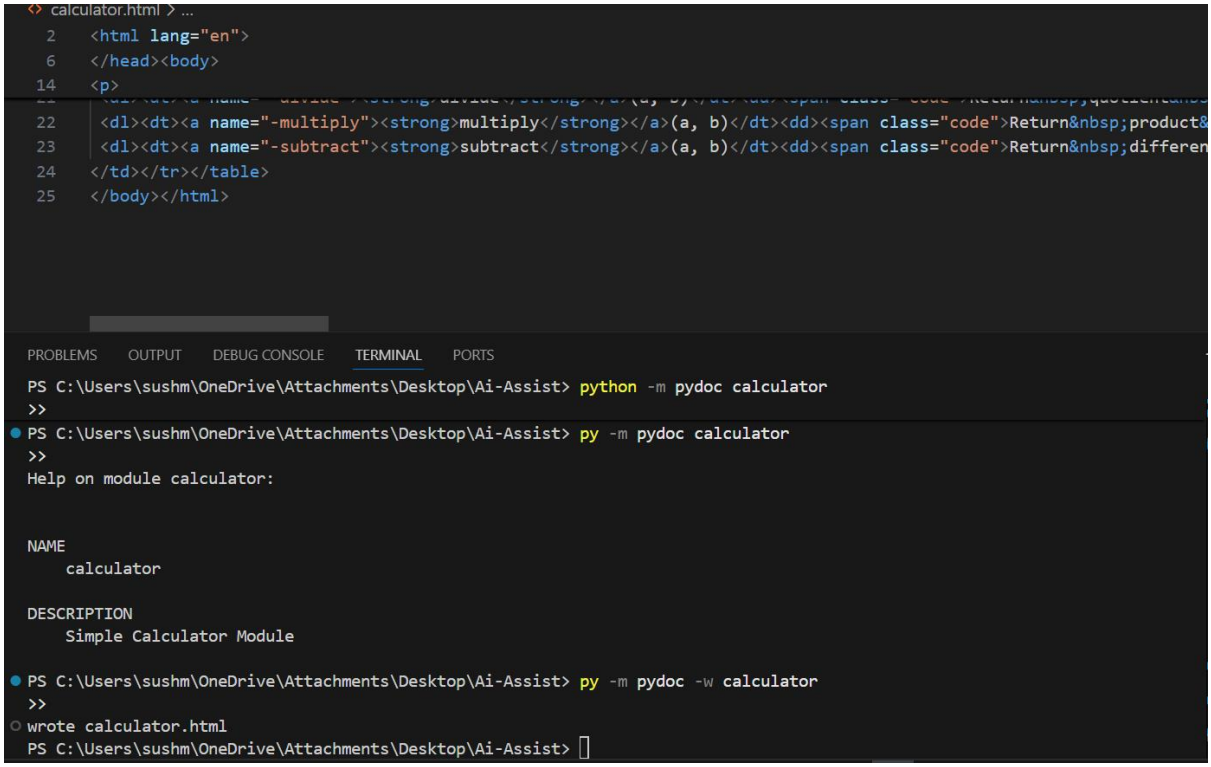
Prompt:

Create a calculator.py module with basic arithmetic functions and generate documentation using pydoc.

Code:







Observation:

The calculator module was created successfully with proper docstrings. Documentation was generated in the terminal and exported as an HTML file.

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)

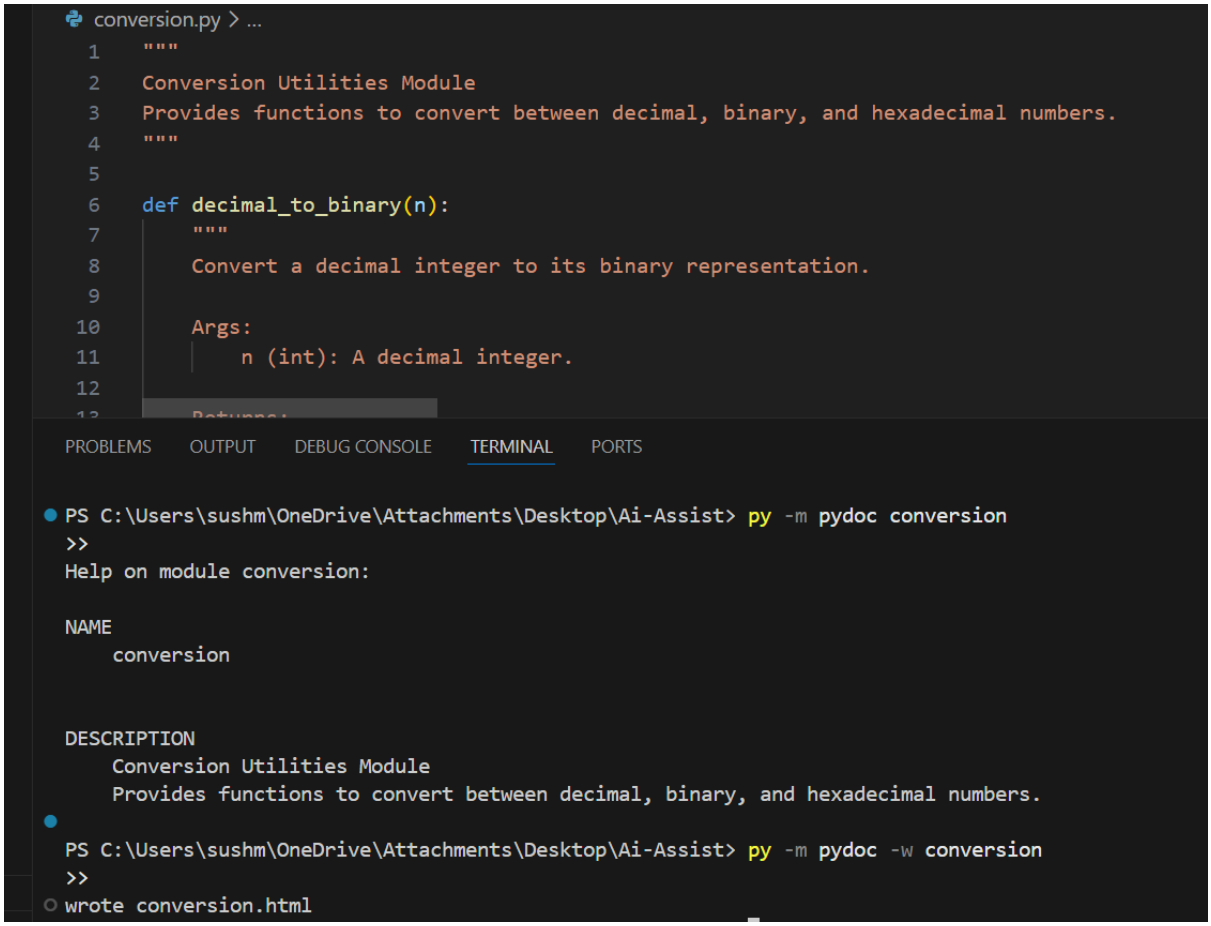
o decimal_to_hexadecimal(n)

Prompt: Create a Python module named conversion.py containing functions to convert:

- Decimal to Binary
- Binary to Decimal
- Decimal to Hexadecimal

Use Copilot to auto-generate docstrings.
Generate documentation in the terminal using pydoc and export it in HTML format.

Code:



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


def binary_to_decimal(b):
    """
    Convert a binary string to its decimal representation.
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> py -m pydoc conversion
>>
Help on module conversion:

NAME
    conversion


DESCRIPTION
    Conversion Utilities Module
    Provides functions to convert between decimal, binary, and hexadecimal numbers.

PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> py -m pydoc -w conversion
>>
wrote conversion.html
```

```
    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):
    """
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> py -m pydoc conversion
>>
Help on module conversion:

NAME
    conversion


DESCRIPTION
    Conversion Utilities Module
    Provides functions to convert between decimal, binary, and hexadecimal numbers.

PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> py -m pydoc -w conversion
>>
wrote conversion.html
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist>
```

```
def decimal_to_hexadecimal(n):
    """
    Convert a decimal integer to its hexadecimal representation.

    Args:
        n (int): A decimal integer.

    Returns:
        str: Hexadecimal representation of the number.
    """
    return hex(n)[2:]

```

Observation:

The conversion.py module was successfully created with three conversion functions.
Docstrings were automatically generated and displayed using pydoc in the terminal.
The documentation was also exported as an HTML file (conversion.html) and opened successfully in a web browser.

Problem 5 – Course Management Module

Task:

1. Create a module course.py with functions:

o add_course(course_id, name, credits)

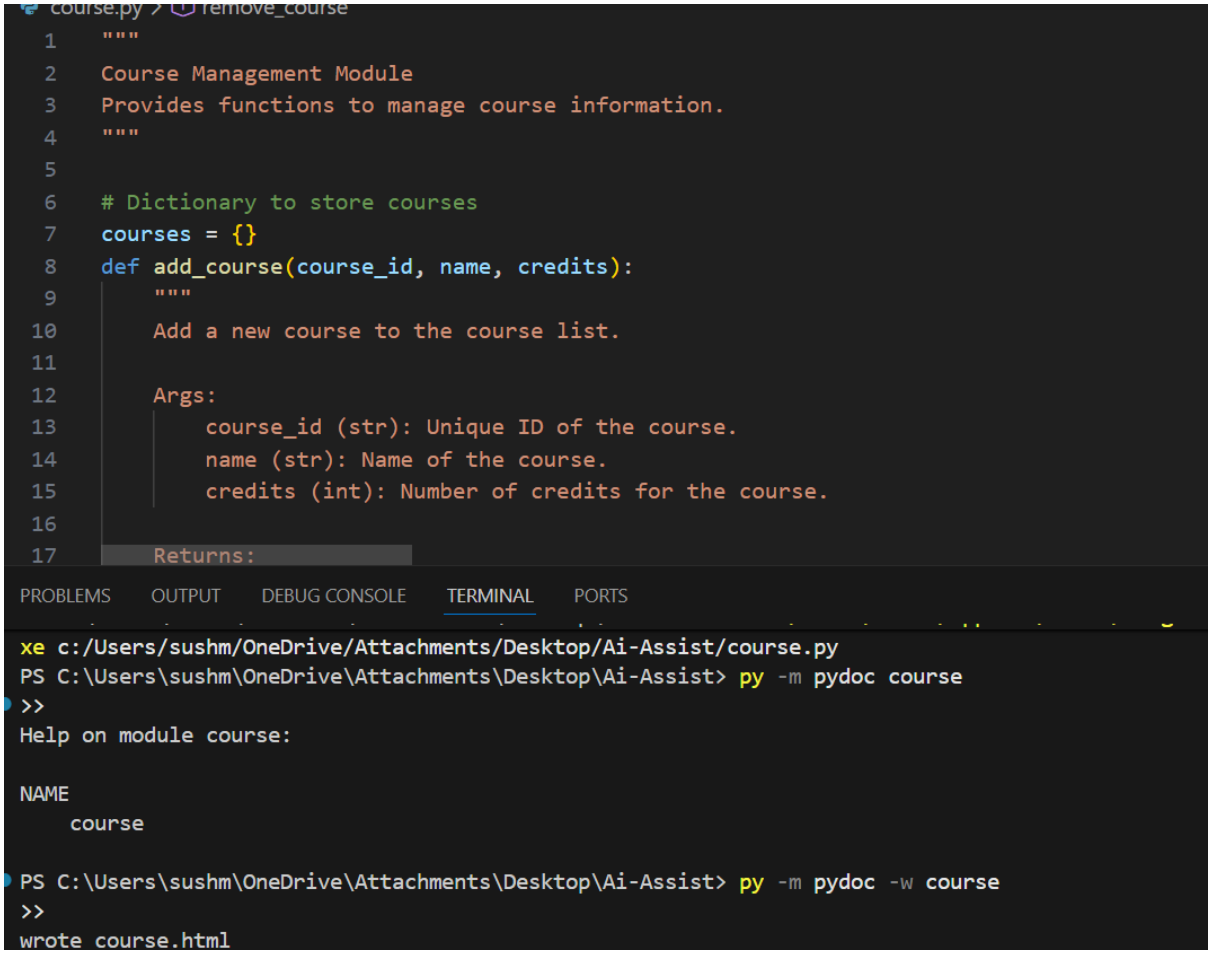
o remove_course(course_id)

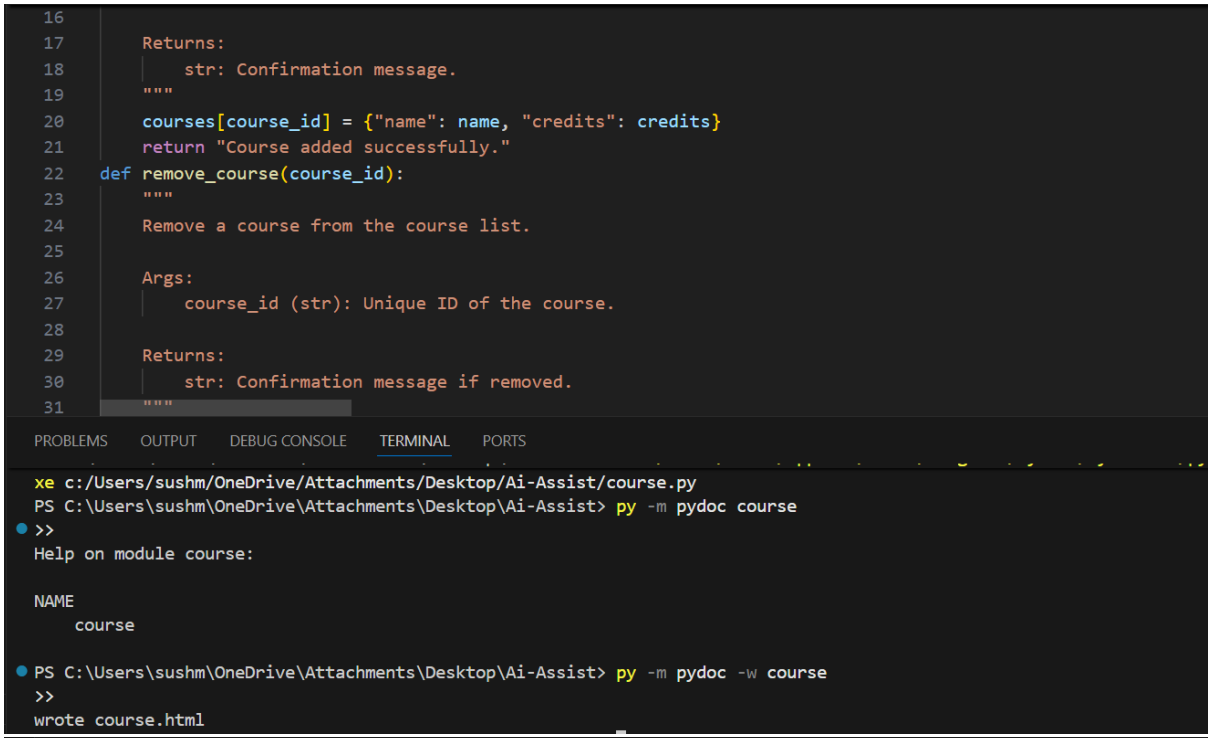
o get_course(course_id)

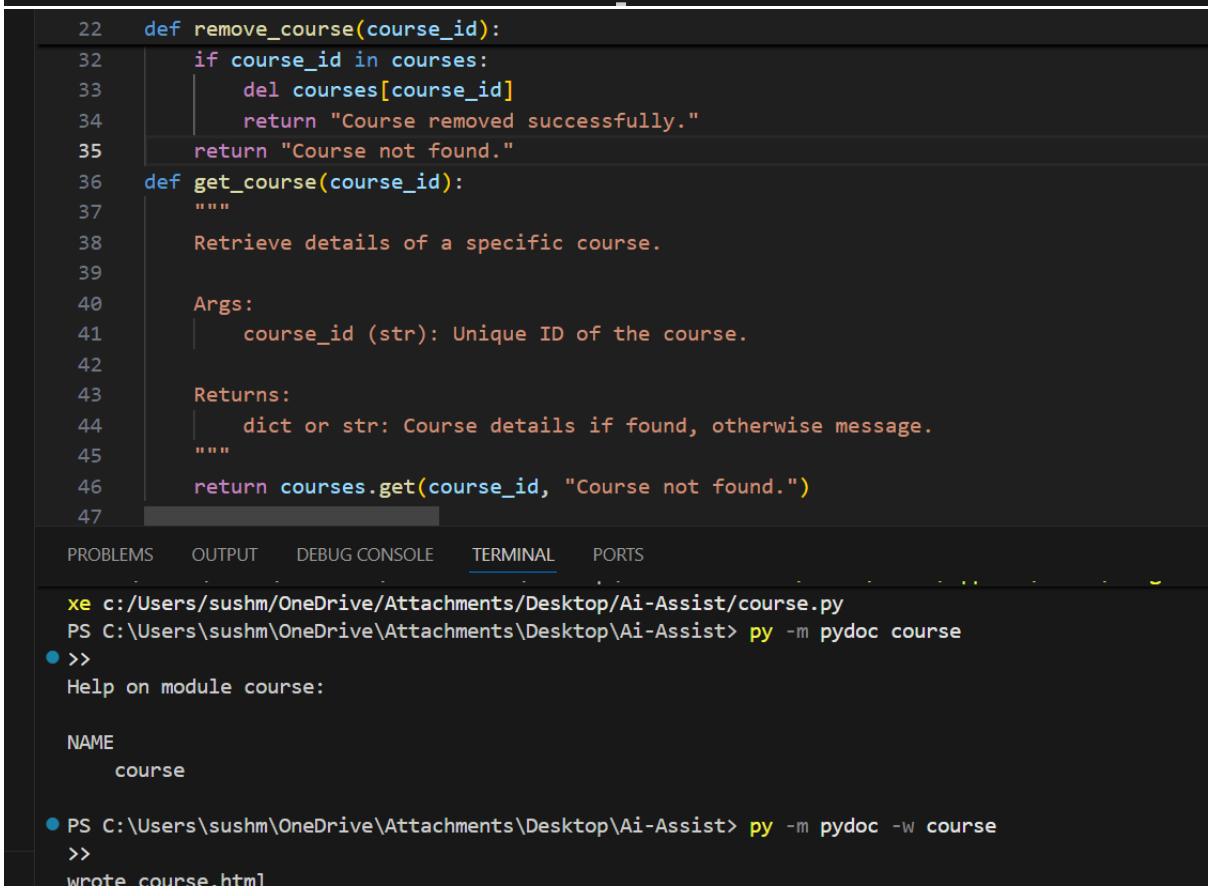
Prompt:

Create a module course.py to manage courses with functions to add, remove, and retrieve course details. Generate documentation using pydoc and export it in HTML format.

Code:


course.py > remove_course
"""
Course Management Module
Provides functions to manage course information.
"""
# Dictionary to store courses
courses = {}
def add_course(course_id, name, credits):
"""
Add a new course to the course list.
Args:
course_id (str): Unique ID of the course.
name (str): Name of the course.
credits (int): Number of credits for the course.
Returns:
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/course.py
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> py -m pydoc course
>>
Help on module course:
NAME
course
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> py -m pydoc -w course
>>
wrote course.html





Observation:

The course.py module was successfully created with three functions for course management. Documentation was generated in the terminal

using pydoc and exported as an HTML file, which opened correctly in a web browser**.**