T. VIKAS

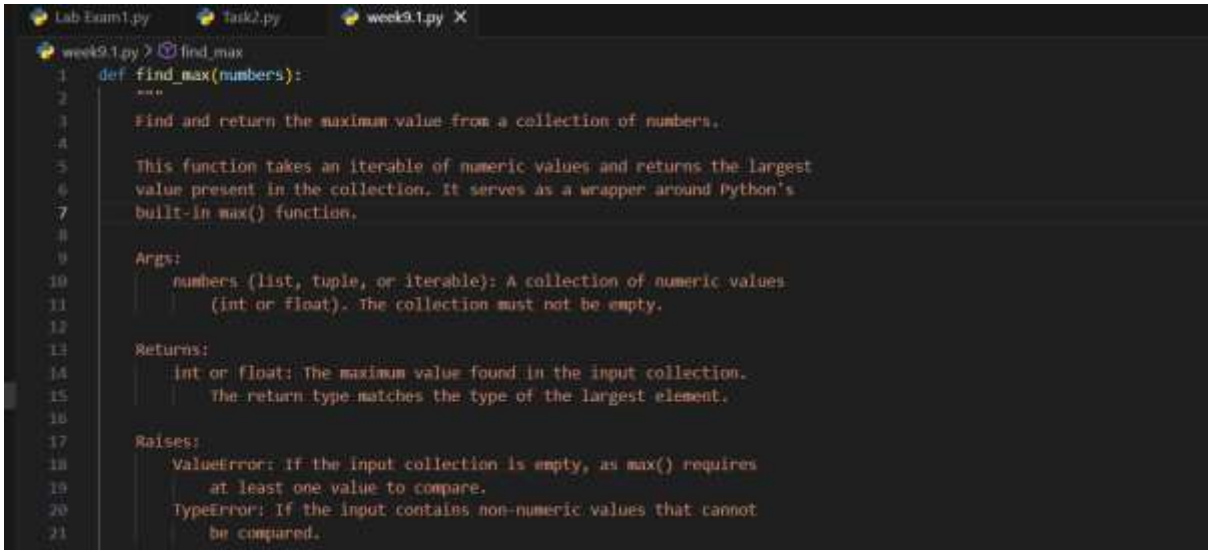
2303A52281

BATCH 43

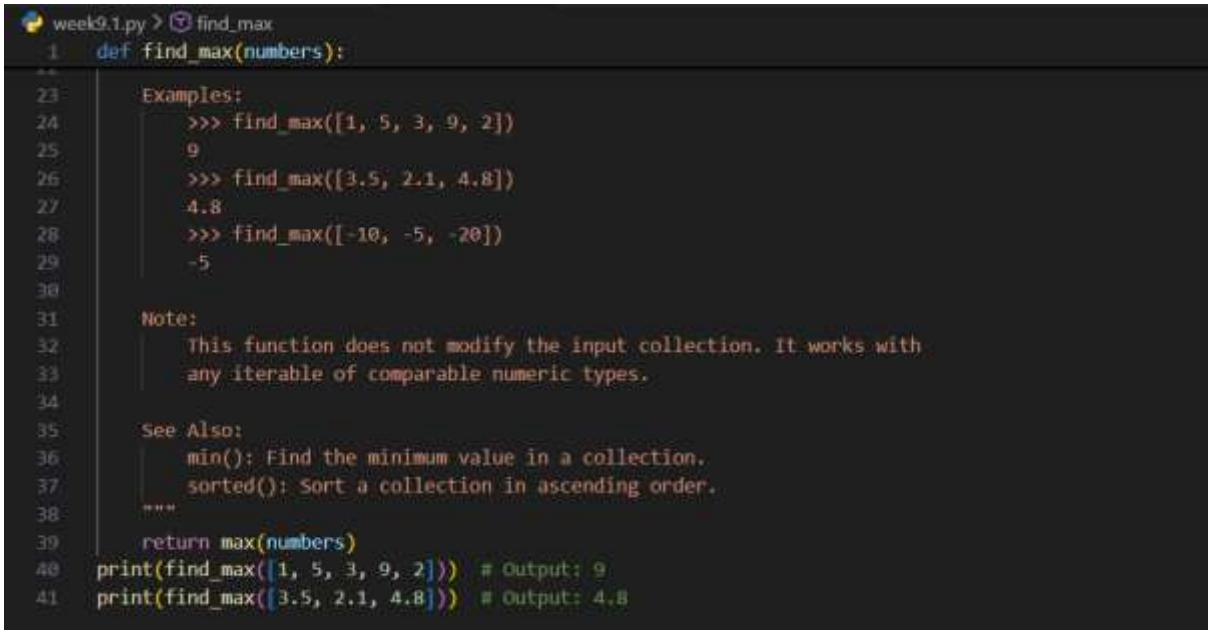
WEEK 9.1

TASK 1:

CODE:



```
def find_max(numbers):
    """
    Find and return the maximum value from a collection of numbers.

    This function takes an iterable of numeric values and returns the largest
    value present in the collection. It serves as a wrapper around Python's
    built-in max() function.

    Args:
        numbers (list, tuple, or iterable): A collection of numeric values
            (int or float). The collection must not be empty.

    Returns:
        int or float: The maximum value found in the input collection.
            The return type matches the type of the largest element.

    Raises:
        ValueError: If the input collection is empty, as max() requires
            at least one value to compare.
        TypeError: If the input contains non-numeric values that cannot
            be compared.
```



```
def find_max(numbers):
    Examples:
        >>> find_max([1, 5, 3, 9, 2])
        9
        >>> find_max([3.5, 2.1, 4.8])
        4.8
        >>> find_max([-10, -5, -20])
        -5

    Note:
        This function does not modify the input collection. It works with
        any iterable of comparable numeric types.

    See Also:
        min(): Find the minimum value in a collection.
        sorted(): Sort a collection in ascending order.
    """
    return max(numbers)
print(find_max([1, 5, 3, 9, 2]))  # Output: 9
print(find_max([3.5, 2.1, 4.8]))  # Output: 4.8
```

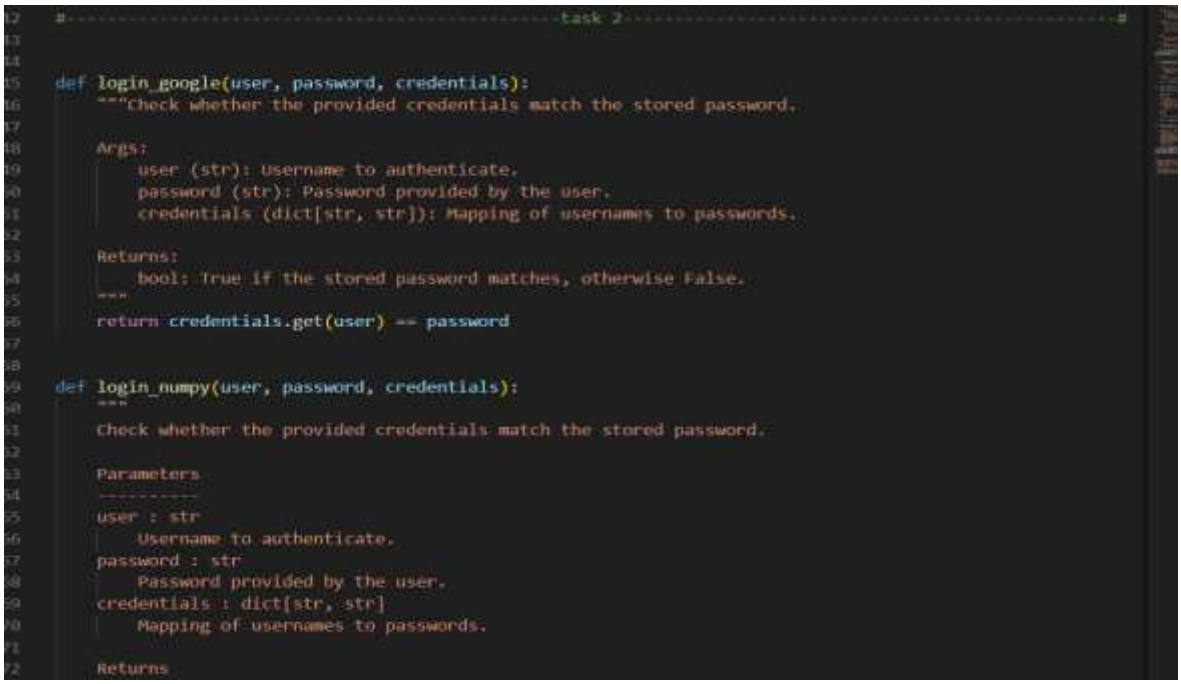
OUTPUT:

```
9
4.8
```

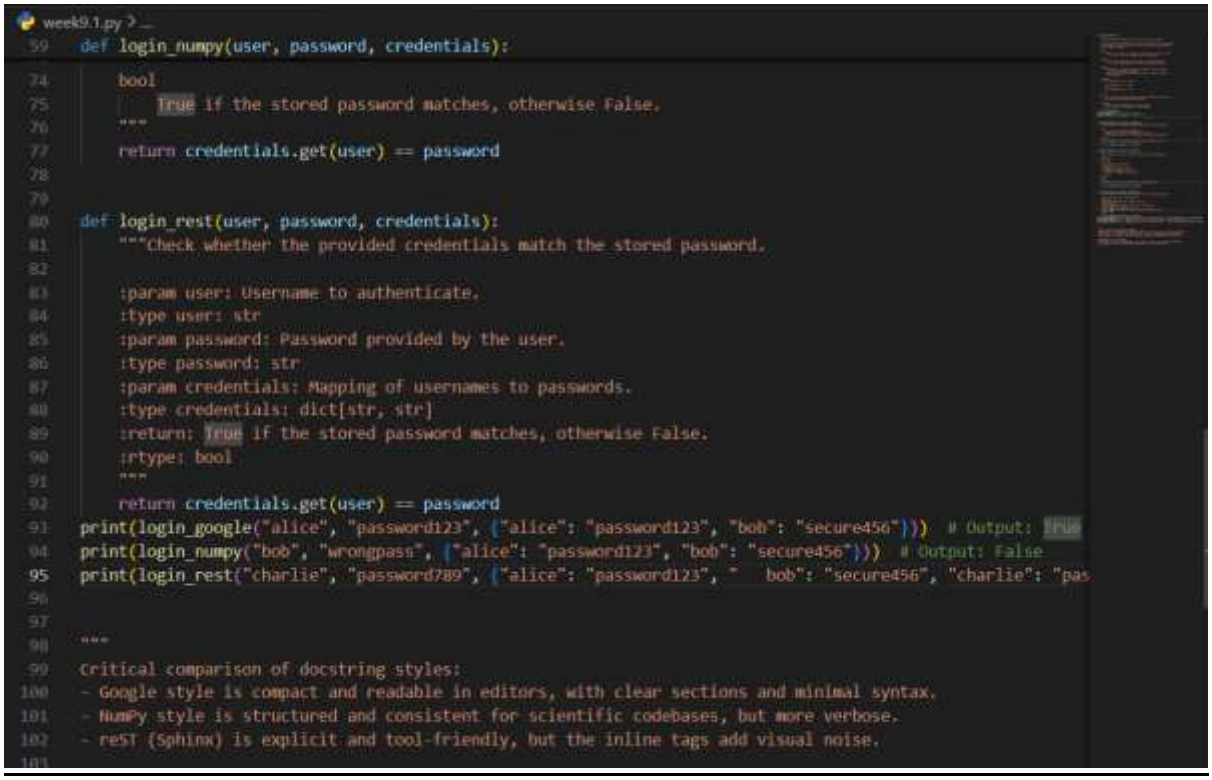
Summary:

This defines find_max(numbers), a simple wrapper around Python's max() that returns the largest value in a non-empty numeric iterable. It includes a detailed docstring, then demonstrates usage by printing the maximum of two example lists (one ints, one floats).

Problem 2:



```
#-----------------------------------------------task 2-----------------------------------------------#


def login_google(user, password, credentials):
    """Check whether the provided credentials match the stored password.

    Args:
        user (str): Username to authenticate.
        password (str): Password provided by the user.
        credentials (dict[str, str]): Mapping of usernames to passwords.

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


def login_numpy(user, password, credentials):
    """
    Check whether the provided credentials match the stored password.

    Parameters
    ----------
    user : str
        Username to authenticate.
    password : str
        Password provided by the user.
    credentials : dict[str, str]
        Mapping of usernames to passwords.

    Returns
```



```
week9.1.py > ...
59  def login_numpy(user, password, credentials):
74      bool
75          True if the stored password matches, otherwise False.
76      """
77      return credentials.get(user) == password
78
79
80  def login_rest(user, password, credentials):
81      """Check whether the provided credentials match the stored password.
82
83      :param user: Username to authenticate.
84      :type user: str
85      :param password: Password provided by the user.
86      :type password: str
87      :param credentials: Mapping of usernames to passwords.
88      :type credentials: dict[str, str]
89      :return: True if the stored password matches, otherwise False.
90      :rtype: bool
91      """
92      return credentials.get(user) == password
93  print(login_google("alice", "password123", {"alice": "password123", "bob": "secure456"}))  # Output: True
94  print(login_numpy("bob", "wrongpass", {"alice": "password123", "bob": "secure456"}))  # Output: False
95  print(login_rest("charlie", "password789", {"alice": "password123", "    bob": "secure456", "charlie": "pas
96
97
98  """
99  Critical comparison of docstring styles:
100 - Google style is compact and readable in editors, with clear sections and minimal syntax.
101 - NumPy style is structured and consistent for scientific codebases, but more verbose.
102 - reST (Sphinx) is explicit and tool-friendly, but the inline tags add visual noise.
```

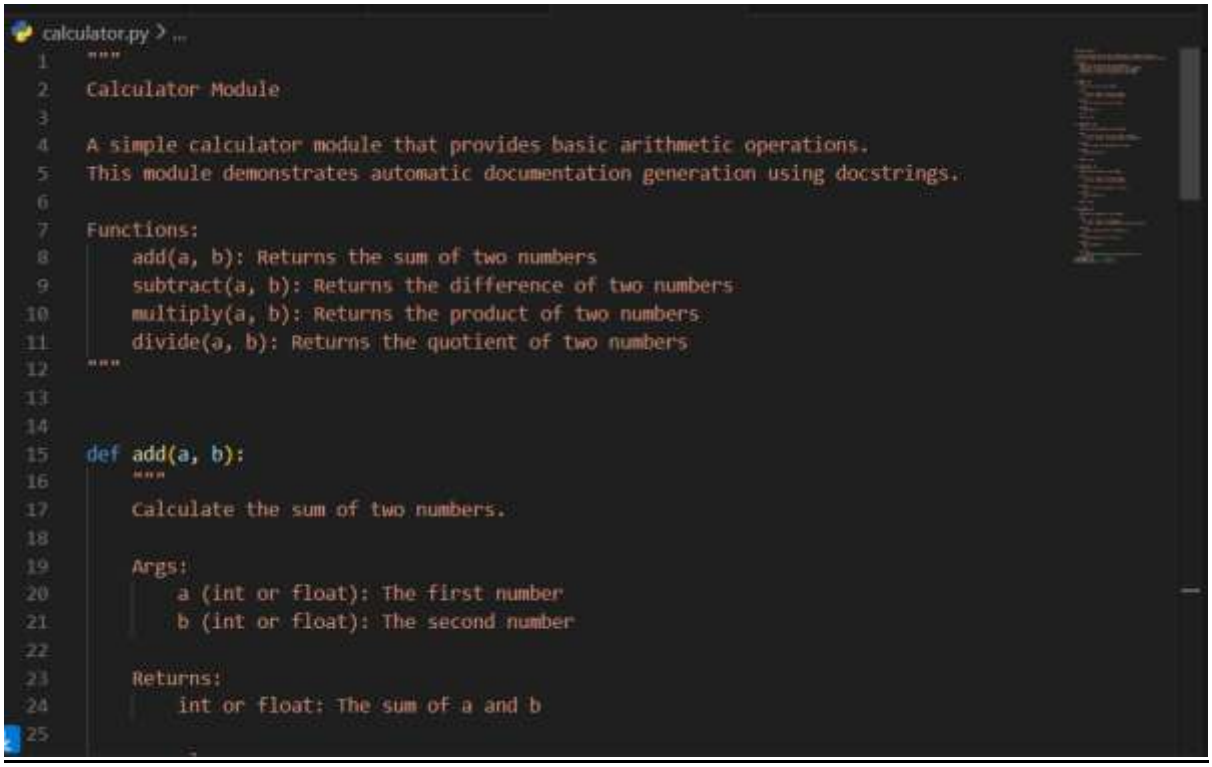
Output:

```
True
True
False
True
PS C:\Users\vikas\Downloads\AI Assist Coding>
```

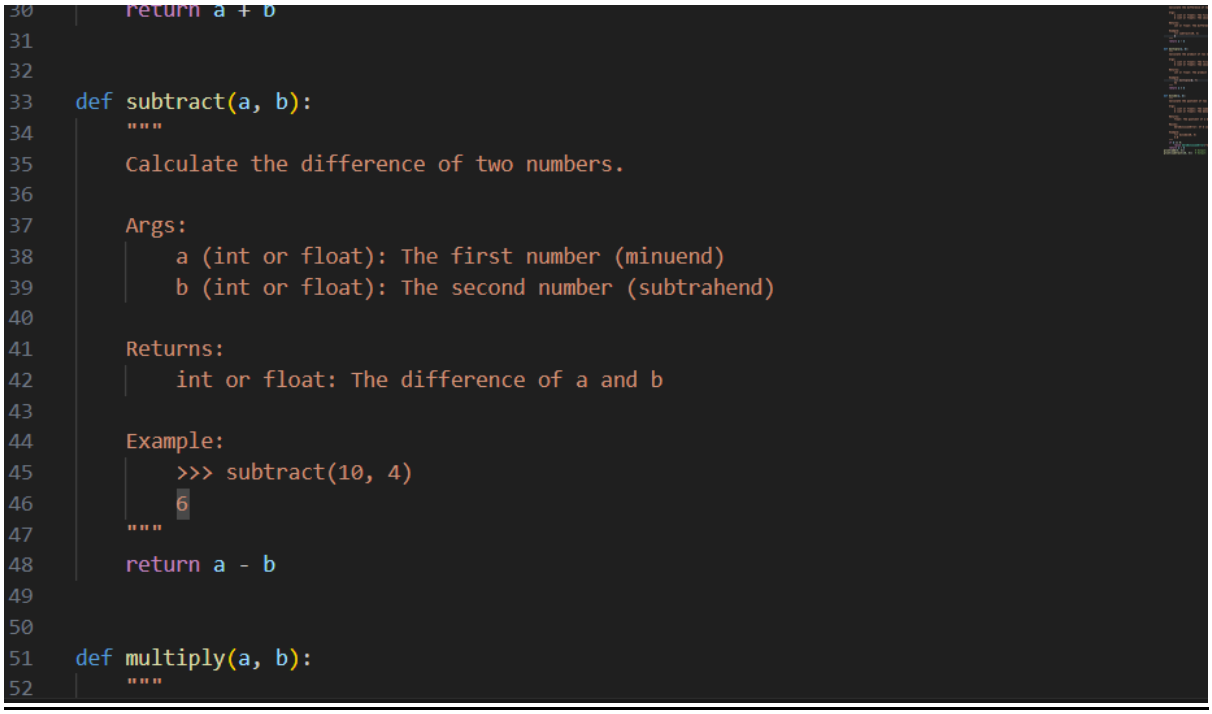
Summary:

This file defines find_max() with a detailed docstring and example prints. It then adds three login functions showing Google, NumPy, and reST docstring formats, followed by example calls and a short comparison plus recommendation that Google style is best for onboarding because it's the most readable with minimal formatting overhead.

Problem 3:



```
calculator.py > ...
"""
Calculator Module

A simple calculator module that provides basic arithmetic operations.
This module demonstrates automatic documentation generation using docstrings.

Functions:
    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
"""


def add(a, b):
    """
    Calculate the sum of two numbers.

    Args:
        a (int or float): The first number
        b (int or float): The second number

    Returns:
        int or float: The sum of a and b
```



```
    return a + b


def subtract(a, b):
    """
    Calculate the difference of two numbers.

    Args:
        a (int or float): The first number (minuend)
        b (int or float): The second number (subtrahend)

    Returns:
        int or float: The difference of a and b

    Example:
        >>> subtract(10, 4)
        6
    """
    return a - b


def multiply(a, b):
    """
```

```python
def multiply(a, b):
    """
    Calculate the product of two numbers.

    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

    Example:
        >>> multiply(6, 7)
        42
    """
    return a * b


def divide(a, b):
    """
    Calculate the quotient of two numbers.

    Args:
        a (int or float): The numerator
        b (int or float): The denominator (must not be zero)

    Returns:
        float: The quotient of a divided by b

    Raises:
        ZeroDivisionError: If b is zero

    Example:
        >>> divide(20, 4)
        5.0
    """
    if b == 0:
        raise ZeroDivisionError("Cannot divide by zero")
    return a / b
print(add(5, 3))        # Output: 8
print(subtract(10, 4))  # Output: 6
```

Output:

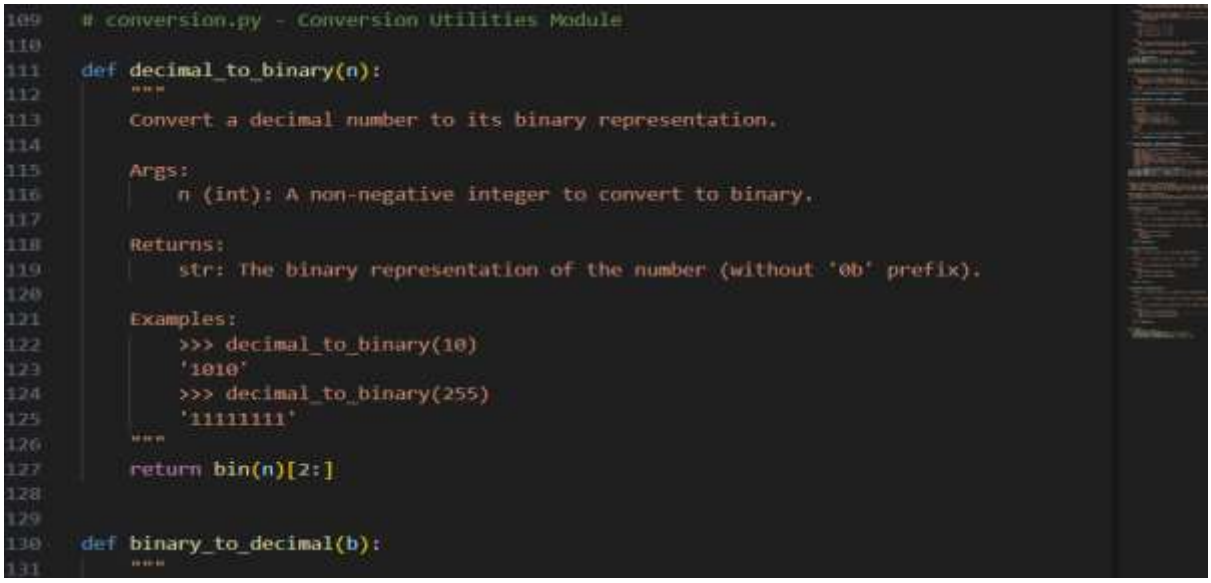
```
Could not find platform independent libraries <prefix>
8
6
PS C:\Users\vikas\Downloads\AI Assist Coding>
```

Summary:

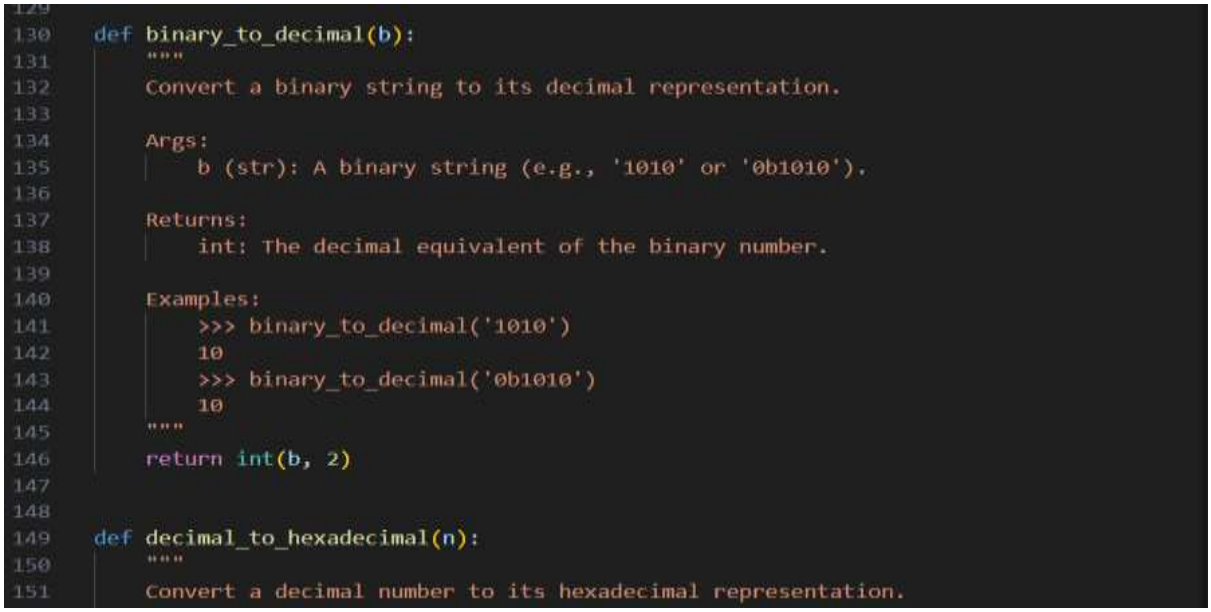
This module defines a simple calculator with add, subtract, multiply, and divide, each documented with clear docstrings and examples. It also includes a top-level module docstring describing the functions, and prints sample outputs for add(5, 3) and subtract(10, 4) when run.

Problem 4:

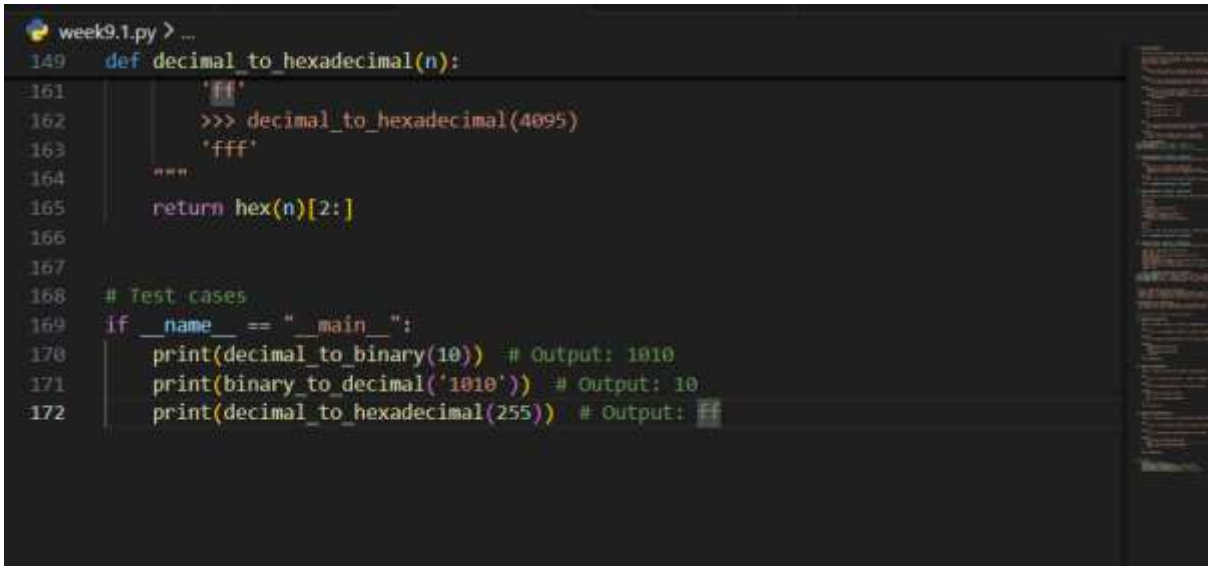
Code:



```python
# conversion.py - Conversion Utilities Module

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

    Args:
        n (int): A non-negative integer to convert to binary.

    Returns:
        str: The binary representation of the number (without '0b' prefix).

    Examples:
        >>> decimal_to_binary(10)
        '1010'
        >>> decimal_to_binary(255)
        '11111111'
    """
    return bin(n)[2:]


def binary_to_decimal(b):
    """
```



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

    Args:
        b (str): A binary string (e.g., '1010' or '0b1010').

    Returns:
        int: The decimal equivalent of the binary number.

    Examples:
        >>> binary_to_decimal('1010')
        10
        >>> binary_to_decimal('0b1010')
        10
    """
    return int(b, 2)


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



Output:

```
1010
10
ff
```

Summary:

This section defines three conversion helpers: decimal_to_binary, binary_to_decimal, and decimal_to_hexadecimal, each with docstrings and examples. The __main__ block runs simple test prints to show the conversions working.

Problem 5:

Code:

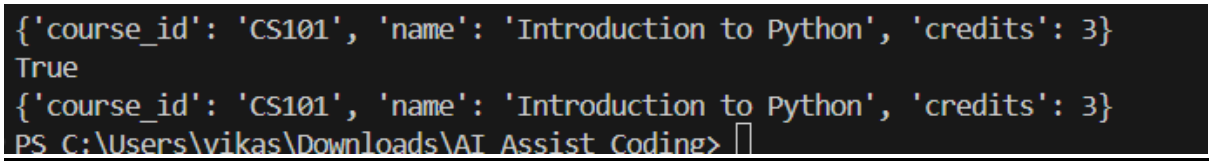
```python
# course.py - Course Management Module

def add_course(course_id, name, credits):
    """
    Add a new course to the course registry.

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

    Returns:
        dict: The course object that was added.

    Examples:
        >>> add_course('CS101', 'Introduction to Python', 3)
        {'course_id': 'CS101', 'name': 'Introduction to Python', 'credits': 3}
    """
    return {'course_id': course_id, 'name': name, 'credits': credits}

```

week9.1.py > ...

```python
def remove_course(course_id):
    """
    Remove a course from the course registry.

    Args:
        course_id (str): Unique identifier of the course to remove.

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

    Examples:
        >>> remove_course('CS101')
        True
    """
    return True


def get_course(course_id):
    """
    Retrieve course information by course ID.
```

```python
    Returns:
        dict or None: Course object if found, else None.

    Examples:
        >>> get_course('CS101')
        {'course_id': 'CS101', 'name': 'Introduction to Python', 'credits': 3}
    """
    return {'course_id': course_id, 'name': 'Introduction to Python', 'credits': 3


# Test cases
if __name__ == "__main__":
    print(add_course('CS101', 'Introduction to Python', 3))
    print(remove_course('CS101'))
    print(get_course('CS101'))
```

Output:



Summary:

This section defines simple course registry helpers: add_course builds and returns a course dict, remove_course always returns True as a placeholder, and get_course returns a fixed sample course. The __main__ block prints example outputs. Observation: these are stubs with no real storage or lookup, so removal and retrieval aren't actually tied to prior adds.