AI Assisted Coding

Assignment – 9.1

Name: G.Samshray

Roll:2303A51670

Batch:23

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.

Code:

(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.

Code:

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

    Args:
        numbers (list): A list of numerical values.
    Returns:
        The maximum value from the list of numbers.
    Example:
        >>> find_max([1, 2, 3, 4, 5])
        5
        >>> find_max([-1, -2, -3])
        -1
    """
    if not numbers:
        raise ValueError("The input list cannot be empty.")
    max_value = numbers[0]

    for num in numbers:
        if num > max_value:
            max_value = num
    return max_value

# (b) Inline comments
def find_max(numbers):
    # Check if the input list is empty and raise an error if it is
    if not numbers:
```

```
        raise ValueError("The input list cannot be empty.")

    # Initialize max_value with the first element of the list
    max_value = numbers[0]

    # Iterate through each number in the list
    for num in numbers:
        # Update max_value if the current number is greater
        if num > max_value:
            max_value = num

    # Return the maximum value found
    return max_value
# (c) Google-style documentation
def find_max(numbers):
    """
    Finds the maximum value in a list of numbers.

    Args:
        numbers (list): A list of numerical values.
    Returns:
        The maximum value from the list of numbers.
    Raises:
        ValueError: If the input list is empty.
    Example:
        >>> find_max([1, 2, 3, 4, 5])
        5
        >>> find_max([-1, -2, -3])
        -1
    """
    if not numbers:
```

```
        raise ValueError("The input list cannot be empty.")
    max_value = numbers[0]

    for num in numbers:
        if num > max_value:
            max_value = num
    return max_value
```

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

Code:

```
# (a) Docstring
def login(user, password, credentials):
    """
    Authenticates a user by comparing the provided password with 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 authentication is successful, False otherwise.
```

```
    Example:
        >>> credentials = {'alice': 'password123', 'bob': 'securepass'}
        >>> login('alice', 'password123', credentials)
        True
        >>> login('bob', 'wrongpass', credentials)
        False
    """
    return credentials.get(user) == password
# (b) Inline comments
def login(user, password, credentials):
    # Retrieve the stored password for the given user from the credentials dictionary
    stored_password = credentials.get(user)

    # Compare the provided password with the stored password and return the result
    return stored_password == password
# (c) Google-style documentation
def login(user, password, credentials):
    """
    Authenticates a user by comparing the provided password with 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 authentication is successful, False otherwise.

    Raises:
        KeyError: If the username does not exist in the credentials.
```

```
    Example:
        >>> credentials = {'alice': 'password123', 'bob': 'securepass'}
        >>> login('alice', 'password123', credentials)
        True
        >>> login('bob', 'wrongpass', credentials)
        False
    """
    if user not in credentials:
        raise KeyError(f"Username '{user}' does not exist.")
    return credentials.get(user) == password
```

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.

Code:

```
# conversion.py
```

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

    Args:
        n (int): A non-negative integer in decimal format.

    Returns:
        str: The binary representation of the input decimal number.

    Example:
        >>> decimal_to_binary(10)
        '1010'
        >>> decimal_to_binary(0)
        '0'
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return bin(n)[2:]
def binary_to_decimal(b):
    """Converts a binary string to its decimal representation.

    Args:
        b (str): A binary string.

    Returns:
        int: The decimal representation of the input binary string.

    Example:
        >>> binary_to_decimal('1010')
        10
```

```
        >>> binary_to_decimal('0')
        0
    """
    try:
        return int(b, 2)
    except ValueError:
        raise ValueError("Input must be a valid binary string.")
def decimal_to_hexadecimal(n):
    """Converts a decimal number to its hexadecimal representation.

    Args:
        n (int): A non-negative integer in decimal format.

    Returns:
        str: The hexadecimal representation of the input decimal number.

    Example:
        >>> decimal_to_hexadecimal(255)
        'ff'
        >>> decimal_to_hexadecimal(0)
        '0'
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return hex(n)[2:]
```

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.

Code:

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

    Args:
        n (int): A non-negative integer in decimal format.

    Returns:
        str: The binary representation of the input decimal number.

    Example:
        >>> decimal_to_binary(10)
        '1010'
        >>> decimal_to_binary(0)
        '0'
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return bin(n)[2:]
def binary_to_decimal(b):
    """Converts a binary string to its decimal representation.

    Args:
```

```
        b (str): A binary string.

    Returns:
        int: The decimal representation of the input binary string.

    Example:
        >>> binary_to_decimal('1010')
        10
        >>> binary_to_decimal('0')
        0
    """
    try:
        return int(b, 2)
    except ValueError:
        raise ValueError("Input must be a valid binary string.")
def decimal_to_hexadecimal(n):
    """Converts a decimal number to its hexadecimal representation.

    Args:
        n (int): A non-negative integer in decimal format.

    Returns:
        str: The hexadecimal representation of the input decimal number.

    Example:
        >>> decimal_to_hexadecimal(255)
        'ff'
        >>> decimal_to_hexadecimal(0)
        '0'
    """
    if n < 0:
```

```
        raise ValueError("Input must be a non-negative integer.")
    return hex(n)[2:]
```

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)

2. Add docstrings with Copilot.

3. Generate documentation in the terminal.

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

Code:

```
# courses.py
courses = {}
def add_course(course_id, name, credits):
    """
    Adds a course to the course management system.

    Args:
        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:
        None
    Example:
        >>> add_course('CS101', 'Introduction to Computer Science', 3)
        >>> add_course('MATH201', 'Calculus I', 4)
    """
```

```
    courses[course_id] = {'name': name, 'credits': credits}
def remove_course(course_id):
    """Removes a course from the course management system.
    Args:
        course_id (str): The unique identifier for the course to be removed.
    Returns:
        None
    Example:
        >>> remove_course('CS101')
        >>> remove_course('MATH201')
    """
    if course_id in courses:
        del courses[course_id]
    else:
        raise KeyError(f"Course ID '{course_id}' does not exist.")
def get_course(course_id):
    """Retrieves course information from the course management system.
    Args:
        course_id (str): The unique identifier for the course to be retrieved.
    Returns:
        dict: A dictionary containing the course name and credits.
    Example:
        >>> get_course('CS101')
        {'name': 'Introduction to Computer Science', 'credits': 3}
        >>> get_course('MATH201')
        {'name': 'Calculus I', 'credits': 4}
    """
    if course_id in courses:
        return courses[course_id]
    else:
        raise KeyError(f"Course ID '{course_id}' does not exist.")
```