# AIAC[LAB-9.1]

## Hall Ticket: 2303A51729

## Ajay p

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.

**Prompt:**

#generate the python code to find the maximum element in a list,use in line comments and doucmentation by docstring and google style documentation.

**Code:**

```python
def find_maximum_element(lst):
    """
    This function takes a list as input and returns the maximum element in the list.

    Args:
        lst (list): A list of elements.

    Returns:
        The maximum element in the list.

    Raises:
        ValueError: If the input list is empty.
    """
    # Check if the list is empty
    if not lst:
        raise ValueError("The input list is empty.")

    # Initialize max_element with the first element of the list
    max_element = lst[0]

    # Iterate through the list to find the maximum element
    for element in lst:
        if element > max_element:
            max_element = element

    return max_element
# Example usage
if __name__ == "__main__":
    my_list = [3, 1, 4, 1, 5, 9]
    max_element = find_maximum_element(my_list)
    print(f"The maximum element in the list is: {max_element}")
```

**Output:**

```
The maximum element in the list is: 9
```

**Observation:**

1. The function `find_maximum_element` is defined to find the maximum element in a list.

2. The function includes a docstring that explains its purpose, arguments, return value, and potential exceptions.

3. The function checks if the input list is empty and raises a ValueError if it is.

4. The function initializes the `max_element` variable with the first element of the list and iterates through the list to find the maximum element.

5. The example usage demonstrates how to call the function and print the result.

6. The code is well-structured and includes error handling for empty lists, making it robust and user-friendly.

**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.

**Prompt:**

#generate a python code for login page that takes username and password as input and checks if the username and password are correct. Use in line comments and documentation by three formats: docstring, google style documentation and in line comments.

**Code:**

```python
def login(username, password):
    """
    This function takes a username and password as input and checks if they are correct.

    Args:
        username (str): The username of the user.
        password (str): The password of the user.
    Returns:
        bool: True if the username and password are correct, False otherwise.
    """
    # Define the correct username and password
    correct_username = "admin"
    correct_password = "password123"

    # Check if the provided username and password match the correct ones
    if username == correct_username and password == correct_password:
        return True
    else:
        return False
# Example usage
if __name__ == "__main__":
    # Get username and password input from the user
    input_username = input("Enter your username: ")
    input_password = input("Enter your password: ")

    # Call the login function and check the result
    if login(input_username, input_password):
        print("Login successful!")
    else:
        print("Login failed. Incorrect username or password.")
```

**Output:**

```
Enter your username: madhu
Enter your password: 8008235720
Login failed. Incorrect username or password.
```

**Observations:**

1. The function `login` is defined to check the validity of the username and password.

2. The function includes a docstring that explains its purpose, arguments, and return

3. The function defines the correct username and password for comparison.

4. The function checks if the provided username and password match the correct ones and returns True or False accordingly.

5. The example usage demonstrates how to get user input for username and password and call the login function.

6. The code is simple and straightforward, making it easy to understand and use for basic login functionality.

**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.

**Prompt:**

#Generate a Python module calculator.py containing functions add, subtract, multiply, and divide with proper docstrings (parameters, return values, and divide-by-zero handling).

#Also show terminal commands to display documentation using help()/pydoc and generate HTML docs using pydoc -w calculator and open the file in a browser.

**Code:**

```python
def add(a, b):
    """
    This function takes two numbers as input and returns their sum.

    Args:
        a (float): The first number.
        b (float): The second number.
    Returns:
        float: The sum of the two numbers.
    """
    return a + b

def subtract(a, b):
    """
    This function takes two numbers as input and returns their difference.

    Args:
        a (float): The first number.
        b (float): The second number.
    Returns:
        float: The difference of the two numbers.
    """
    return a - b


def multiply(a, b):
    """
    This function takes two numbers as input and returns their product.

    Args:
        a (float): The first number.
        b (float): The second number.
    Returns:
        float: The product of the two numbers.
    """
    return a * b

def divide(a, b):
    """
    This function takes two numbers as input and returns their quotient.

    Args:
        a (float): The first number.
        b (float): The second number.
    Returns:
        float: The quotient of the two numbers.
    Raises:
        ValueError: If the second number (denominator) is zero.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b
# Example usage
if __name__ == "__main__":
    num1 = 10
    num2 = 5
    print(f"Addition: {add(num1, num2)}")
```

**Output:**

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
```

**Observations:**

1.The module `calculator.py` contains four functions: `add`, `subtract`, `multiply`, and `divide`, each performing basic arithmetic operations.

2.Each function includes a docstring that explains its purpose, arguments, return value, and in the case of `divide`, potential exceptions.

3.The `divide` function includes error handling to raise a ValueError if the second number (denominator) is zero, preventing division by zero errors.

4.The example usage demonstrates how to call each function and print the results.

5.The terminal commands provided show how to access the documentation using both `help()` in the Python interactive shell and `pydoc` in the terminal, as well as how to generate and open HTML documentation.

6.The code is well-structured and includes proper documentation, making it easy for users

**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.

**Prompt:**

#Create a Python module conversion.py with functions decimal_to_binary(n), binary_to_decimal(b), and decimal_to_hexadecimal(n) including proper docstrings. Also show how to auto-generate documentation using pydoc and export it to HTML to view in a browser.

**Code**:

```python
def decimal_to_binary(n):
    """
    This function converts a decimal number to its binary representation.

    Args:
        n (int): A decimal number.

    Returns:
        str: The binary representation of the decimal number.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return bin(n).replace("0b", "")
def binary_to_decimal(b):
    """
    This function converts a binary string to its decimal representation.
    Args:
        b (str): A binary string.
    Returns:
        int: The decimal representation of the binary string.
    """
    try:
        return int(b, 2)
    except ValueError:
        raise ValueError("Input must be a valid binary string.")
def decimal_to_hexadecimal(n):
    """
    This function converts a decimal number to its hexadecimal representation.

    Args:
        n (int): A decimal number.

    Returns:
        str: The hexadecimal representation of the decimal number.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return hex(n).replace("0x", "").upper()
# Example usage
if __name__ == "__main__":
    decimal_number = 42
    binary_string = "101010"

    print(f"Decimal to Binary: {decimal_number} -> {decimal_to_binary(decimal_number)}")
    print(f"Binary to Decimal: {binary_string} -> {binary_to_decimal(binary_string)}")
    print(f"Decimal to Hexadecimal: {decimal_number} -> {decimal_to_hexadecimal(decimal_number)}")
```

**Output:**

```
Decimal to Binary: 42 -> 101010
Binary to Decimal: 101010 -> 42
Decimal to Hexadecimal: 42 -> 2A
```

#Create a Python module conversion.py with functions decimal_to_binary(n), binary_to_decimal(b), and decimal_to_hexadecimal(n) including proper docstrings. Also show how to auto-generate documentation using pydoc and export it to HTML to view in a browser.

**Functions**

**binary_to_decimal**(b)
This function converts a binary string to its decimal representation.
Args:
b (str): A binary string.
Returns:
int: The decimal representation of the binary string.

**decimal_to_binary**(n)
This function converts a decimal number to its binary representation.

Args:
n (int): A decimal number.

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

**decimal_to_hexadecimal**(n)
This function converts a decimal number to its hexadecimal representation.

Args:
n (int): A decimal number.

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

## Observation:

1.The module conversion.py contains three functions: decimal_to_binary, binary_to_decimal, and decimal_to_hexadecimal,each performing a specific conversion between number systems.

2.Each function includes a docstring that explains its purpose, arguments, return value, and potential exceptions.

3.The example usage demonstrates how to call each function and print the results.

4.The terminal commands provided show how to use the help() function to display documentation in the Python interactive shell and how to use pydoc to generate HTML documentation for the module.

5.The generated HTML documentation can be opened in a web browser for easy viewing, making it accessible for users who prefer a graphical interface.

6.The code includes error handling to ensure that the functions receive valid input, making it robust and user-friendly.

## 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.

**Prompt:**

#Create course.py implementing course management functions with docstrings.

#Generate and export HTML documentation using pydoc and open it in a browser.

**Code:**

```
class Course:
    """
    A class to represent a course and manage its details.

    Attributes:
        name (str): The name of the course.
        instructor (str): The name of the instructor.
        students (list): A list of students enrolled in the course.
    """

    def __init__(self, name, instructor):
        """
        Initializes a new Course instance.

        Args:
            name (str): The name of the course.
            instructor (str): The name of the instructor.
        """
        self.name = name
        self.instructor = instructor
        self.students = []

    def add_student(self, student_name):
        """
        Adds a student to the course.

        Args:
            student_name (str): The name of the student to be added.
        """
        self.students.append(student_name)

    def remove_student(self, student_name):
        """
        Removes a student from the course.

        Args:
            student_name (str): The name of the student to be removed.
        Raises:
            ValueError: If the student is not found in the course.
        """
        if student_name in self.students:
            self.students.remove(student_name)
        else:
            raise ValueError(f"Student '{student_name}' not found in the course.")
    def get_course_info(self):
        """
        Returns a string containing the course information.

        Returns:
            str: A string with the course name, instructor, and enrolled students.
        """
        return f"Course Name: {self.name}\nInstructor: {self.instructor}\nStudents: {', '.join(self.students)}"
# Example usage
if __name__ == "__main__":
    course = Course("Introduction to Python", "Dr. Smith")
    course.add_student("Alice")
    course.add_student("Bob")
    print(course.get_course_info())
    course.remove_student("Alice")
    print(course.get_course_info())
```

**Output:**

```
Course Name: Introduction to Python
Instructor: Dr. Smith
Students: Alice, Bob
Course Name: Introduction to Python
Instructor: Dr. Smith
Students: Bob
```

Create course.py implementing course management functions with docstrings.
Generate and export HTML documentation using pydoc and open it in a browser.

**Classes**

builtins.object
Course

class **Course**(builtins.object)
Course(name, instructor)

A class to represent a course and manage its details.

Attributes:
name (str): The name of the course.
instructor (str): The name of the instructor.
students (list): A list of students enrolled in the course.

Methods defined here:

**__init__**(self, name, instructor)
Initializes a new Course instance.

Args:
name (str): The name of the course.
instructor (str): The name of the instructor.

**add_student**(self, student_name)
Adds a student to the course.

Args:
student_name (str): The name of the student to be added.

**get_course_info**(self)
Returns a string containing the course information.

Returns:
str: A string with the course name, instructor, and enrolled students.

**remove_student**(self, student_name)
Removes a student from the course.

Args:
student_name (str): The name of the student to be removed.
Raises:
ValueError: If the student is not found in the course.

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables

**__weakref__**
list of weak references to the object

## Observation:

1. The Course class is defined to represent a course with attributes for the course name, instructor, and a list of enrolled students.

2. The class includes methods to add and remove students, as well as a method to retrieve course information.

3. Each method is documented with a docstring that explains its purpose, arguments, return value, and potential exceptions.

4. The example usage demonstrates how to create a Course instance, add and remove students, and print course information.

5. The terminal commands provided allow users to view the documentation using help() and generate HTML documentation using pydoc, making it easy to access and understand the functionality of the Course class.

6. The code is well-structured and includes error handling for cases where a student to be removed is not found in the course, enhancing its robustness.