ASSIGNMENT 9.1

Name: Maripelli Chodhitha

HT No: 2303A52039

Problem 1: Documentation styles for `find_max`

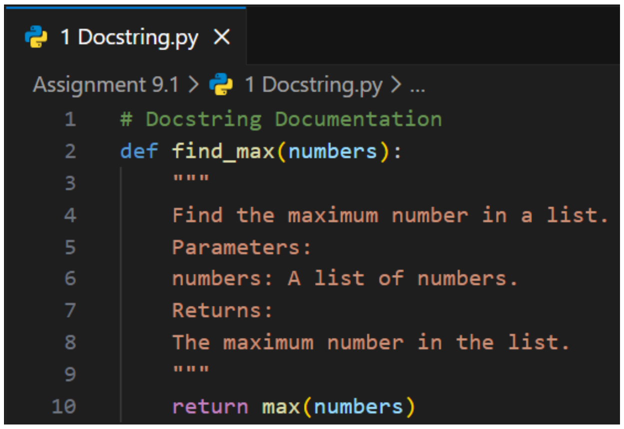
Given function (Python):

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

Task:

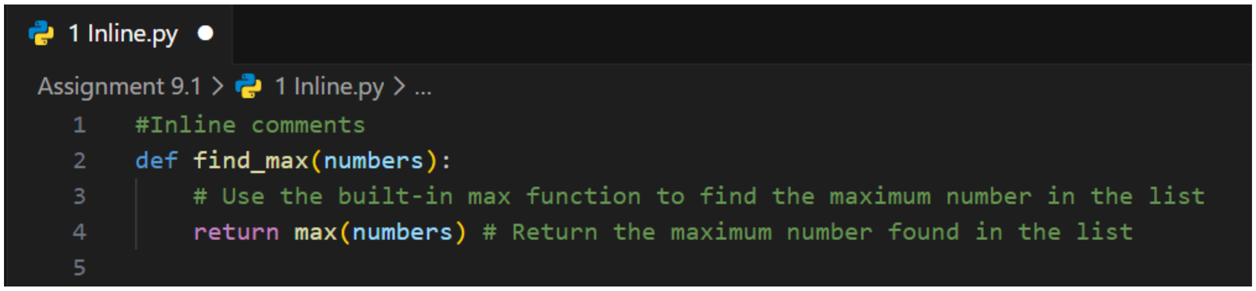
- Write documentation for the function in all three formats:
    - (a) Docstring
    - (b) Inline comments
    - (c) Google-style documentation
- Critically compare the three approaches. Discuss advantages, disadvantages, and suitable use cases of each style.
- Recommend which documentation style is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

(a) Docstring



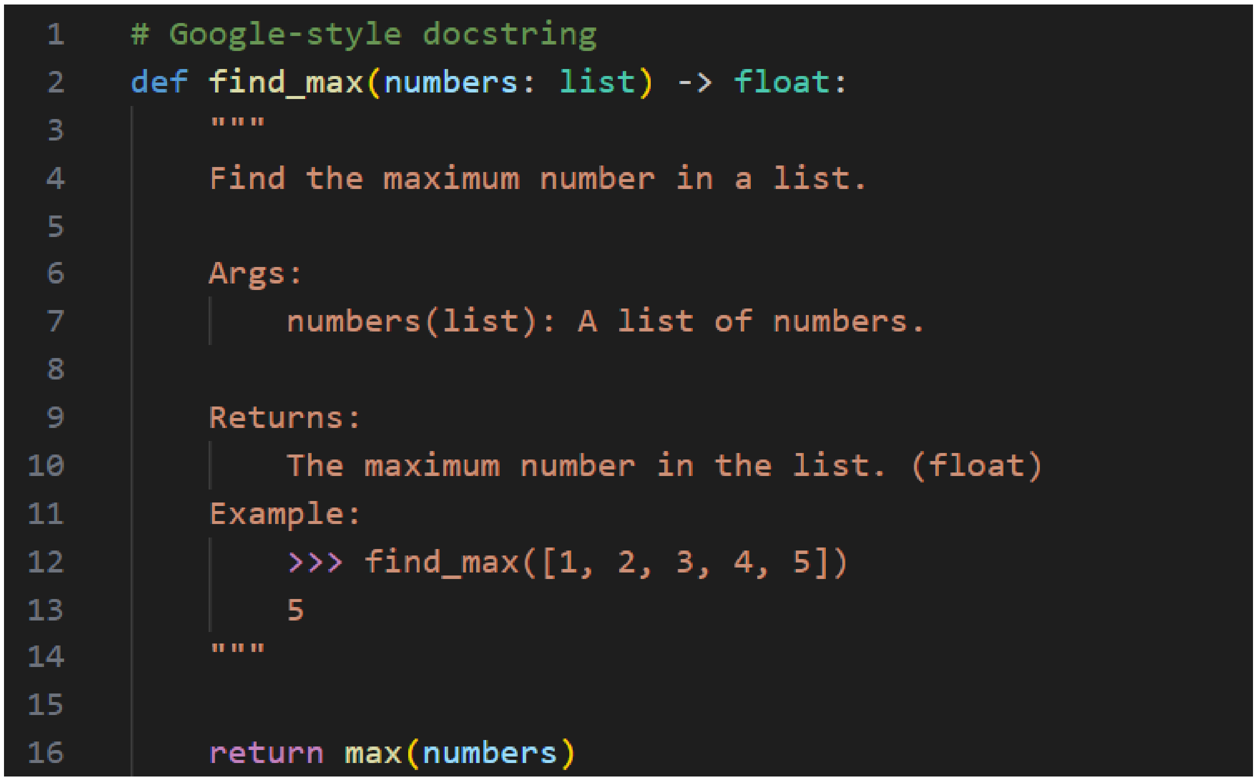
```python
# Docstring Documentation
def find_max(numbers):
    """
    Find the maximum number in a list.
    Parameters:
    numbers: A list of numbers.
    Returns:
    The maximum number in the list.
    """
    return max(numbers)
```

(b) Inline comments



```python
#Inline comments
def find_max(numbers):
    # Use the built-in max function to find the maximum number in the list
    return max(numbers) # Return the maximum number found in the list

```

(c) Google-style documentation



```python
# Google-style docstring
def find_max(numbers: list) -> float:
    """
    Find the maximum number in a list.

    Args:
        numbers(list): A list of numbers.

    Returns:
        The maximum number in the list. (float)
    Example:
        >>> find_max([1, 2, 3, 4, 5])
        5
    """

    return max(numbers)
```

Critical comparison

Aspect	Docstring	Inline Comments	Google-style
Readability	Good	Minimal	Excellent
IDE Support	Native	None	Native (very good)
Formality	Standard	Casual	Professional
Maintenance	Easy	Tedious	Structured
Type hints / parameter clarity	Optional	Unclear	Explicit
Examples	Rare	Never	Common

Format	Advantages	Disadvantages
Docstring	Works with IDE tooltips and help(), simple to add	Can be less structured if not written carefully
Inline comments	Explains specific lines of logic right where they occur	Can clutter code and is not used by documentation tools
Google-style	Professional, structured, supports types and examples well	More verbose and takes more time to write

Recommendation (for a mathematical utilities library):

Google-style documentation is the most effective because it is clear, structured, and consistent across many functions. It also scales well and supports automated documentation generation.

Problem 2: Documentation styles for `login`

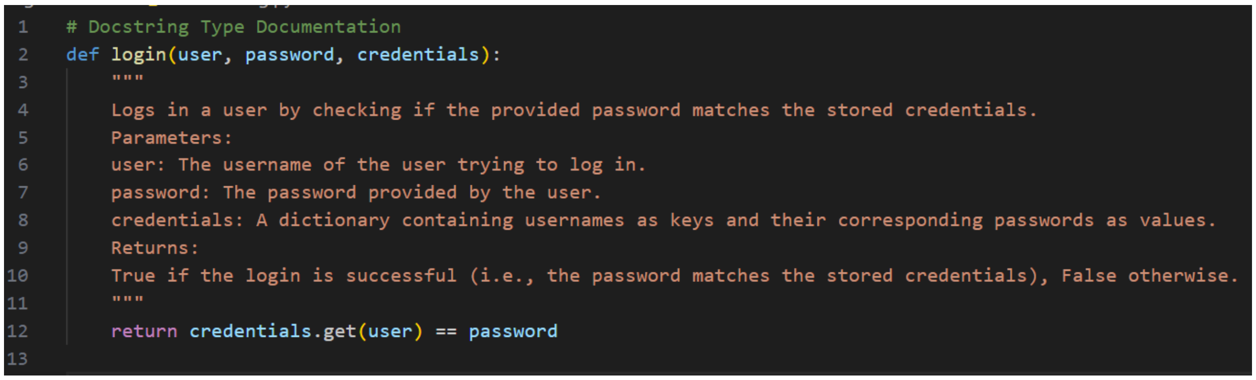
Given function (Python):

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

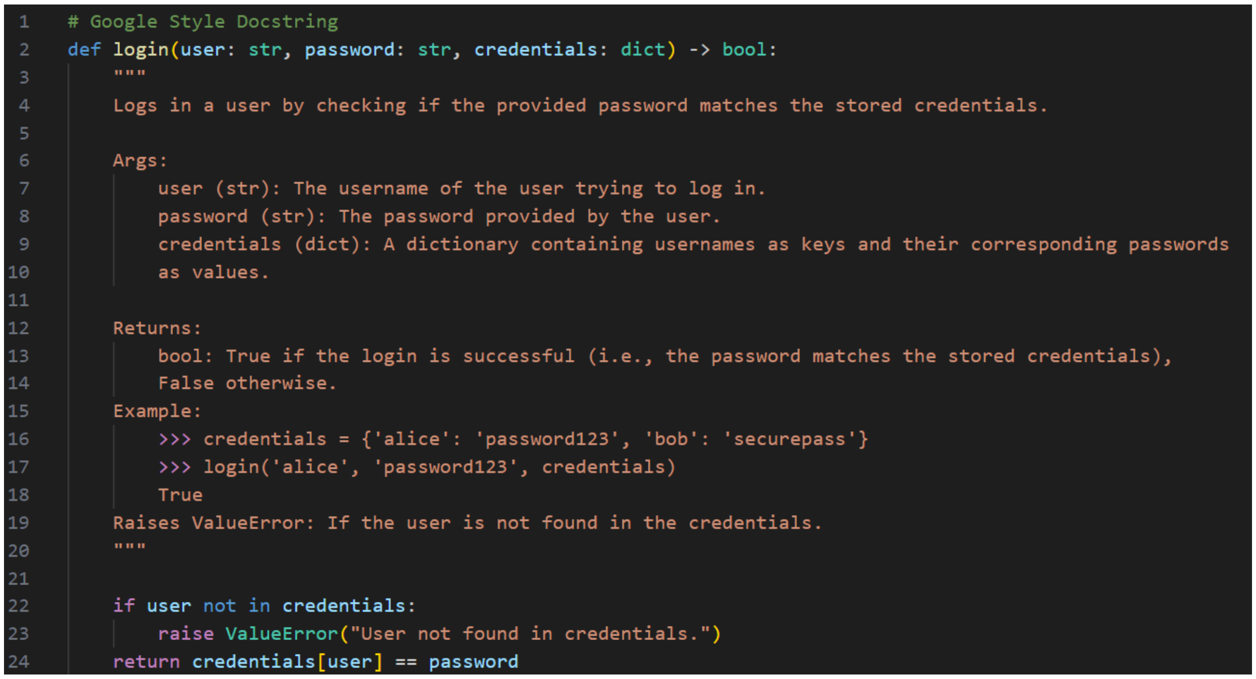
Task:

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.

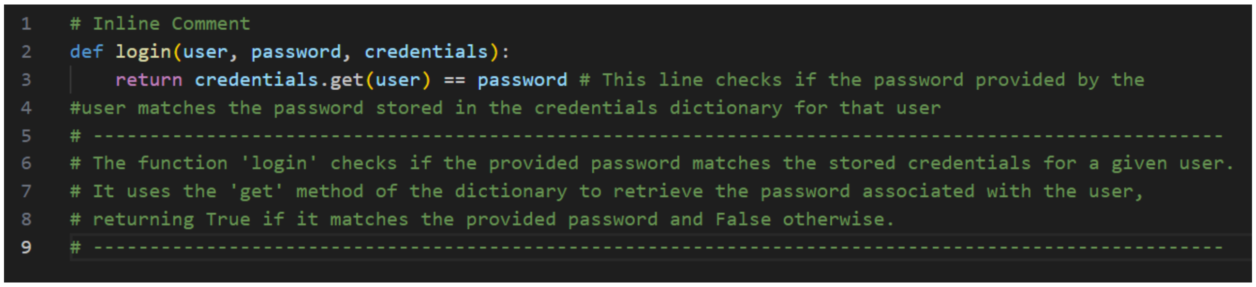
Outputs



```
# Docstring Type Documentation
def login(user, password, credentials):
    """
    Logs in a user by checking if the provided password matches the stored credentials.
    Parameters:
    user: The username of the user trying to log in.
    password: The password provided by the user.
    credentials: A dictionary containing usernames as keys and their corresponding passwords as values.
    Returns:
    True if the login is successful (i.e., the password matches the stored credentials), False otherwise.
    """
    return credentials.get(user) == password

```



```python
# Google Style Docstring
def login(user: str, password: str, credentials: dict) -> bool:
    """
    Logs in a user by checking if the provided password matches the stored credentials.

    Args:
        user (str): The username of the user trying to log in.
        password (str): The password provided by the user.
        credentials (dict): A dictionary containing usernames as keys and their corresponding passwords
        as values.

    Returns:
        bool: True if the login is successful (i.e., the password matches the stored credentials),
        False otherwise.
    Example:
        >>> credentials = {'alice': 'password123', 'bob': 'securepass'}
        >>> login('alice', 'password123', credentials)
        True
    Raises ValueError: If the user is not found in the credentials.
    """

    if user not in credentials:
        raise ValueError("User not found in credentials.")
    return credentials[user] == password
```



```python
# Inline Comment
def login(user, password, credentials):
    return credentials.get(user) == password # This line checks if the password provided by the
#user matches the password stored in the credentials dictionary for that user
# -----------------------------------------------------------------------------------------------
# The function 'login' checks if the provided password matches the stored credentials for a given user.
# It uses the 'get' method of the dictionary to retrieve the password associated with the user,
# returning True if it matches the provided password and False otherwise.
# -----------------------------------------------------------------------------------------------
```

Comparison

Point	Inline	Docstring	Google Style
Clear for beginners	No	Yes	Yes (very clear)
Structured	No	Somewhat	Yes
Professional	No	Medium	Yes
Best for large projects	No	Okay	Yes
Best for onboarding	No	Good	Best

Recommendation (for onboarding):

Google-style documentation is the most helpful because new developers need clarity, structure, and consistency. It clearly explains inputs, outputs, and edge cases, and it supports automated documentation generation.

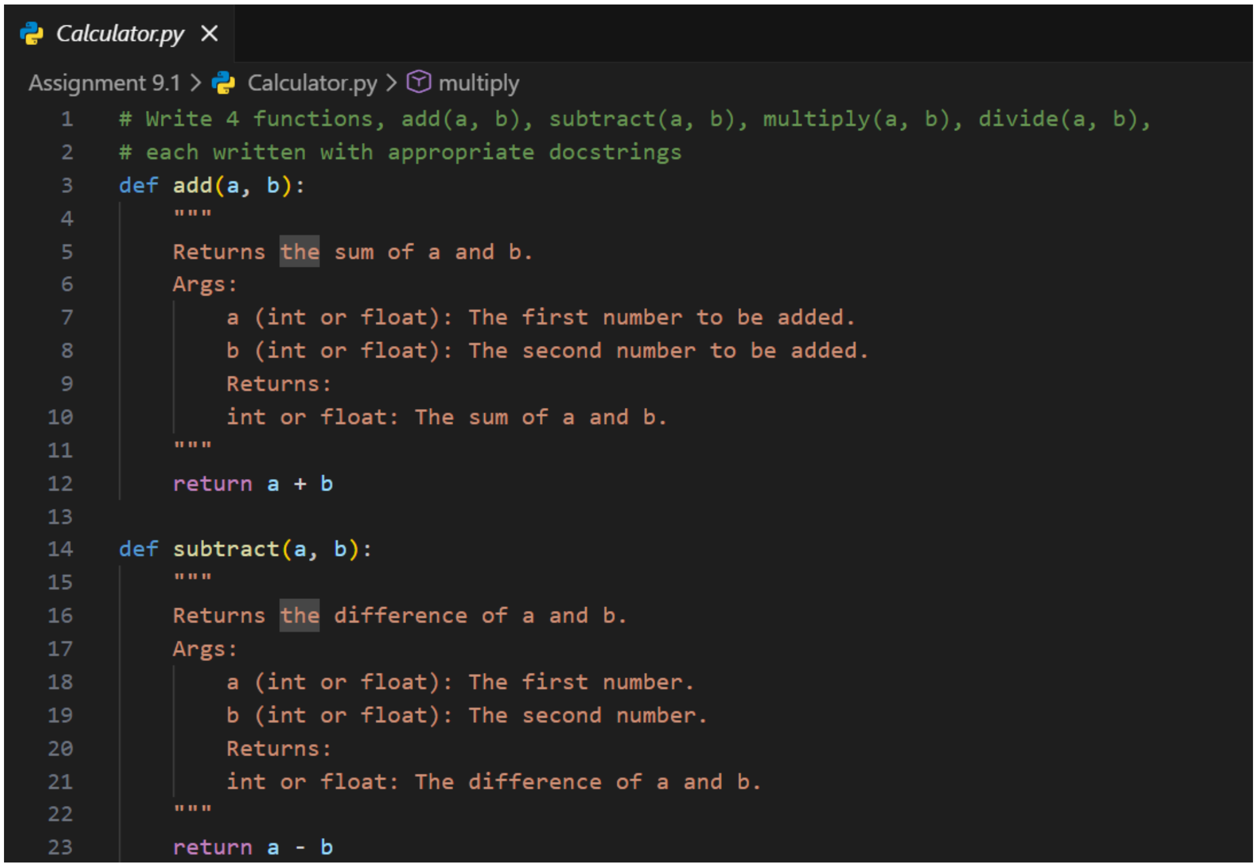
Problem 3: Calculator (Automatic Documentation Generation)

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

Instructions:

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

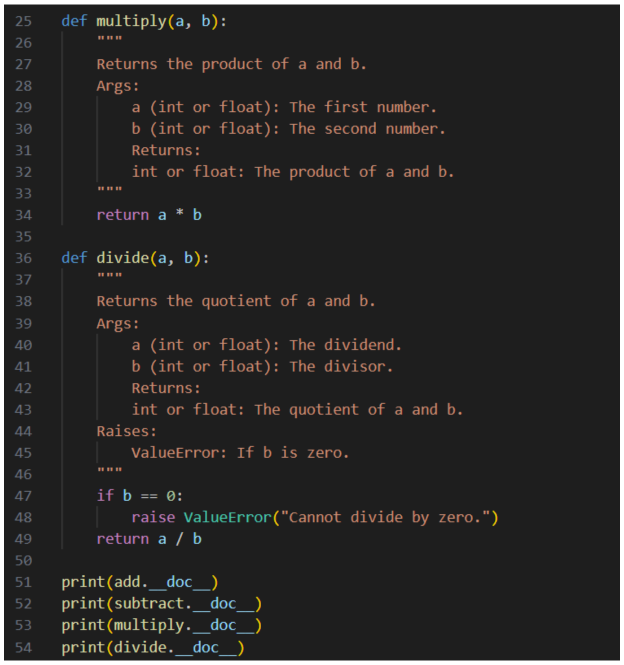
Outputs



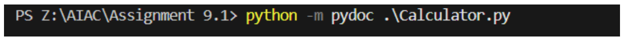
Calculator.py

Assignment 9.1 > Calculator.py > multiply

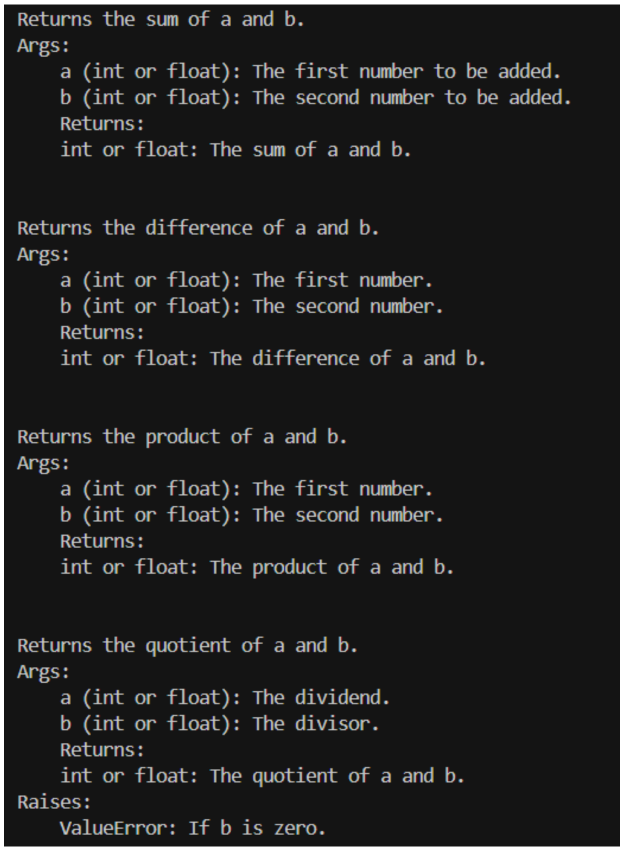
```python
# Write 4 functions, add(a, b), subtract(a, b), multiply(a, b), divide(a, b),
# each written with appropriate docstrings
def add(a, b):
    """
    Returns the sum of a and b.
    Args:
        a (int or float): The first number to be added.
        b (int or float): The second number to be added.
        Returns:
        int or float: The sum of a and b.
    """
    return a + b

def subtract(a, b):
    """
    Returns the difference of a and b.
    Args:
        a (int or float): The first number.
        b (int or float): The second number.
        Returns:
        int or float: The difference of a and b.
    """
    return a - b
```



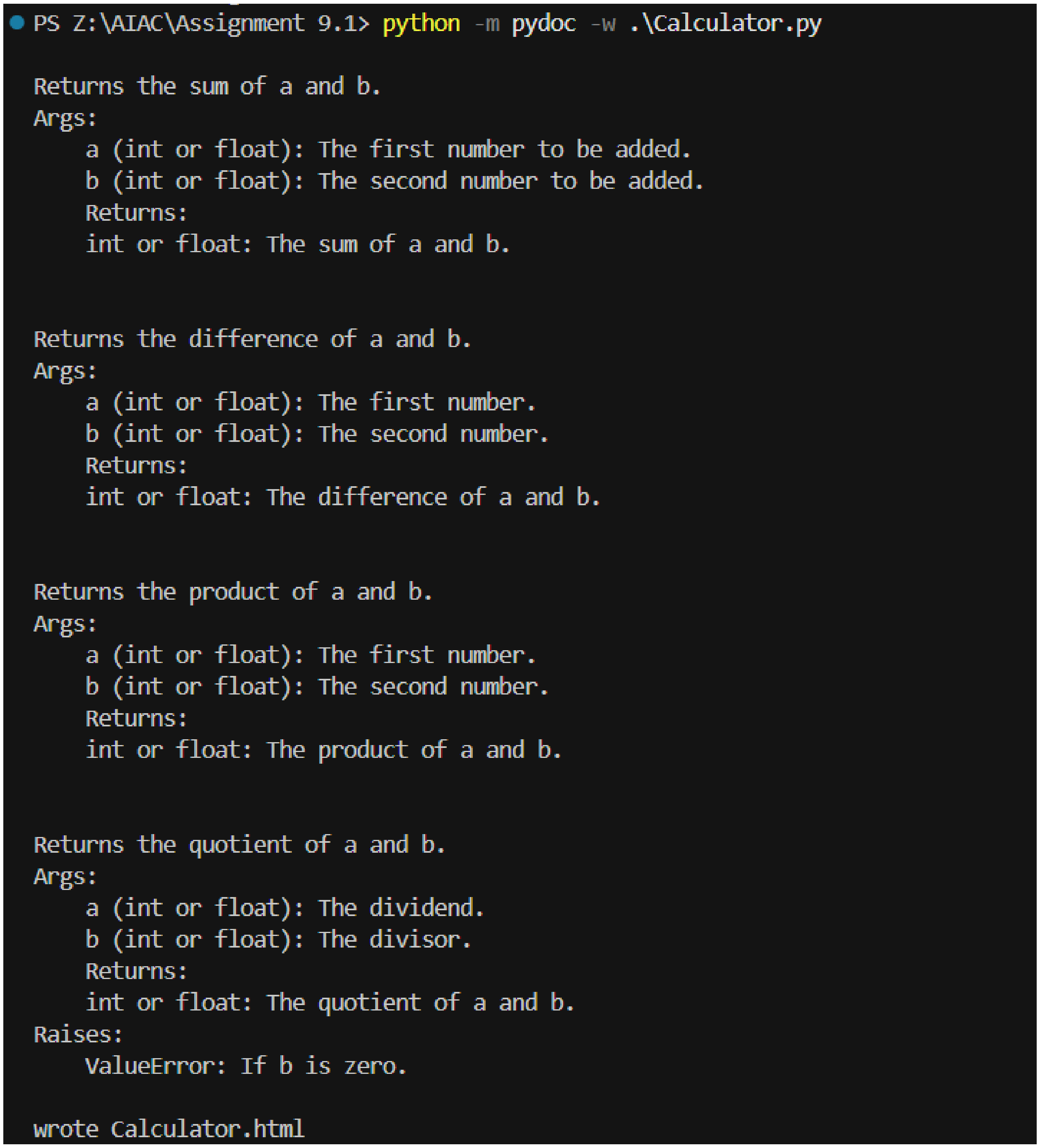
```python
def multiply(a, b):
    """
    Returns the product of a and b.
    Args:
        a (int or float): The first number.
        b (int or float): The second number.
        Returns:
        int or float: The product of a and b.
    """
    return a * b

def divide(a, b):
    """
    Returns the quotient of a and b.
    Args:
        a (int or float): The dividend.
        b (int or float): The divisor.
        Returns:
        int or float: The quotient of a and b.
    Raises:
        ValueError: If b is zero.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b

print(add.__doc__)
print(subtract.__doc__)
print(multiply.__doc__)
print(divide.__doc__)
```



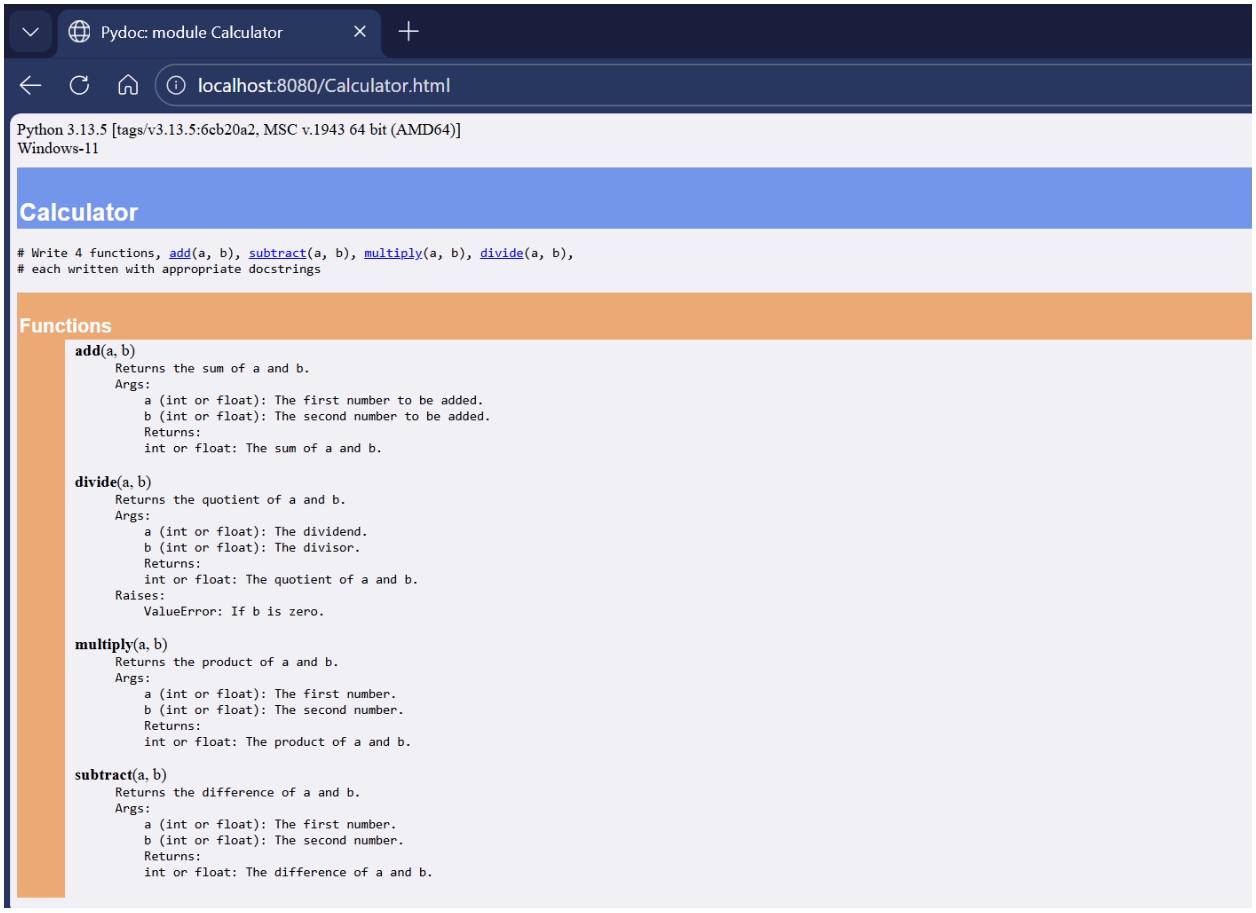
```
PS Z:\AIAC\Assignment 9.1> python -m pydoc .\Calculator.py
```



```
Returns the sum of a and b.
Args:
    a (int or float): The first number to be added.
    b (int or float): The second number to be added.
    Returns:
    int or float: The sum of a and b.


Returns the difference of a and b.
Args:
    a (int or float): The first number.
    b (int or float): The second number.
    Returns:
    int or float: The difference of a and b.


Returns the product of a and b.
Args:
    a (int or float): The first number.
    b (int or float): The second number.
    Returns:
    int or float: The product of a and b.


Returns the quotient of a and b.
Args:
    a (int or float): The dividend.
    b (int or float): The divisor.
    Returns:
    int or float: The quotient of a and b.
Raises:
    ValueError: If b is zero.
```



```
PS Z:\AIAC\Assignment 9.1> python -m pydoc -w .\Calculator.py

Returns the sum of a and b.
Args:
    a (int or float): The first number to be added.
    b (int or float): The second number to be added.
    Returns:
    int or float: The sum of a and b.


Returns the difference of a and b.
Args:
    a (int or float): The first number.
    b (int or float): The second number.
    Returns:
    int or float: The difference of a and b.


Returns the product of a and b.
Args:
    a (int or float): The first number.
    b (int or float): The second number.
    Returns:
    int or float: The product of a and b.


Returns the quotient of a and b.
Args:
    a (int or float): The dividend.
    b (int or float): The divisor.
    Returns:
    int or float: The quotient of a and b.
Raises:
    ValueError: If b is zero.

wrote Calculator.html
```



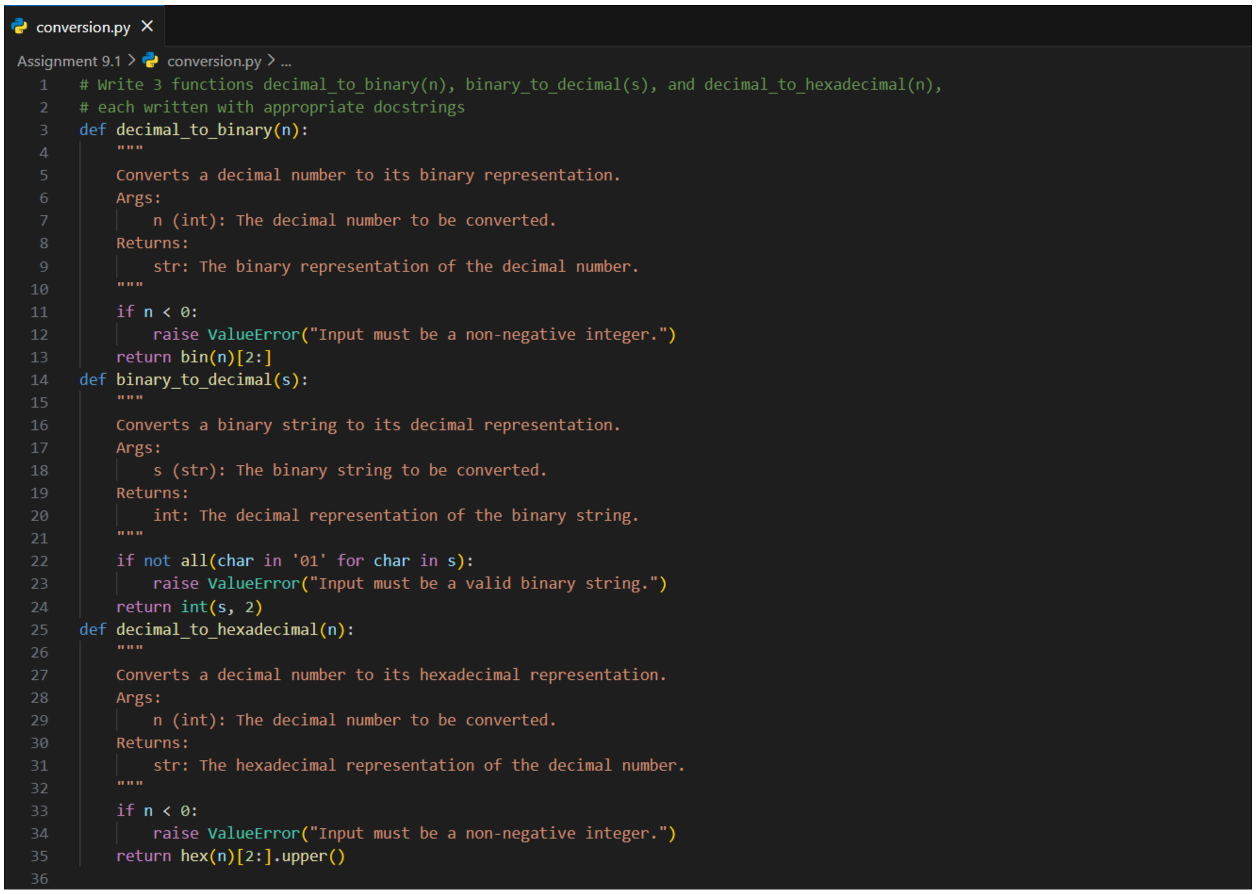
```
Python 3.13.5 [tags/v3.13.5:6cb20a2, MSC v.1943 64 bit (AMD64)]
Windows-11

Calculator

# Write 4 functions, add(a, b), subtract(a, b), multiply(a, b), divide(a, b),
# each written with appropriate docstrings

Functions
add(a, b)
    Returns the sum of a and b.
    Args:
        a (int or float): The first number to be added.
        b (int or float): The second number to be added.
        Returns:
        int or float: The sum of a and b.

divide(a, b)
    Returns the quotient of a and b.
    Args:
        a (int or float): The dividend.
        b (int or float): The divisor.
        Returns:
        int or float: The quotient of a and b.
    Raises:
        ValueError: If b is zero.

multiply(a, b)
    Returns the product of a and b.
    Args:
        a (int or float): The first number.
        b (int or float): The second number.
        Returns:
        int or float: The product of a and b.

subtract(a, b)
    Returns the difference of a and b.
    Args:
        a (int or float): The first number.
        b (int or float): The second number.
        Returns:
        int or float: The difference of a and b.
```

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.

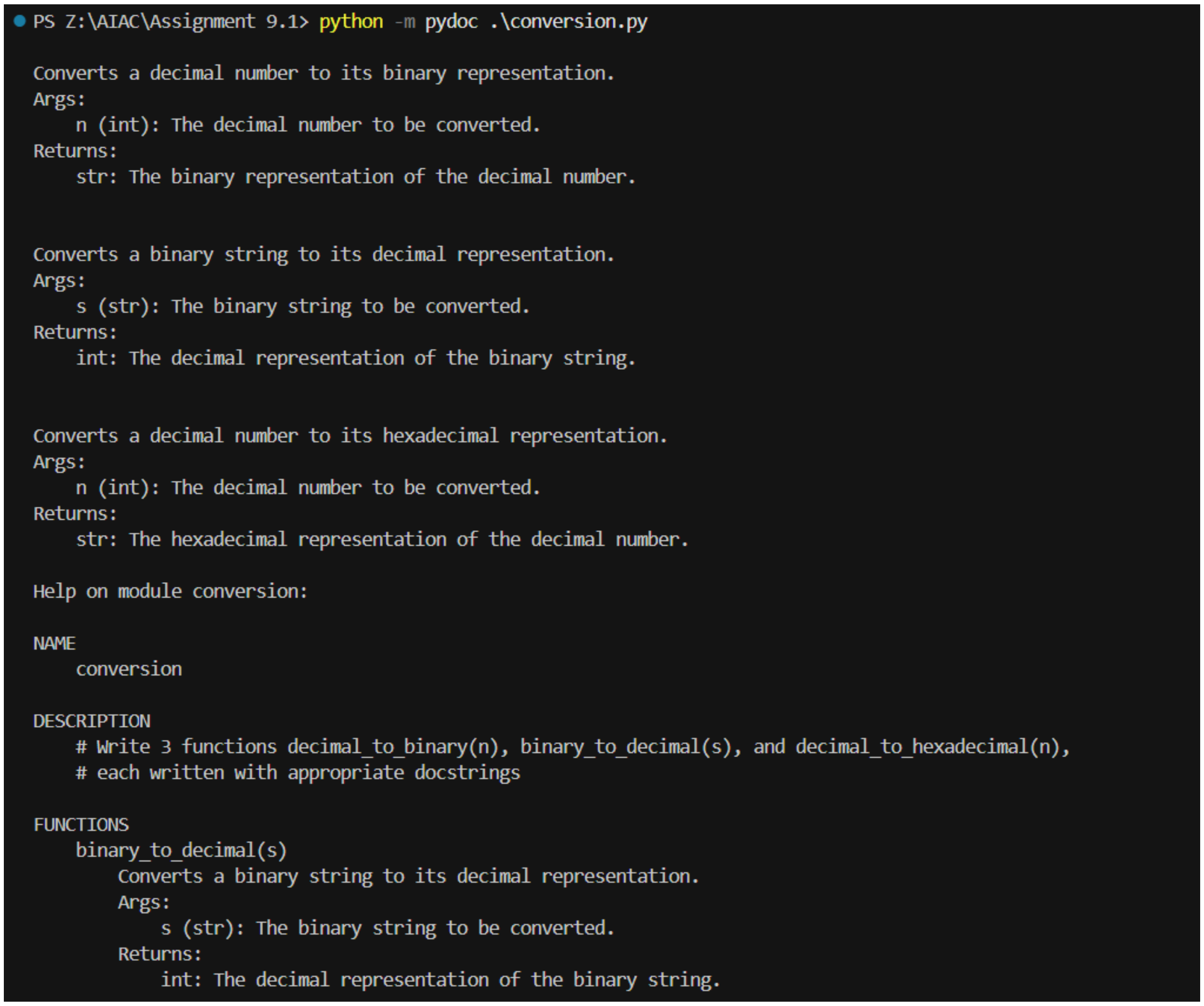
Outputs



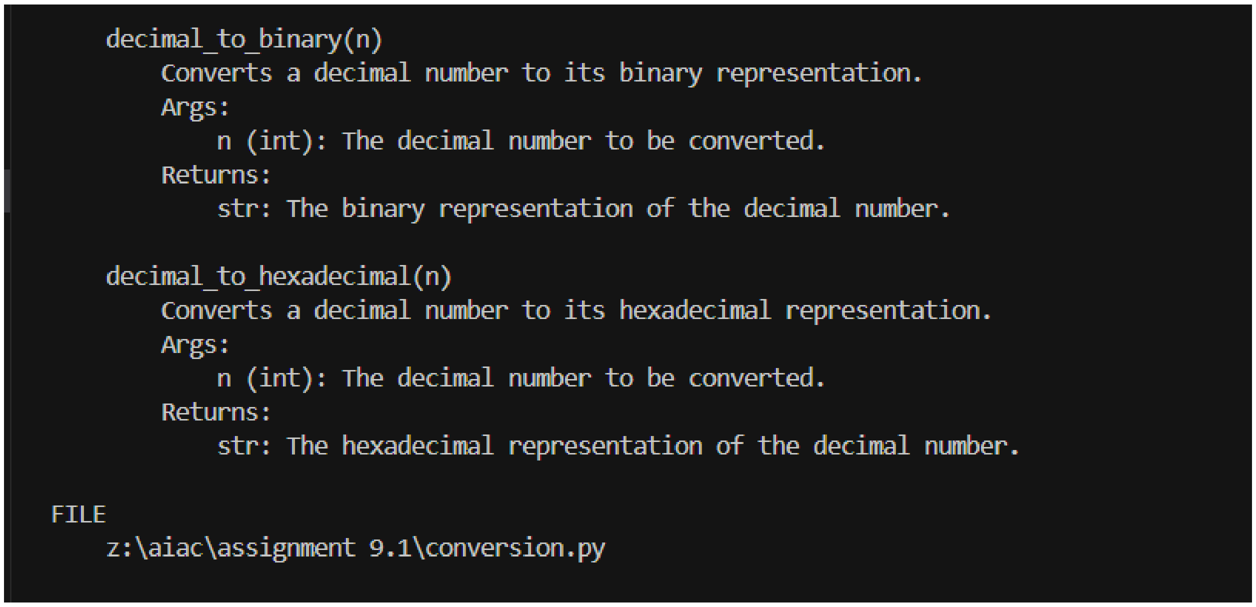
conversion.py

Assignment 9.1 > conversion.py > ...

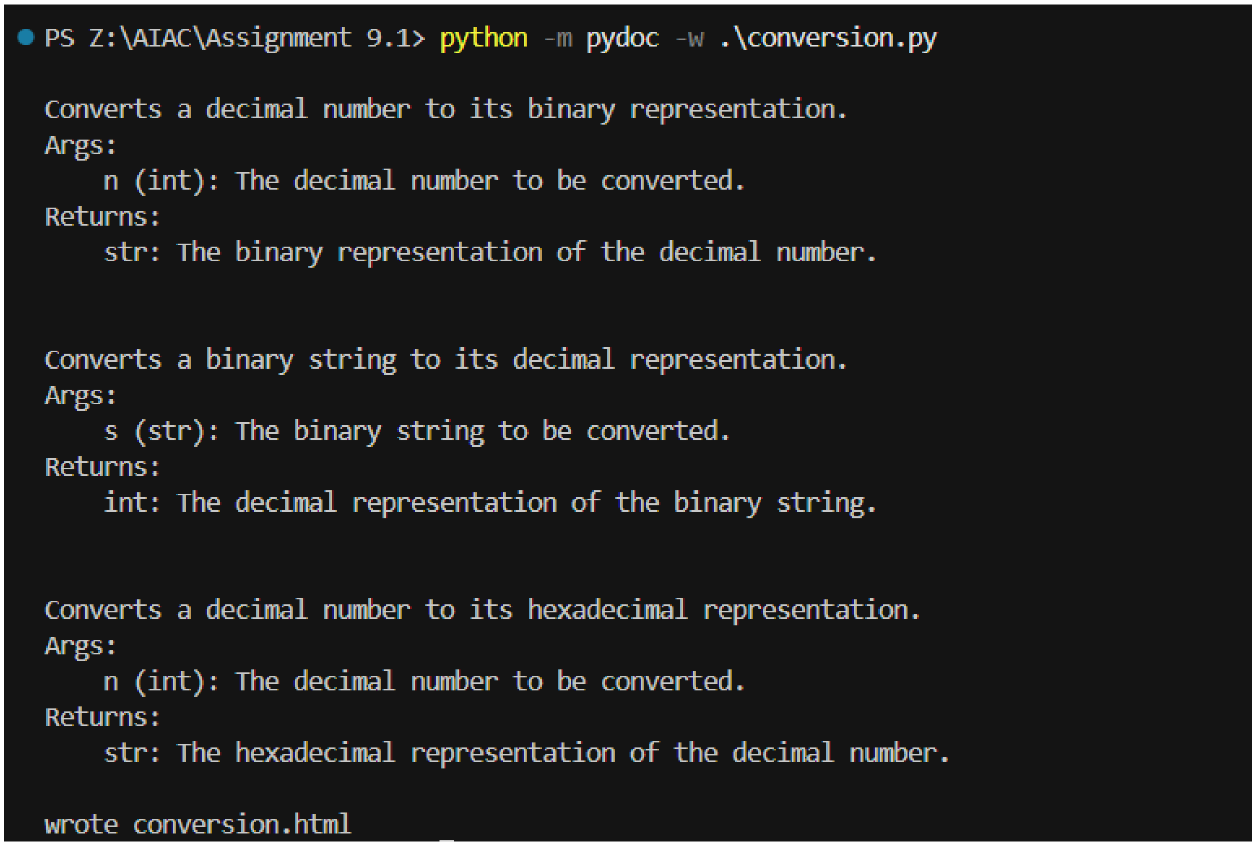
```python
# Write 3 functions decimal_to_binary(n), binary_to_decimal(s), and decimal_to_hexadecimal(n),
# each written with appropriate docstrings
def decimal_to_binary(n):
    """
    Converts a decimal number to its binary representation.
    Args:
        n (int): The decimal number to be converted.
    Returns:
        str: The binary representation of the decimal number.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return bin(n)[2:]
def binary_to_decimal(s):
    """
    Converts a binary string to its decimal representation.
    Args:
        s (str): The binary string to be converted.
    Returns:
        int: The decimal representation of the binary string.
    """
    if not all(char in '01' for char in s):
        raise ValueError("Input must be a valid binary string.")
    return int(s, 2)
def decimal_to_hexadecimal(n):
    """
    Converts a decimal number to its hexadecimal representation.
    Args:
        n (int): The decimal number to be converted.
    Returns:
        str: The hexadecimal representation of the decimal number.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return hex(n)[2:].upper()

```



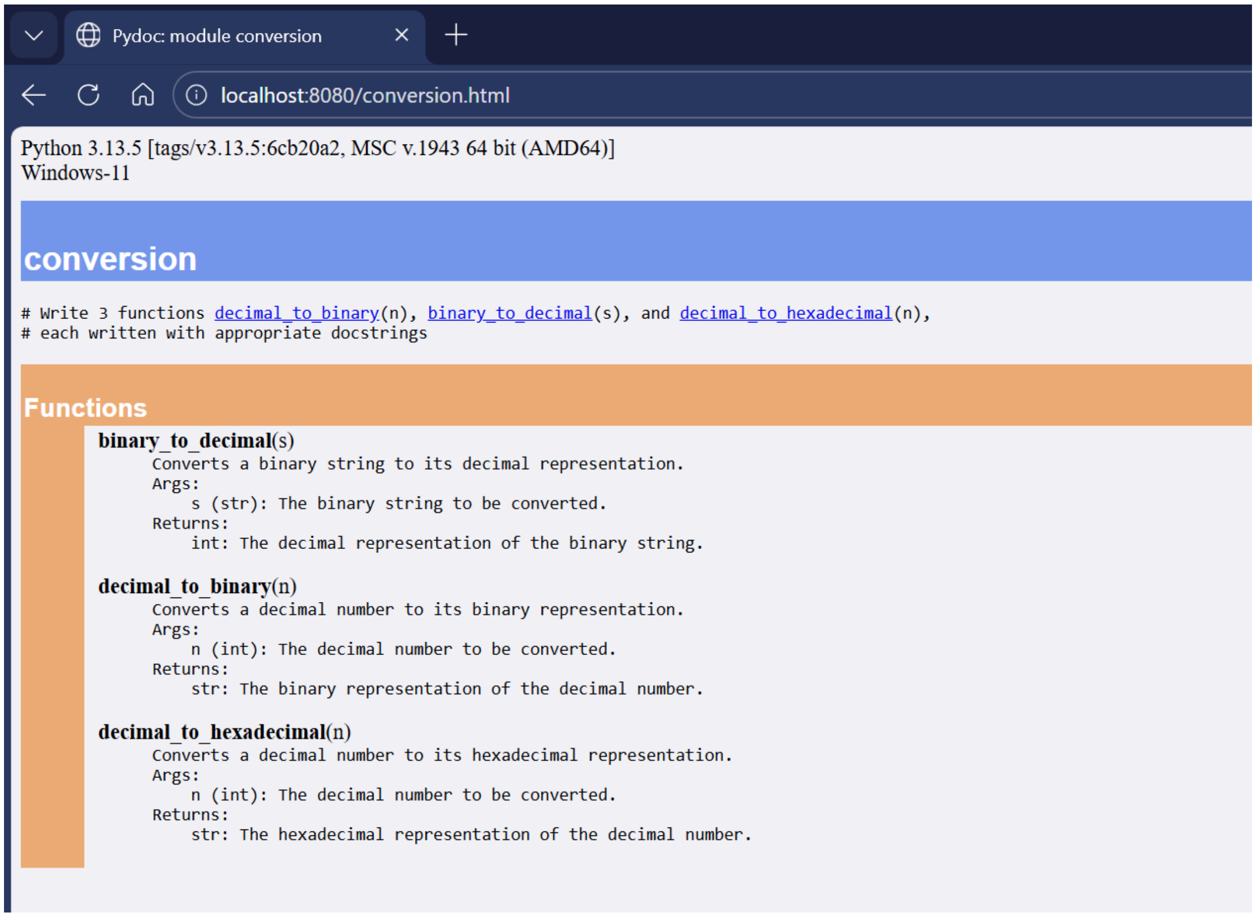
```
PS Z:\AIAC\Assignment 9.1> python -m pydoc .\conversion.py

Converts a decimal number to its binary representation.
Args:
    n (int): The decimal number to be converted.
Returns:
    str: The binary representation of the decimal number.


Converts a binary string to its decimal representation.
Args:
    s (str): The binary string to be converted.
Returns:
    int: The decimal representation of the binary string.


Converts a decimal number to its hexadecimal representation.
Args:
    n (int): The decimal number to be converted.
Returns:
    str: The hexadecimal representation of the decimal number.

Help on module conversion:

NAME
    conversion

DESCRIPTION
    # Write 3 functions decimal_to_binary(n), binary_to_decimal(s), and decimal_to_hexadecimal(n),
    # each written with appropriate docstrings

FUNCTIONS
    binary_to_decimal(s)
        Converts a binary string to its decimal representation.
        Args:
            s (str): The binary string to be converted.
        Returns:
            int: The decimal representation of the binary string.
```



```
    decimal_to_binary(n)
        Converts a decimal number to its binary representation.
        Args:
            n (int): The decimal number to be converted.
        Returns:
            str: The binary representation of the decimal number.

    decimal_to_hexadecimal(n)
        Converts a decimal number to its hexadecimal representation.
        Args:
            n (int): The decimal number to be converted.
        Returns:
            str: The hexadecimal representation of the decimal number.

FILE
    z:\aiac\assignment 9.1\conversion.py
```



```
● PS Z:\AIAC\Assignment 9.1> python -m pydoc -w .\conversion.py

  Converts a decimal number to its binary representation.
  Args:
      n (int): The decimal number to be converted.
  Returns:
      str: The binary representation of the decimal number.


  Converts a binary string to its decimal representation.
  Args:
      s (str): The binary string to be converted.
  Returns:
      int: The decimal representation of the binary string.


  Converts a decimal number to its hexadecimal representation.
  Args:
      n (int): The decimal number to be converted.
  Returns:
      str: The hexadecimal representation of the decimal number.

  wrote conversion.html
```



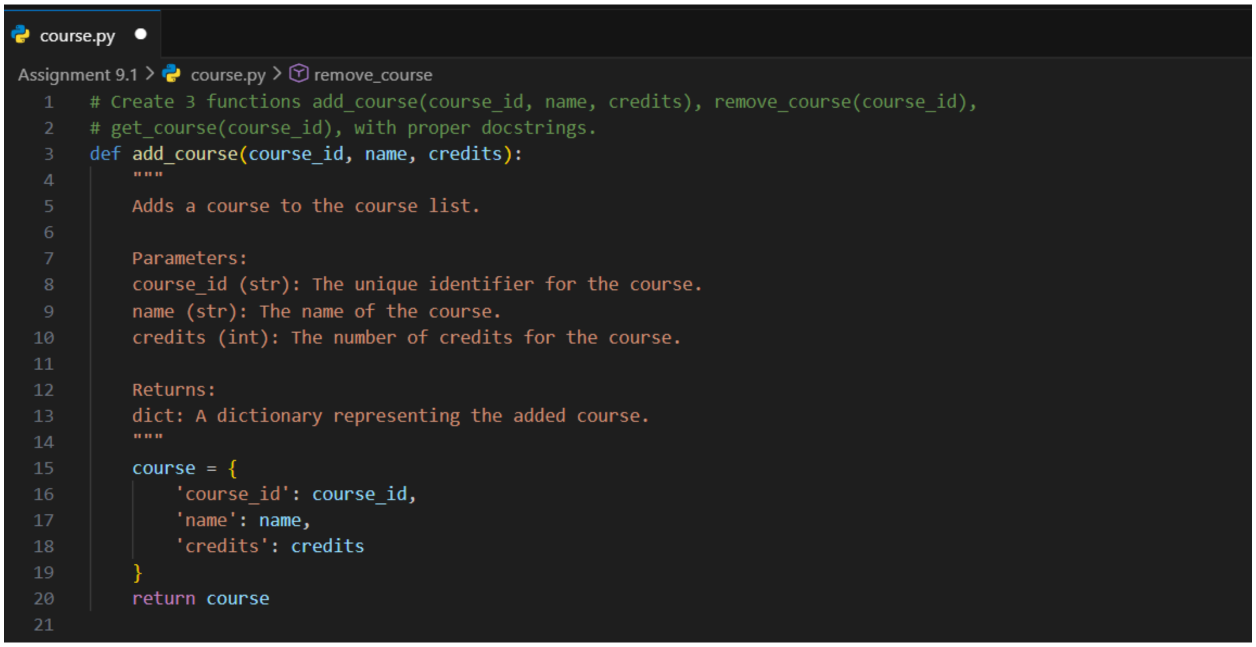
```
Python 3.13.5 [tags/v3.13.5:6cb20a2, MSC v.1943 64 bit (AMD64)]
Windows-11

conversion

# Write 3 functions decimal_to_binary(n), binary_to_decimal(s), and decimal_to_hexadecimal(n),
# each written with appropriate docstrings

Functions
binary_to_decimal(s)
    Converts a binary string to its decimal representation.
    Args:
        s (str): The binary string to be converted.
    Returns:
        int: The decimal representation of the binary string.

decimal_to_binary(n)
    Converts a decimal number to its binary representation.
    Args:
        n (int): The decimal number to be converted.
    Returns:
        str: The binary representation of the decimal number.

decimal_to_hexadecimal(n)
    Converts a decimal number to its hexadecimal representation.
    Args:
        n (int): The decimal number to be converted.
    Returns:
        str: The hexadecimal representation of the decimal number.
```

Problem 5: Course Management Module

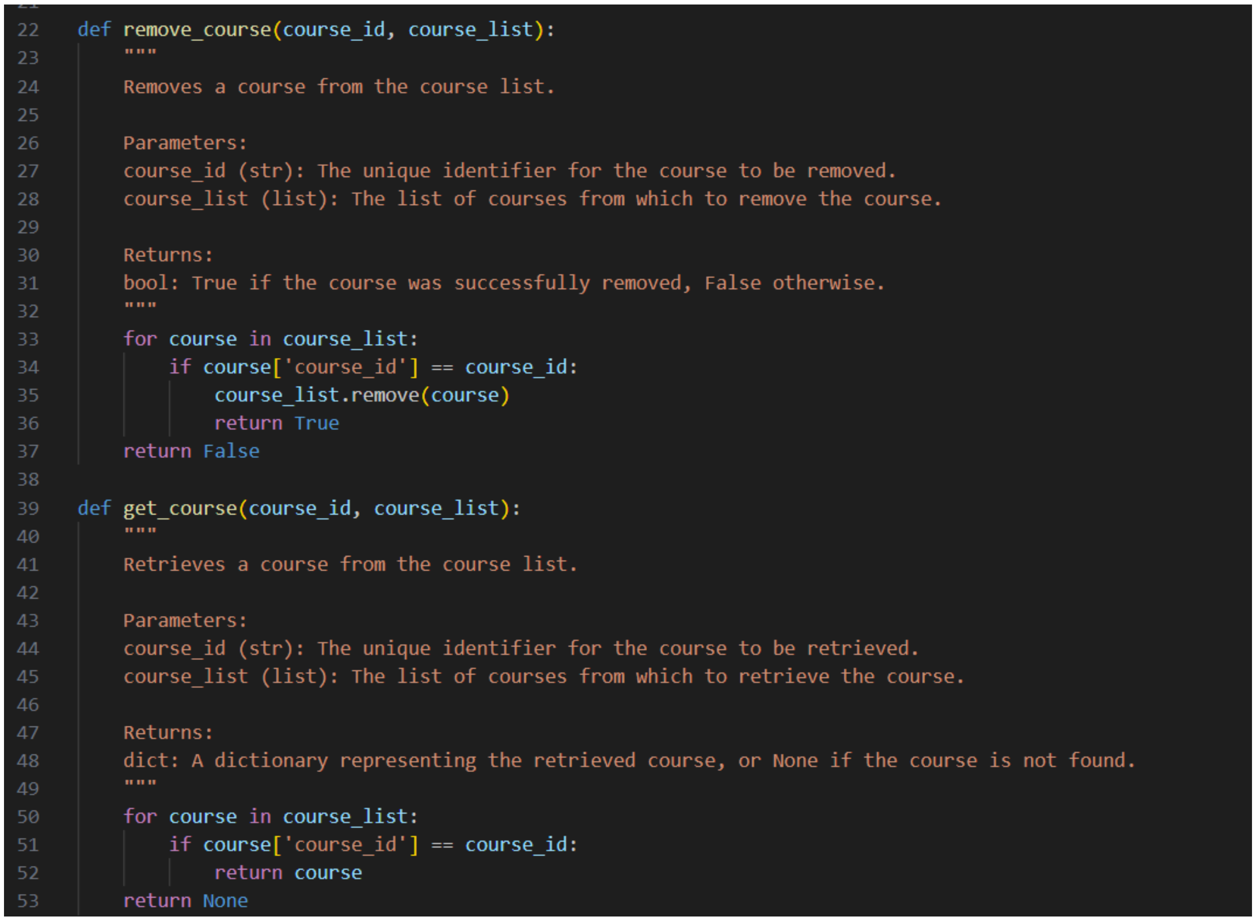
Task:

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.

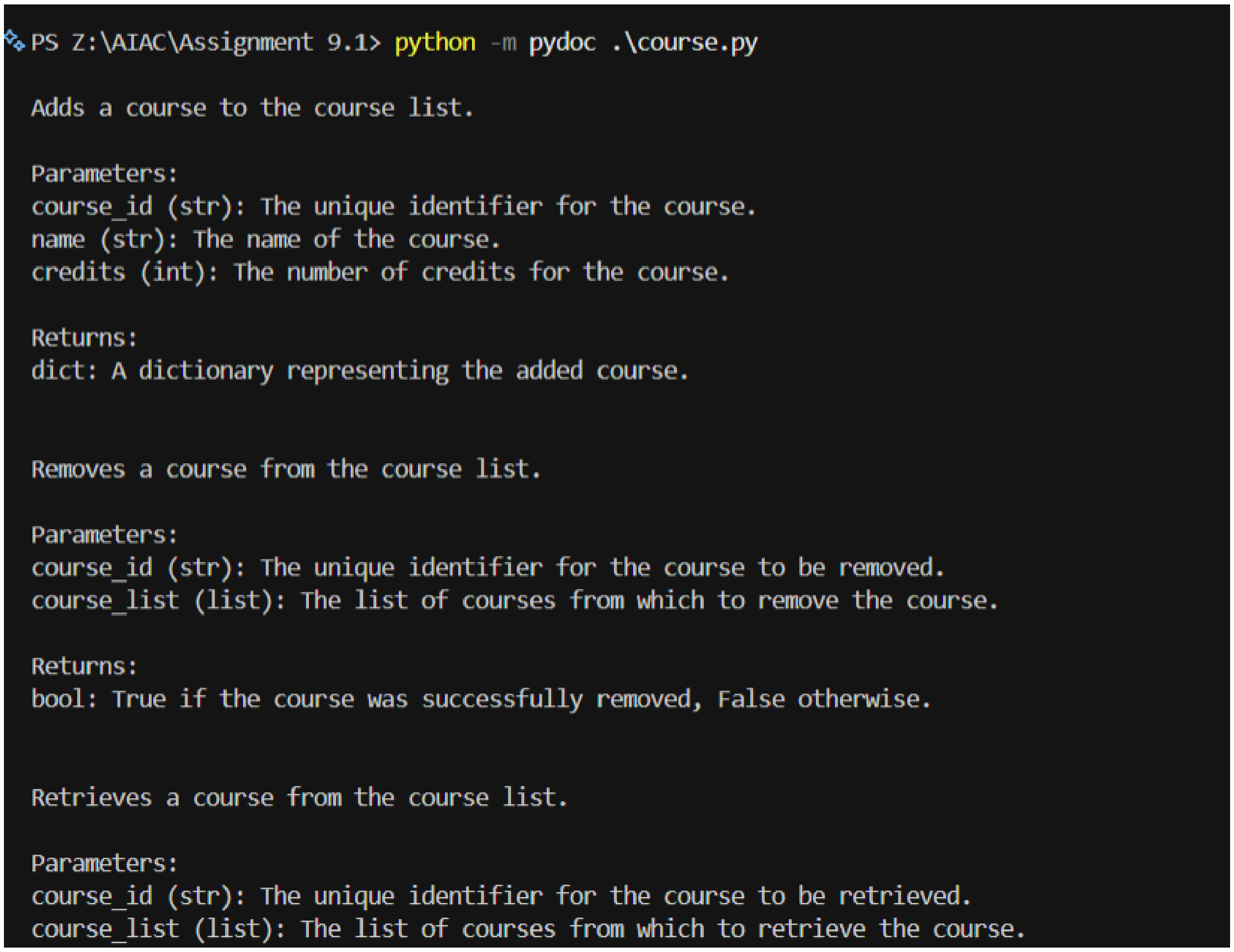
Outputs



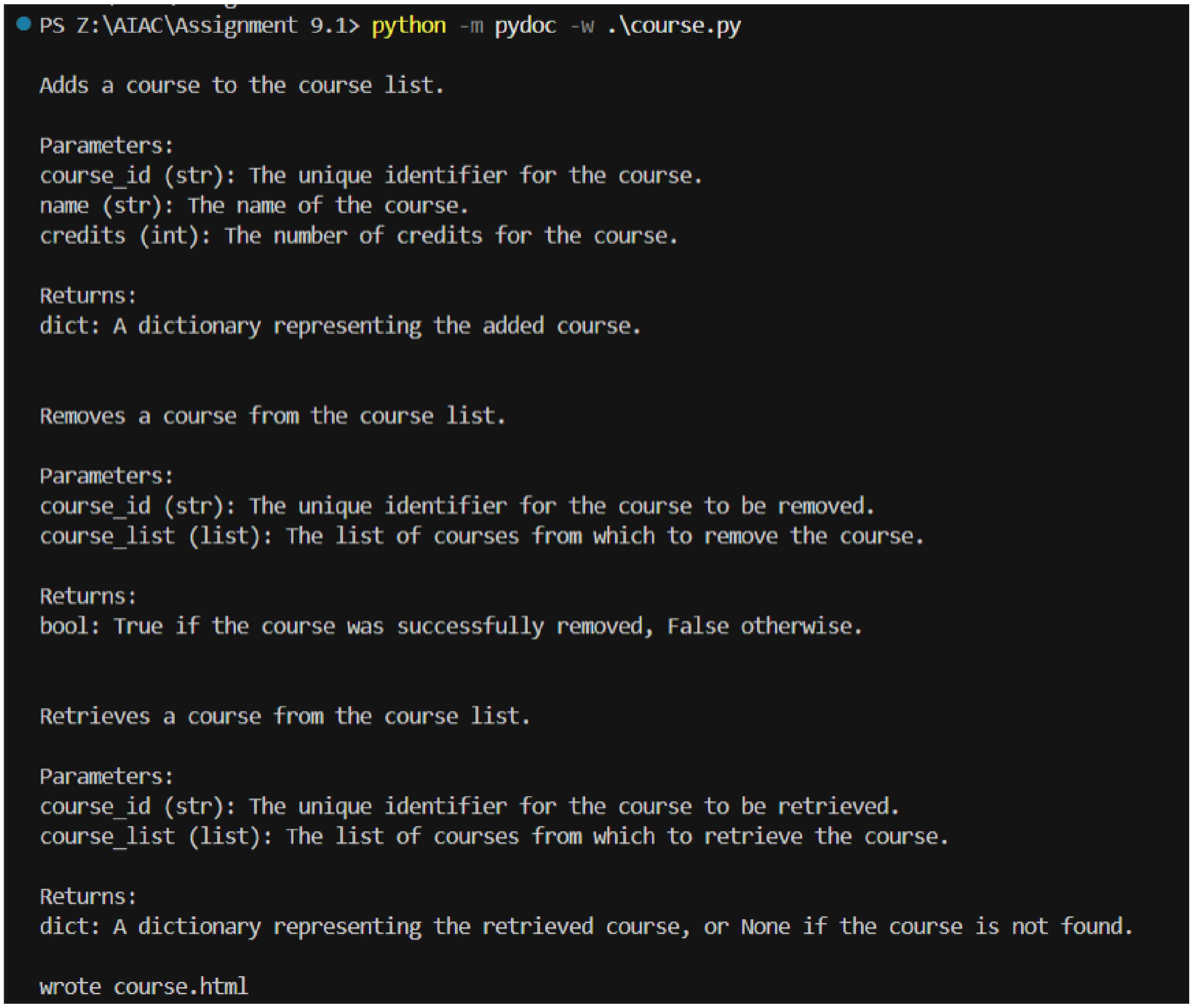
```python
# Create 3 functions add_course(course_id, name, credits), remove_course(course_id),
# get_course(course_id), with proper docstrings.
def add_course(course_id, name, credits):
    """
    Adds a course to the course list.

    Parameters:
    course_id (str): The unique identifier for the course.
    name (str): The name of the course.
    credits (int): The number of credits for the course.

    Returns:
    dict: A dictionary representing the added course.
    """
    course = {
        'course_id': course_id,
        'name': name,
        'credits': credits
    }
    return course

```



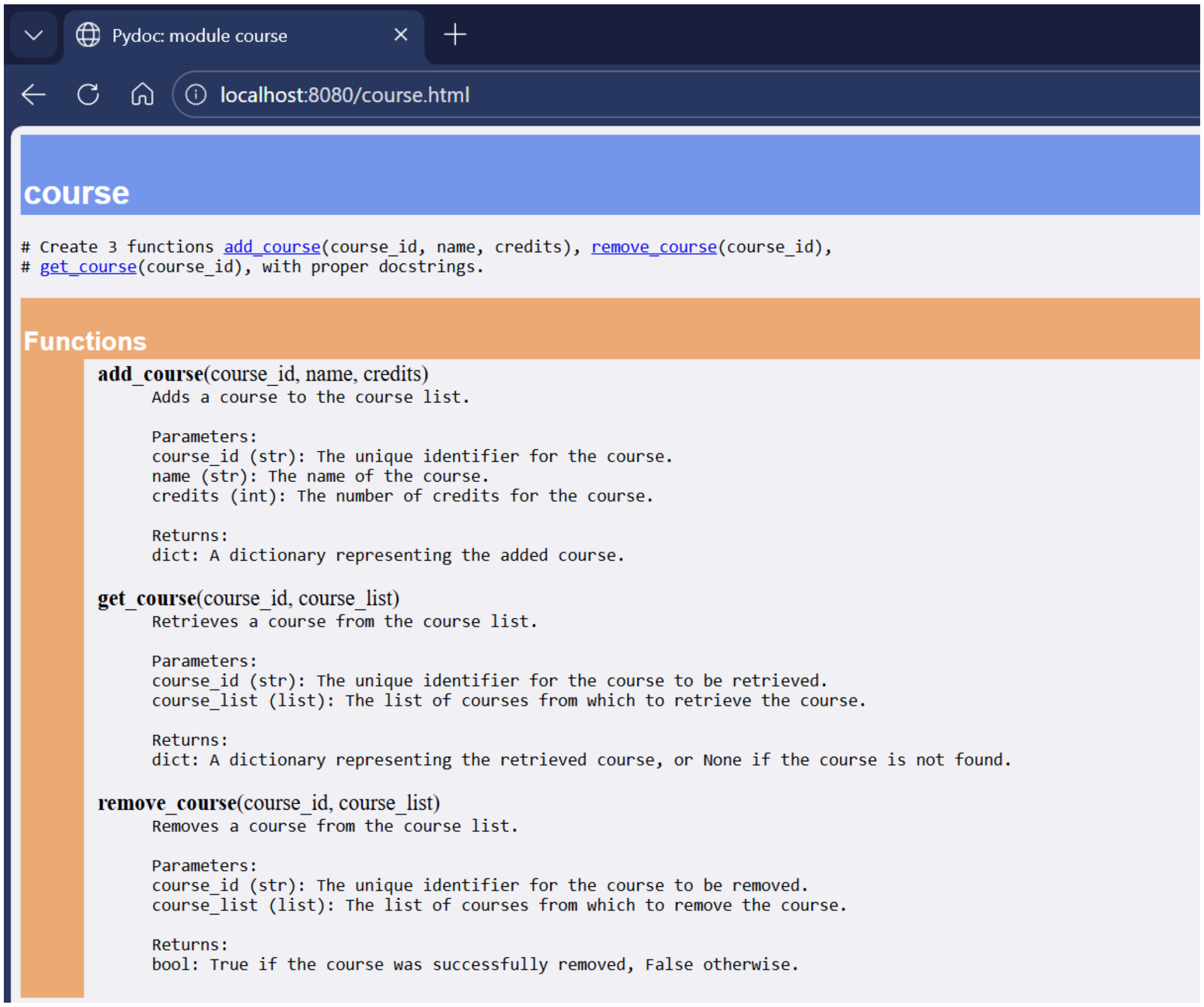
```python
def remove_course(course_id, course_list):
    """
    Removes a course from the course list.

    Parameters:
    course_id (str): The unique identifier for the course to be removed.
    course_list (list): The list of courses from which to remove the course.

    Returns:
    bool: True if the course was successfully removed, False otherwise.
    """
    for course in course_list:
        if course['course_id'] == course_id:
            course_list.remove(course)
            return True
    return False

def get_course(course_id, course_list):
    """
    Retrieves a course from the course list.

    Parameters:
    course_id (str): The unique identifier for the course to be retrieved.
    course_list (list): The list of courses from which to retrieve the course.

    Returns:
    dict: A dictionary representing the retrieved course, or None if the course is not found.
    """
    for course in course_list:
        if course['course_id'] == course_id:
            return course
    return None
```



```
PS Z:\AIAC\Assignment 9.1> python -m pydoc .\course.py

Adds a course to the course list.

Parameters:
course_id (str): The unique identifier for the course.
name (str): The name of the course.
credits (int): The number of credits for the course.

Returns:
dict: A dictionary representing the added course.


Removes a course from the course list.

Parameters:
course_id (str): The unique identifier for the course to be removed.
course_list (list): The list of courses from which to remove the course.

Returns:
bool: True if the course was successfully removed, False otherwise.


Retrieves a course from the course list.

Parameters:
course_id (str): The unique identifier for the course to be retrieved.
course_list (list): The list of courses from which to retrieve the course.
```



```
PS Z:\AIAC\Assignment 9.1> python -m pydoc -w .\course.py

Adds a course to the course list.

Parameters:
course_id (str): The unique identifier for the course.
name (str): The name of the course.
credits (int): The number of credits for the course.

Returns:
dict: A dictionary representing the added course.


Removes a course from the course list.

Parameters:
course_id (str): The unique identifier for the course to be removed.
course_list (list): The list of courses from which to remove the course.

Returns:
bool: True if the course was successfully removed, False otherwise.


Retrieves a course from the course list.

Parameters:
course_id (str): The unique identifier for the course to be retrieved.
course_list (list): The list of courses from which to retrieve the course.

Returns:
dict: A dictionary representing the retrieved course, or None if the course is not found.

wrote course.html
```


Pydoc: module course

localhost:8080/course.html

course

```
# Create 3 functions add_course(course_id, name, credits), remove_course(course_id),
# get_course(course_id), with proper docstrings.
```

Functions

add_course(course_id, name, credits)

```
Adds a course to the course list.

Parameters:
course_id (str): The unique identifier for the course.
name (str): The name of the course.
credits (int): The number of credits for the course.

Returns:
dict: A dictionary representing the added course.
```

get_course(course_id, course_list)

```
Retrieves a course from the course list.

Parameters:
course_id (str): The unique identifier for the course to be retrieved.
course_list (list): The list of courses from which to retrieve the course.

Returns:
dict: A dictionary representing the retrieved course, or None if the course is not found.
```

remove_course(course_id, course_list)

```
Removes a course from the course list.

Parameters:
course_id (str): The unique identifier for the course to be removed.
course_list (list): The list of courses from which to remove the course.

Returns:
bool: True if the course was successfully removed, False otherwise.
```