

## AIAC Assignment – 9.2

**K.Lokesh**

**2303A51201,B21**

### Task Description -1 (Documentation – Function Summary Generation)

Task:

Use AI to generate concise functional summaries for each Python function in a given script.

Instructions:

- Provide a Python script to the AI.
- Ask the AI to write a short summary describing the purpose of each function.
- Ensure summaries are brief and technically accurate.
- Do not include code implementation details.

```
Python script as follows below:
def add_numbers(a, b):
    return a + b

def is_even(num):
    return num % 2 == 0

def find_max(lst):
    return max(lst)

def sort_list(lst):
    return sorted(lst)'''
1. 'add_numbers(a, b)': This function takes two numbers as input and returns their sum.
2. 'is_even(num)': This function checks if a given number is even and returns a boolean value accordingly.
3. 'find_max(lst)': This function takes a list as input and returns the maximum value from that list.
4. 'sort_list(lst)': This function takes a list as input and returns a new list that is sorted in ascending order.
```

### Task Description -2 (Documentation – Logical Explanation for Conditions and Loops)

Task:

Use AI to document the logic behind conditional statements and loops in a Python program.

Instructions:

- Provide a Python program without comments.
- Instruct AI to explain only decision-making logic and loop behavior.
- Skip basic syntax explanations.

```
def check_number(num):
    if num > 0:
        return "Positive"
    elif num < 0:
        return "Negative"
    else:
        return "Zero"

def count_down(n):
    while n > 0:
        print(n)
        n -= 1
    print("Liftoff!")

# Explanation of the logic:
# 1. The function 'check_number' takes a number as input and uses conditional statements to determine if the number is positive, negative, or zero.
# 2. The 'if' statement checks if the number is greater than zero, returning "Positive" if true.
# 3. The 'elif' statement checks if the number is less than zero, returning "Negative" if true.
# 4. If neither condition is met, the 'else' statement returns "Zero".
# 5. The function 'count_down' takes a number 'n' and uses a 'while' loop to count down from 'n' to 1.
# 6. The loop continues to execute as long as 'n' is greater than zero, printing the current value of 'n' and then decrementing 'n' by 1.
# 7. Once 'n' reaches zero, the loop terminates and prints "Liftoff!" to indicate the end of the countdown.
```

### Task Description -3 (Documentation – File-Level Overview)

#### Task:

Use AI to generate a high-level overview describing the functionality of an entire Python file.

#### Instructions:

- Provide the complete Python file to AI.
- Ask AI to write a brief overview summarizing the file's purpose and functionality.
- Place the overview at the top of the file.

#### Expected Output -3:

A Python file with a clear and concise file-level overview at the beginning.

```
"""
This Python file implements a simple student management system.
It provides functions to add student records, calculate average marks,
determine grades, and display student details. The program demonstrates
basic data processing and modular design using functions.
"""

def add_student(name, marks):
    student = {
        "name": name,
        "marks": marks
    }
    return student

def calculate_average(marks):
    return sum(marks) / len(marks)

def assign_grade(avg):
    if avg >= 90:
        return "A"
    elif avg >= 75:
        return "B"
    elif avg >= 50:
        return "C"
    else:
        return "Fail"

def display_student(student):
    avg = calculate_average(student["marks"])
    grade = assign_grade(avg)
    print("Name:", student["name"])
    print("Marks:", student["marks"])
    print("Average:", avg)
    print("Grade:", grade)

def main():
    student1 = add_student("Loki", [85, 90, 78])
    student2 = add_student("Ravi", [60, 70, 65])

    display_student(student1)
    print("-----")
    display_student(student2)

if __name__ == "__main__":
    main()
```

#### Task Description -4 (Documentation – Refine Existing Documentation)

##### Task:

Use AI to improve clarity and consistency of existing documentation in Python code.

##### Instructions:

- Provide Python code containing basic or unclear comments.
- Ask AI to rewrite the documentation to improve clarity and consistency.
- Ensure technical meaning remains unchanged.

##### Expected Output -4:

Python code with refined and improved documentation that is clear and consistent.

```
def binary_search(arr, target):  
    # search element  
    left = 0  
    right = len(arr) - 1  
  
    while left <= right:  
        mid = (left + right) // 2  
  
        # check mid  
        if arr[mid] == target:  
            return mid  
  
        # go right side  
        elif arr[mid] < target:  
            left = mid + 1  
  
        # go left side  
        else:  
            right = mid - 1  
  
    # not found  
    return -1  
  
def main():  
    # sorted list  
    data = [2, 5, 8, 12, 16, 23, 38, 56]  
  
    # element to find  
    key = 23  
  
    # call function  
    result = binary_search(data, key)  
  
    # print result  
    if result != -1:  
        print("Found at index", result)  
    else:  
        print("Not found")  
  
if __name__ == "__main__":  
    main()
```

```

#Refined Output:
def binary_search(arr, target):
    """
    Performs binary search on a sorted list to find the target value.

    Returns the index of the target if found, otherwise returns -1.
    """
    left = 0
    right = len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        # Check if the middle element matches the target
        if arr[mid] == target:
            return mid

        # If target is greater, search in the right half
        elif arr[mid] < target:
            left = mid + 1

        # If target is smaller, search in the left half
        else:
            right = mid - 1

    # Target not found in the list
    return -1

def main():
    """
    Demonstrates the usage of binary search on a sorted list.
    """
    data = [2, 5, 8, 12, 16, 23, 38, 56]
    key = 23

    result = binary_search(data, key)

    if result != -1:
        print("Found at index", result)
    else:
        print("Not found")

if __name__ == "__main__":
    main()

```

#### Task Description -5 (Documentation – Prompt Detail Impact Study)

##### Task:

Study the impact of prompt detail on AI-generated documentation quality.

##### Instructions:

Create two prompts: one brief and one detailed.

- Use both prompts to document the same Python function.
- Compare the generated outputs.

##### Expected Output -5:

A comparison table highlighting differences in completeness, clarity, and accuracy of documentation.

```
# Brief Prompt
Brief Prompt: "Document the following Python function that calculates the factorial of a number."
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

# Detailed Prompt
Detailed Prompt: "Document the following Python function that calculates the factorial of a number. Include a description of what the function does, its parameters, return value, and any exceptions it may raise."
def factorial(n):
    """
    Calculate the factorial of a number.

    Parameters:
    n (int): The number for which to calculate the factorial. Must be a non-negative integer.

    Returns:
    int: The factorial of the input number. For example, factorial(5) returns 120.

    Raises:
    ValueError: If n is a negative integer.
    TypeError: If n is not an integer.
    """
    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")

    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

# Comparison Table


| Aspect       | Brief Prompt Documentation                                | Detailed Prompt Documentation                                                           |
|--------------|-----------------------------------------------------------|-----------------------------------------------------------------------------------------|
| Completeness | Provides a basic description of the function.             | Offers a comprehensive description, including parameters, return value, and exceptions. |
| Clarity      | May be unclear about the function's purpose and usage.    | Clearly explains the function's purpose, how to use it, and potential errors.           |
| Accuracy     | May miss important details about the function's behavior. | Accurately describes the function's behavior, including edge cases and error handling.  |


```