AI-Assisted-Coding Assignment-9.1

2303A51878

B-28

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

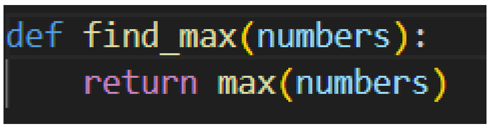
(b) Inline comments

(c) Google-style documentation

• Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.

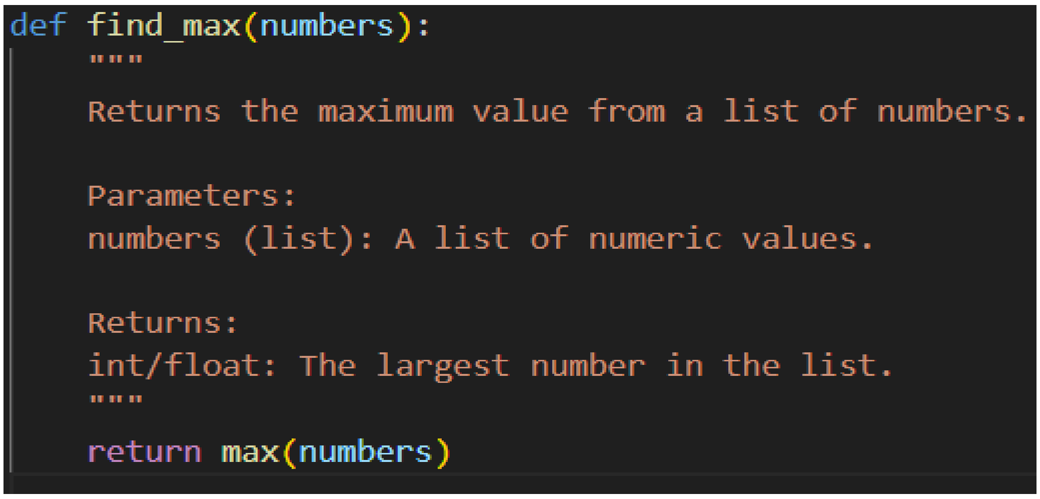
• Recommend which documentation style is most effective for a mathematical utilities library and justify your answer.

Given Function



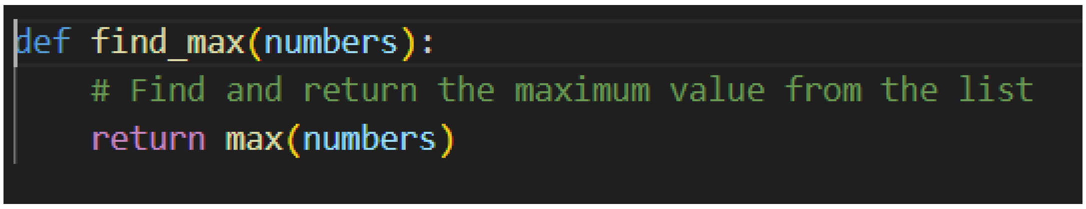
```python
def find_max(numbers):
    return max(numbers)
```

(a) Documentation Using Docstring



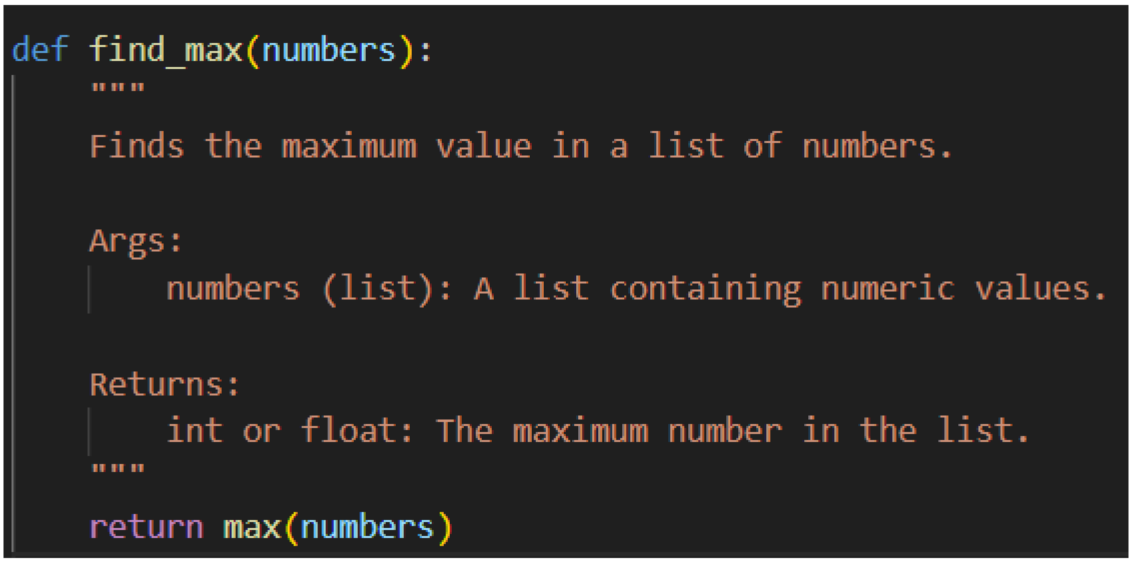
```python
def find_max(numbers):
    """
    Returns the maximum value from a list of numbers.

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

    Returns:
    int/float: The largest number in the list.
    """
    return max(numbers)
```

(b) Documentation Using Inline Comments



```python
def find_max(numbers):
    # Find and return the maximum value from the list
    return max(numbers)
```

(c) Google-Style Documentation



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

    Args:
        numbers (list): A list containing numeric values.

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

Critical Comparison of Documentation Styles

Style	**Advantages**	**Disadvantages**	**Suitable Use Cases**
Docstring (Basic)	Easy to write, readable, accessible via help()	No strict format, can become inconsistent	Small scripts, academic work
Inline Comments	Simple, improves immediate readability	Not accessible to documentation tools, redundant for simple logic	Complex logic explanation
Google-Style Docstring	Structured, professional, tool-friendly	Slightly longer to write	Libraries, APIs, team projects

Recommendation for a Mathematical Utilities Library

Best Choice: Google-Style Documentation

Justification:

- Mathematical utility libraries are reused by many developers
- Google-style docstrings:
    - Clearly describe inputs, outputs, and types
    - Are compatible with documentation generators like Sphinx
    - Improve maintainability and consistency
- Inline comments are unnecessary for simple math operations
- Basic docstrings lack structure for large libraries

Conclusion:

For a mathematical utilities library, Google-style documentation is the most effective and professional choice because it balances clarity, structure, and scalability.

Problem 2: Consider the following Python function:

def login(user, password, credentials):

return credentials.get(user) == password

Task:

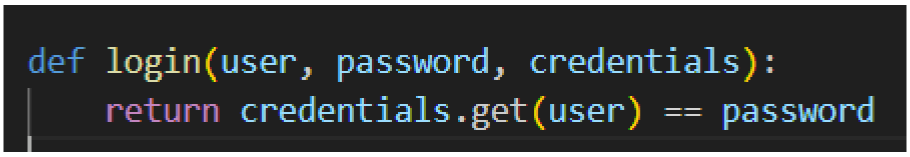
1. Write documentation in all three formats.

2. Critically compare the approaches.

3. Recommend which style would be most helpful for new

developers onboarding a project, and justify your choice.

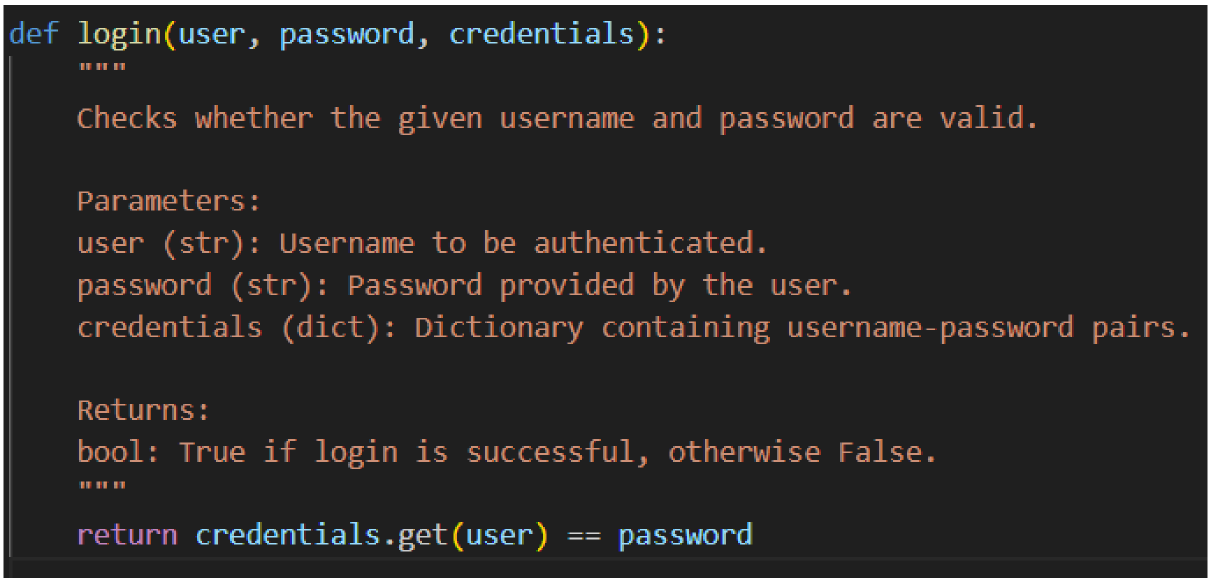
Given Function



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

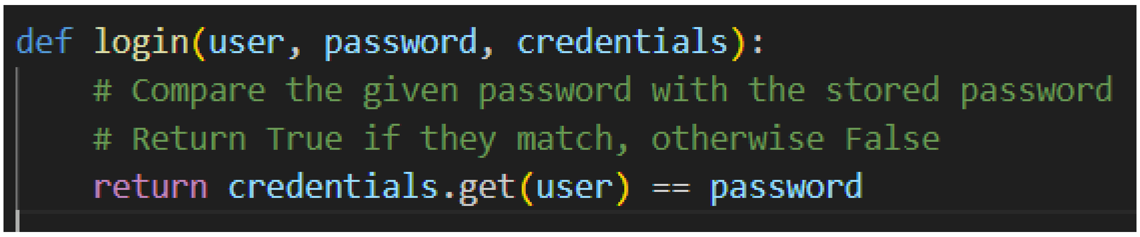
1. Documentation in All Three Formats

(a) Docstring Documentation



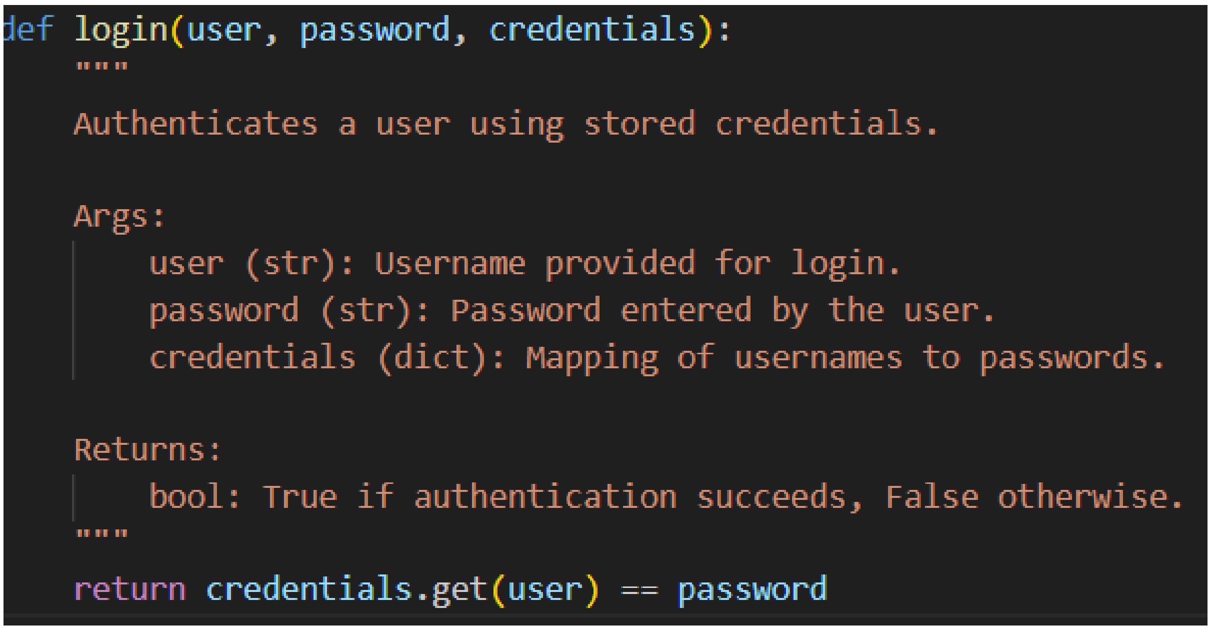
```python
def login(user, password, credentials):
    """
    Checks whether the given username and password are valid.

    Parameters:
    user (str): Username to be authenticated.
    password (str): Password provided by the user.
    credentials (dict): Dictionary containing username-password pairs.

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

(b) Inline Comments



```python
def login(user, password, credentials):
    # Compare the given password with the stored password
    # Return True if they match, otherwise False
    return credentials.get(user) == password
```

(c) Google-Style Documentation



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

    Args:
        user (str): Username provided for login.
        password (str): Password entered by the user.
        credentials (dict): Mapping of usernames to passwords.

    Returns:
        bool: True if authentication succeeds, False otherwise.
    """
    return credentials.get(user) == password
```

2. Critical Comparison of the Three Approaches

Documentation Style	Advantages	Disadvantages	Best Use Case
Docstring (Basic)	Simple, readable, available via help()	No strict format, may vary between developers	Small projects, assignments
Inline Comments	Easy to understand line-by-line	Not visible to documentation tools, unnecessary for simple logic	Explaining complex logic
Google-Style Docstring	Clear structure, professional, tool-friendly	Slightly verbose	Team projects, APIs, onboarding

3. Recommended Style for New Developer Onboarding

Best Choice: Google-Style Documentation

Justification:

- New developers need clear understanding of function purpose
- Google-style documentation:
    - Clearly explains what the function does
    - Describes inputs, outputs, and data types
    - Is consistent and easy to read
- Helps new team members understand code without reading implementation
- Works well with automated documentation tools

Conclusion:

For onboarding new developers, Google-style documentation is the most helpful because it is structured, readable, and reduces confusion in collaborative projects.

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

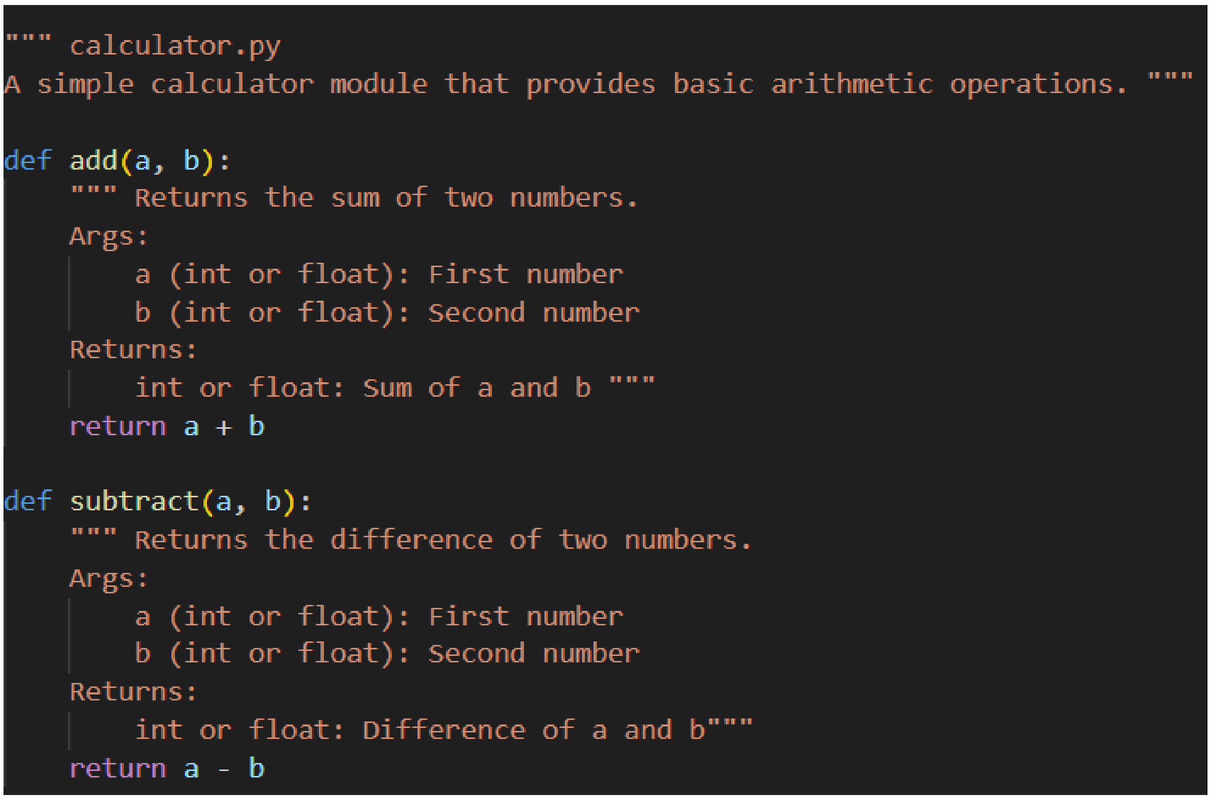
o multiply(a, b) – returns the product of two numbers

o divide(a, b) – returns the quotient of two numbers

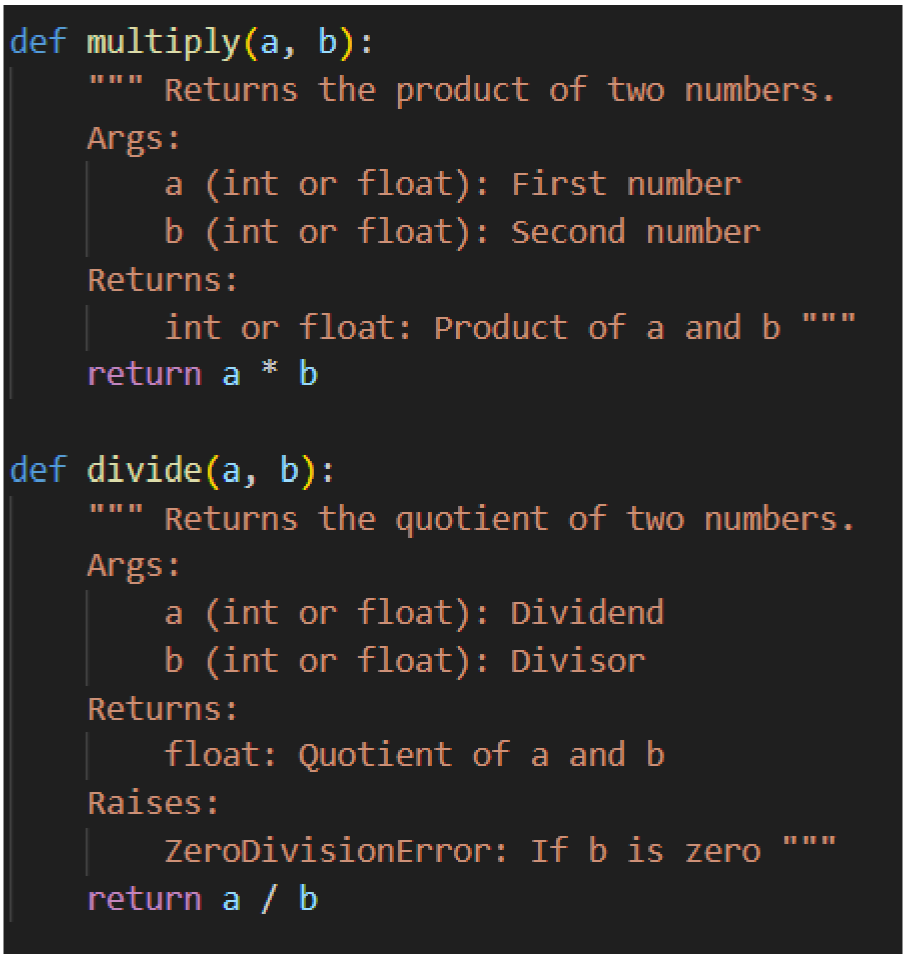
2. Display the module documentation in the terminal using Python's documentation tools.

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.

1. Create the Python module calculator.py



```
""" calculator.py
A simple calculator module that provides basic arithmetic operations. """

def add(a, b):
    """ Returns the sum of two numbers.
    Args:
        a (int or float): First number
        b (int or float): Second number
    Returns:
        int or float: Sum of a and b """
    return a + b

def subtract(a, b):
    """ Returns the difference of two numbers.
    Args:
        a (int or float): First number
        b (int or float): Second number
    Returns:
        int or float: Difference of a and b"""
    return a - b
```



```python
def multiply(a, b):
    """ Returns the product of two numbers.
    Args:
        a (int or float): First number
        b (int or float): Second number
    Returns:
        int or float: Product of a and b """
    return a * b

def divide(a, b):
    """ Returns the quotient of two numbers.
    Args:
        a (int or float): Dividend
        b (int or float): Divisor
    Returns:
        float: Quotient of a and b
    Raises:
        ZeroDivisionError: If b is zero """
    return a / b
```

2. Display Module Documentation in the Terminal

Python provides built-in documentation tools.

Method 1: Using help() in Python Interpreter

python

import calculator

help(calculator)

This displays:

- Module description
- Function names
- Function docstrings
- Parameters and return values

Method 2: Using pydoc in Terminal

pydoc calculator

This prints the complete module documentation directly in the terminal.

3. Generate HTML Documentation Using pydoc

Step 1: Generate HTML file

Run this command in the directory containing calculator.py:

pydoc -w calculator

This creates a file named:

calculator.html

Step 2: Open the HTML File in a Browser

- Double-click calculator.html
  OR
- Open it manually in any web browser (Chrome, Edge, Firefox)

You will see:

- Module description
- Function list
- Detailed docstrings in a clean HTML layout

Explanation Docstrings allow automatic documentation generation

- Python's help() and pydoc tools read docstrings directly
- pydoc -w exports professional-looking HTML documentation
- This approach reduces manual documentation effort and ensures accuracy

Conclusion:

Automatic documentation generation using docstrings and pydoc is an efficient and reliable way to maintain clear and up-to-date documentation for Python modules.

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)

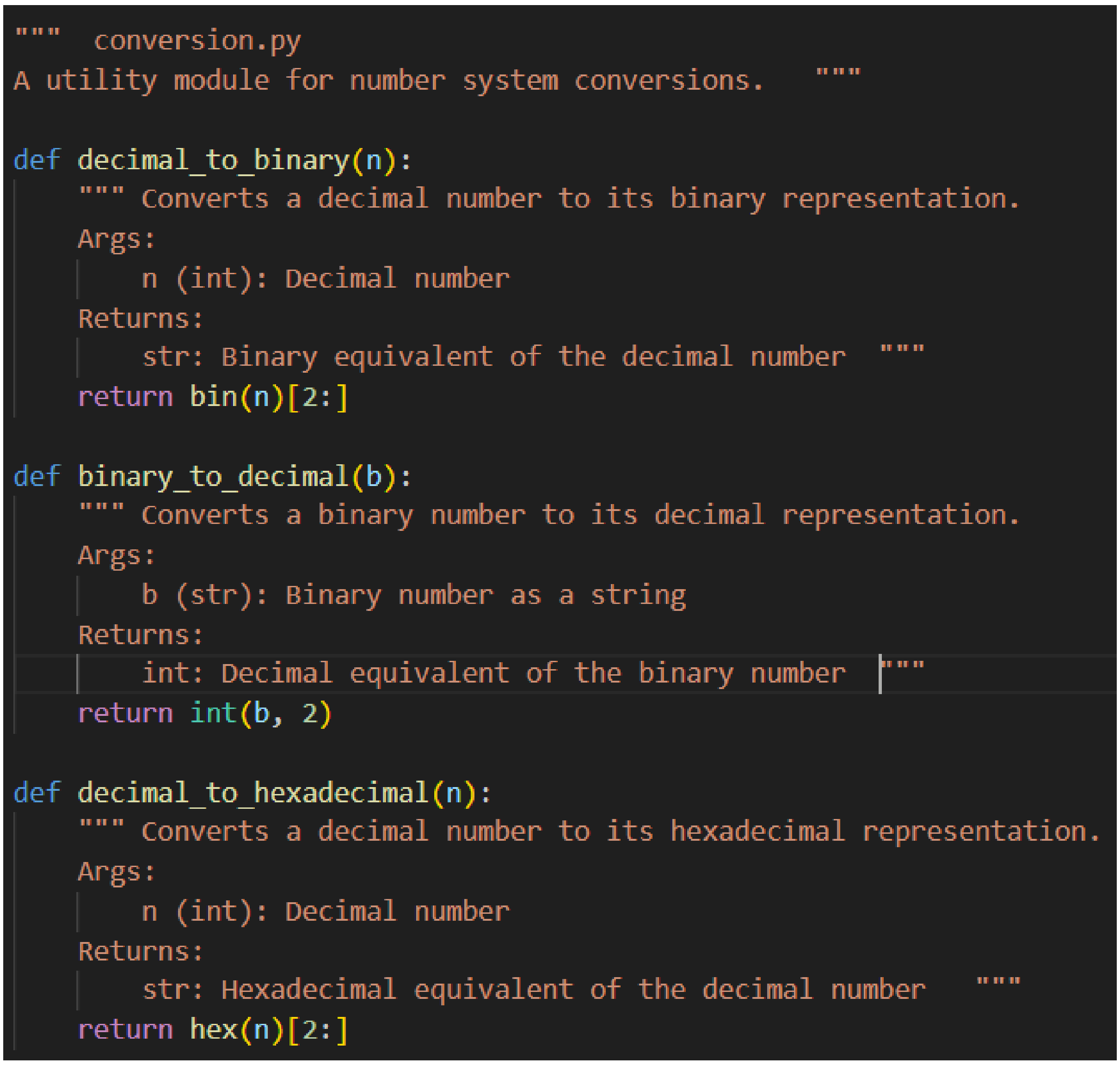
o decimal_to_hexadecimal(n)

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.

1. Create the module conversion.py



```python
"""  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): Decimal number
    Returns:
        str: Binary equivalent of the decimal number  """
    return bin(n)[2:]

def binary_to_decimal(b):
    """ Converts a binary number to its decimal representation.
    Args:
        b (str): Binary number as a string
    Returns:
        int: Decimal equivalent of the binary number  """
    return int(b, 2)

def decimal_to_hexadecimal(n):
    """ Converts a decimal number to its hexadecimal representation.
    Args:
        n (int): Decimal number
    Returns:
        str: Hexadecimal equivalent of the decimal number   """
    return hex(n)[2:]
```

Note on Copilot usage:
Copilot suggests structured docstrings automatically based on function names, parameters, and return values, reducing manual documentation effort.

2. Generate Documentation in the Terminal

Method 1: Using help()

Open Python interpreter:

python

import conversion

help(conversion)

Displays:

- Module description
- Function list
- Parameters and return types
- Docstrings

Method 2: Using pydoc

pydoc conversion

Shows complete module documentation directly in the terminal.

3. Export Documentation in HTML Format

Step 1: Generate HTML documentation

Run the following command in the directory containing conversion.py:

pydoc -w conversion

This creates: conversion.html

Step 2: Open in Web Browser

- Double-click conversion.html
  OR
- Open it manually using Chrome / Edge / Firefox

You will see:

- Module overview
- All conversion functions
- Well-formatted docstrings in HTML

Explanation

- Copilot helps generate consistent and meaningful docstrings
- Python's help() and pydoc automatically extract documentation
- pydoc -w converts docstrings into HTML documentation
- This ensures automatic, accurate, and maintainable documentation

Conclusion:

Automatic documentation using Copilot-assisted docstrings and pydoc improves productivity, consistency, and code readability in utility modules.

Problem 5 – Course Management Module

Task:

1. Create a module course.py with functions:

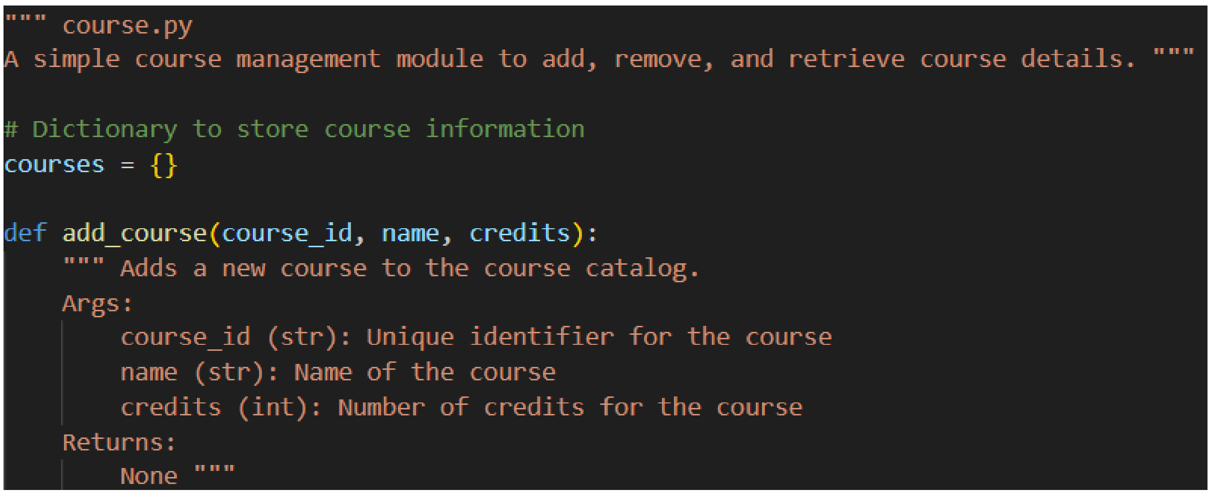
- **add_course(course_id, name, credits)**
- **remove_course(course_id)**
- **get_course(course_id)**

2. Add docstrings with Copilot.

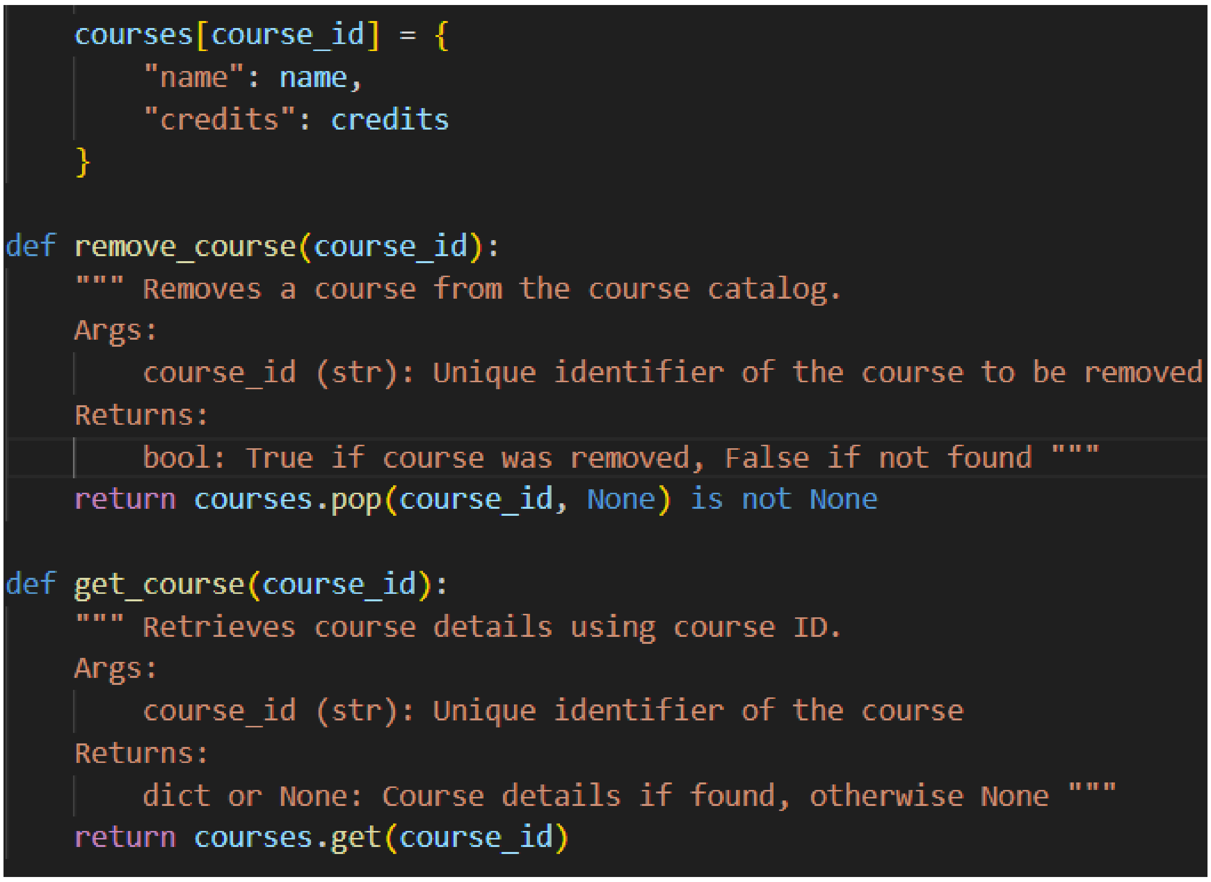
3. Generate documentation in the terminal.

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

1. Create the module course.py



```python
""" course.py
A simple course management module to add, remove, and retrieve course details. """

# Dictionary to store course information
courses = {}

def add_course(course_id, name, credits):
    """ Adds a new course to the course catalog.
    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 from the course catalog.
    Args:
        course_id (str): Unique identifier of the course to be removed
    Returns:
        bool: True if course was removed, False if not found """
    return courses.pop(course_id, None) is not None

def get_course(course_id):
    """ Retrieves course details using course ID.
    Args:
        course_id (str): Unique identifier of the course
    Returns:
        dict or None: Course details if found, otherwise None """
    return courses.get(course_id)
```

2. Generate Documentation in the Terminal

Method 1: Using help()

Open Python interpreter:

python

import course

help(course)

Displays:

- Module description
- Function names
- Parameters and return values
- Docstrings

Method 2: Using pydoc

pydoc course

Shows complete module documentation in the terminal.

3. Export Documentation in HTML Format

Step 1: Generate HTML file

Run this command in the folder containing course.py:

pydoc -w course

This creates: course.html

Step 2: Open in Web Browser

- Double-click course.html
  OR
- Open it manually in any web browser

✓ The HTML page displays:

- Module overview
- All course management functions
- Structured docstrings

Explanation

- Copilot-assisted docstrings reduce manual effort
- Python automatically extracts documentation using help() and pydoc
- pydoc -w converts docstrings into readable HTML documentation
- This approach ensures consistent and maintainable documentation

Conclusion:

Automatic documentation generation using Copilot and pydoc is an efficient way to document course management modules in Python projects.