AI Assisted Coding
Assignment - 9.4
2303A510B7
Batch - 14

TASK 1 – Auto-Generating Function Documentation

Code:

```python
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    if b == 0:
        return "Cannot divide by zero"
    return a / b
def is_even(n):
    return n % 2 == 0
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def main():
    print("Add:", add(10, 5))
    print("Subtract:", subtract(10, 5))
    print("Multiply:", multiply(10, 5))
    print("Divide:", divide(10, 5))
    print("Is Even:", is_even(10))
    print("Factorial:", factorial(5))
if __name__ == "__main__":
    main()
```

Prompt: #generate Google-style function docstrings for each function for above code

Modified Code:

```
def add(a, b):
    """
    Adds two numbers.
    Args:
        a (int or float): The first number.
        b (int or float): The second number.
    Returns:
        int or float: The sum of the two numbers.
    """
    return a + b
def subtract(a, b):
    """
    Subtracts the second number from the first number.
    Args:
        a (int or float): The first number.
        b (int or float): The second number.
    Returns:
        int or float: The result of the subtraction.
    """
    return a - b
def multiply(a, b):
    """
    Multiplies two numbers.
    Args:
        a (int or float): The first number.
        b (int or float): The second number.
    Returns:
        int or float: The product of the two numbers.
    """
    return a * b
def divide(a, b):
    """
    Divides the first number by the second number.
    Args:
        a (int or float): The numerator.
        b (int or float): The denominator.
    Returns:
        int, float, or str: The result of the division, or a string if division by zero occurs.
    """
    if b == 0:
        return "Cannot divide by zero"
    return a / b
def is_even(n):
    """
    Checks if a number is even.
    Args:
        n (int): The number to check.
    Returns:
        bool: True if the number is even, False otherwise.
    """
    return n % 2 == 0
def factorial(n):
    """
    Calculates the factorial of a number.
    Args:
        n (int): The number to calculate the factorial of.
    Returns:
        int: The factorial of the number.
    """
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    if b == 0:
        return "Cannot divide by zero"
    return a / b
def is_even(n):
    return n % 2 == 0
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def main():
    print("Add:", add(10, 5))
    print("Subtract:", subtract(10, 5))
    print("Multiply:", multiply(10, 5))
    print("Divide:", divide(10, 5))
    print("Is Even:", is_even(10))
    print("Factorial:", factorial(5))
if __name__ == "__main__":
    main()
```

Task 2 : Enhancing Readability Through AI-Generated Inline **Comments**

Code:

```python
# Fibonacci sequence using loop and condition
def fibonacci(n):
    fib = []
    a, b = 0, 1
    for i in range(n):
        fib.append(a)
        a, b = b, a + b
    return fib
# Bubble Sort algorithm using loops and conditionals
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:   # conditional logic
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
# Linear Search algorithm
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:   # conditional logic
            return i
    return -1
# Main program
def main():
    # Fibonacci
    print("Fibonacci Sequence:")
    print(fibonacci(10))
    # Sorting
    numbers = [64, 34, 25, 12, 22, 11, 90]
    print("\nOriginal List:", numbers)
    sorted_numbers = bubble_sort(numbers.copy())
    print("Sorted List:", sorted_numbers)
    # Searching
    target = 22
    index = linear_search(sorted_numbers, target)
    if index != -1:
        print("\nElement", target, "found at index:", index)
    else:
        print("\nElement not found")
if __name__ == "__main__":
    main()
```

Prompt:

Automatically insert inline comments only for complex or non-obvious logic

Avoid commenting on trivial or self-explanatory syntax The goal is to improve clarity without cluttering the code.

Modified Code:

```python
# Fibonacci sequence using loop and condition
def fibonacci(n):
    fib = []
    a, b = 0, 1
    for i in range(n):
        fib.append(a)
        a, b = b, a + b
    return fib
# Bubble Sort algorithm using loops and conditionals
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:   # conditional logic
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
# Linear Search algorithm
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:   # conditional logic
            return i
    return -1
# Main program
def main():
    # Fibonacci
    print("Fibonacci Sequence:")
    print(fibonacci(10))
    # Sorting
    numbers = [64, 34, 25, 12, 22, 11, 90]
    print("\nOriginal List:", numbers)
    sorted_numbers = bubble_sort(numbers.copy())  # Use a copy to preserve the original list
    print("Sorted List:", sorted_numbers)

    # Searching
    target = 22
    index = linear_search(sorted_numbers, target)
    if index != -1:
        print("\nElement", target, "found at index:", index)
    else:
        print("\nElement not found")
if __name__ == "__main__":
    main()
```

Task 3 : Generating Module-Level Documentation for a Python **Package**

Code:

```python
# ------------------------------
# Student Utility Module
# ------------------------------
# Class definition
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def grade(self):
        if self.marks >= 90:
            return "A"
        elif self.marks >= 75:
            return "B"
        elif self.marks >= 50:
            return "C"
        else:
            return "Fail"
# Function to calculate average marks
def calculate_average(marks_list):
    total = 0
    for m in marks_list:
        total += m
    return total / len(marks_list)


# Fibonacci algorithm
def fibonacci(n):
    seq = []
    a, b = 0, 1
    for _ in range(n):
        seq.append(a)
        a, b = b, a + b
    return seq
# Sorting algorithm (Bubble Sort)
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
# Searching algorithm (Linear Search)
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1
# Main function (module test)
def main():
    # Student class test
    s1 = Student("Harini", 85)
    s2 = Student("Ravi", 45)

    print("Student:", s1.name, "Grade:", s1.grade())
    print("Student:", s2.name, "Grade:", s2.grade())
    # Average marks
    marks = [85, 90, 78, 92, 88]
    print("\nAverage Marks:", calculate_average(marks))
    # Fibonacci
    print("\nFibonacci Sequence:", fibonacci(10))
    # Sorting
    nums = [64, 12, 34, 25, 9]
    print("\nOriginal List:", nums)
    print("Sorted List:", bubble_sort(nums))
    # Searching
    target = 25
    index = linear_search(nums, target)
    if index != -1:
        print("\nElement", target, "found at index", index)
    else:
        print("\nElement not found")
# Run module directly
if __name__ == "__main__":
    main()
```

Prompt: generate a module-level docstring at the top of the file that includes

Modified code:

```python
# Function to calculate average marks
def calculate_average(marks_list):
    total = 0
    for m in marks_list:
        total += m
    return total / len(marks_list)


"""
ASS-9.4(10B7).py

This module demonstrates the implementation of various algorithms and utility functions,
including Fibonacci sequence generation, Bubble Sort, Linear Search, and average calculation.
It also includes a Student class to demonstrate object-oriented programming concepts.

Functions:
    fibonacci(n): Generates the first 'n' numbers in the Fibonacci sequence.
    bubble_sort(arr): Sorts a list of numbers using the Bubble Sort algorithm.
    linear_search(arr, target): Searches for a target value in a list using Linear Search.
    calculate_average(marks_list): Calculates the average of a list of marks.
    main(): Tests the implemented functions and demonstrates their usage.

Classes:
    Student: Represents a student with a name and marks, and provides a method
             to calculate the grade.

Example Usage:
    Run the script directly to see the output of the implemented functions,
    including Fibonacci sequence generation, sorting, searching, and average calculation.

Author: [Your Name]
Date: [Date]
"""

# Fibonacci algorithm
def fibonacci(n):
    seq = []
    a, b = 0, 1
    for _ in range(n):
        seq.append(a)
        a, b = b, a + b
```

Task 4 : Converting Developer Comments into Structured Docstrings

Code:

```python
# ----------------------------------------
# Function to generate Fibonacci sequence
# ----------------------------------------
def fibonacci(n):
    # Create an empty list to store Fibonacci numbers
    fib_list = []
    # Initialize first two Fibonacci numbers
    a = 0
    b = 1
    # Loop n times to generate n Fibonacci numbers
    for i in range(n):
        # Append the current value of 'a' to the list
        fib_list.append(a)
        # Update values: next number = a + b
        a, b = b, a + b
    # Return the complete Fibonacci list
    return fib_list
# ----------------------------------------
# Function to sort a list using Bubble Sort
# ----------------------------------------
def bubble_sort(arr):
    # Get the length of the list
    n = len(arr)
    # Outer loop for number of passes
    for i in range(n):
        # Inner loop to compare adjacent elements
        for j in range(0, n - i - 1):
            # If the current element is greater than the next element
            if arr[j] > arr[j + 1]:
                # Swap the elements
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    # Return the sorted list
    return arr
# ----------------------------------------
# Function to search an element using Linear Search
# ----------------------------------------
def linear_search(arr, target):
    # Loop through each element in the list
    for index in range(len(arr)):
        # Check if current element matches the target
        if arr[index] == target:
            # Return the index if found
            return index
    # Return -1 if the element is not found
    return -1
# ----------------------------------------
# Main function to test all algorithms
# ----------------------------------------
def main():
    # -------- Fibonacci Test --------
    print("Fibonacci Sequence (10 terms):")
    fib = fibonacci(10)
    print(fib)
    # -------- Sorting Test --------
    numbers = [45, 12, 78, 34, 23, 9]
    print("\nOriginal List:", numbers)

    sorted_numbers = bubble_sort(numbers.copy())  # Use copy to keep original unchanged
    print("Sorted List:", sorted_numbers)

    # -------- Searching Test --------
    target = 34
    result = linear_search(sorted_numbers, target)

    # Conditional logic to check search result
    if result != -1:
        print("\nElement", target, "found at index:", result)
    else:
        print("\nElement not found")
# ----------------------------------------
# Run the program only if executed directly
# ----------------------------------------
if __name__ == "__main__":
    main()
```

Prompt : Convert inline comments into Google- or NumPy-style docstrings, preserve meaning, and remove redundant inline comments.

Modified Code:

```python
def fibonacci(n):
    """
    Generates the first 'n' numbers in the Fibonacci sequence.

    Args:
        n (int): The number of Fibonacci numbers to generate.

    Returns:
        list: A list containing the first 'n' Fibonacci numbers.
    """
    fib_list = []
    a, b = 0, 1
    for i in range(n):
        fib_list.append(a)
        a, b = b, a + b
    return fib_list
def bubble_sort(arr):
    """
    Sorts a list of numbers using the Bubble Sort algorithm.

    Args:
        arr (list): The list of numbers to be sorted.

    Returns:
        list: The sorted list of numbers.
    """
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
def linear_search(arr, target):
    """
    Searches for a target value in a list using the Linear Search algorithm.

    Args:
        arr (list): The list of elements to search through.
        target (any): The value to search for.

    Returns:
        int: The index of the target value if found, otherwise None.
    """
    for index in range(len(arr)):
        if arr[index] == target:
            return index
    # Return -1 if the element is not found
    return -1
# ----------------------------------------
# Main function to test all algorithms
# ----------------------------------------
def main():
    # -------- Fibonacci Test --------
    print("Fibonacci Sequence (10 terms):")
    fib = fibonacci(10)
    print(fib)
    # -------- Sorting Test --------
    numbers = [45, 12, 78, 34, 23, 9]
    print("\nOriginal List:", numbers)

    sorted_numbers = bubble_sort(numbers.copy())  # Use copy to keep original unchanged
    print("Sorted List:", sorted_numbers)

    # -------- Searching Test --------
    target = 34
    result = linear_search(sorted_numbers, target)

    # Conditional logic to check search result
    if result != -1:
        print("\nElement", target, "found at index:", result)
    else:
        print("\nElement not found")
# ----------------------------------------
# Run the program only if executed directly
# ----------------------------------------
if __name__ == "__main__":
    main()
```

Task 5 : Building a Mini Automatic Documentation Generator

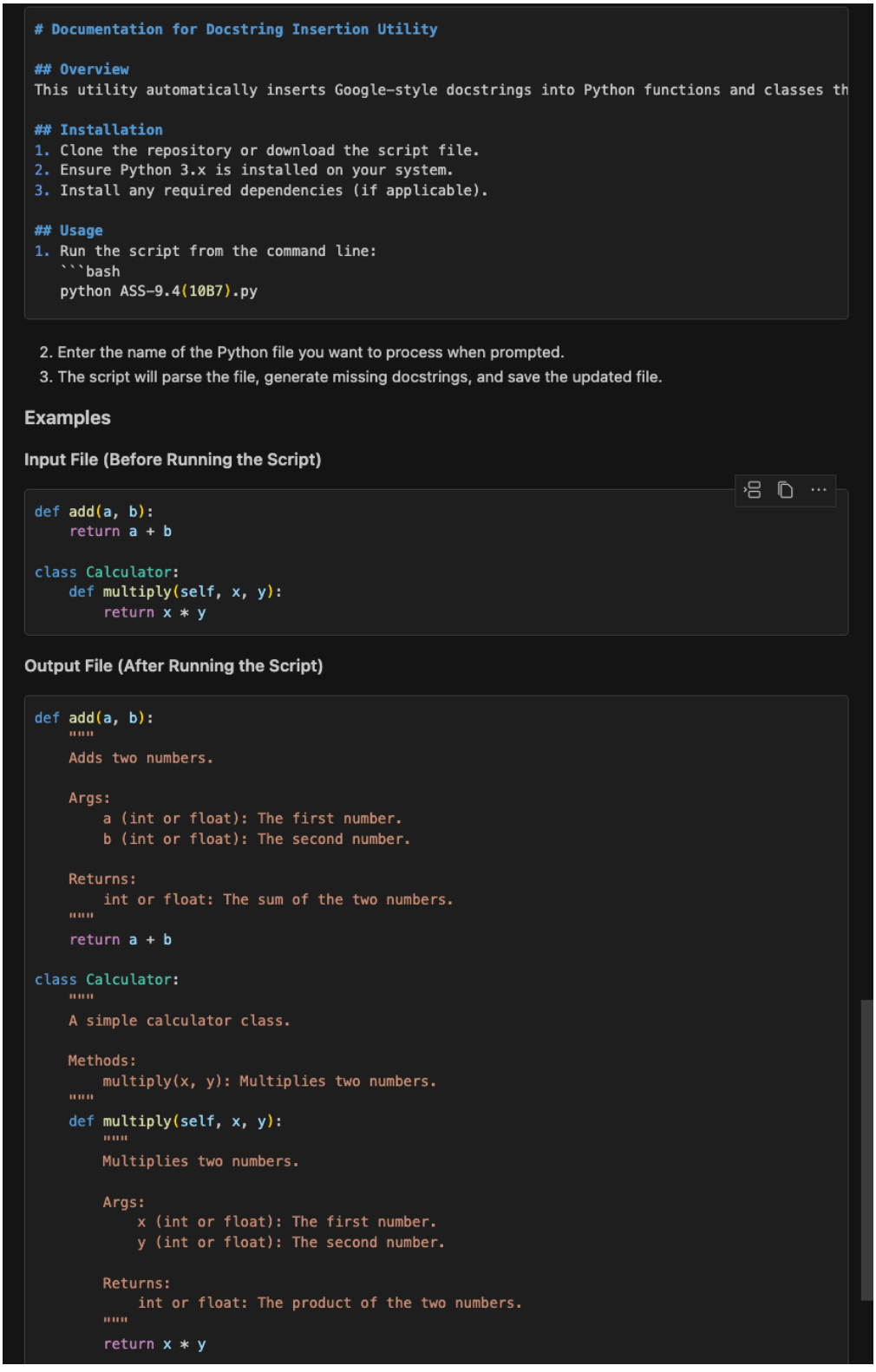
Code:

```
import ast
def generate_function_docstring(func_name, args):
    params = "\n".join([f"        {arg}: TYPE - description." for arg in args])
    return f'''    """{func_name} summary.
    Args:
{params if params else "        None"}
    Returns:
        TYPE: description.
    """
'''
def generate_class_docstring(class_name):
    return f'''    """{class_name} summary.
    Attributes:
        None
    """
'''
def insert_docstrings(file_path):
    with open(file_path, "r") as f:
        source = f.read()
    tree = ast.parse(source)
    lines = source.split("\n")
    # Store insert positions and docstrings
    inserts = []
    for node in ast.walk(tree):
        # Detect functions
        if isinstance(node, ast.FunctionDef):
            if not ast.get_docstring(node):
                args = [arg.arg for arg in node.args.args]
                doc = generate_function_docstring(node.name, args)
                inserts.append((node.lineno, doc))
        # Detect classes
        if isinstance(node, ast.ClassDef):
            if not ast.get_docstring(node):
                doc = generate_class_docstring(node.name)
                inserts.append((node.lineno, doc))
    # Insert docstrings in reverse order (to avoid line shift)
    for lineno, doc in sorted(inserts, reverse=True):
        lines.insert(lineno, doc)

    with open(file_path, "w") as f:
        f.write("\n".join(lines))
    print("Docstrings inserted successfully!")
# Run from command line
if __name__ == "__main__":
    file_name = input("Enter Python file name: ")
    insert_docstrings(file_name)
```

Prompt : Assist in generating or refining documentation scaffolding for this utility. Focus on creating a clear structure (overview, installation, usage, examples, limitations, and future work) rather than perfect or complete documentation."

Modified Code:



```
# Documentation for Docstring Insertion Utility

## Overview
This utility automatically inserts Google-style docstrings into Python functions and classes th

## Installation
1. Clone the repository or download the script file.
2. Ensure Python 3.x is installed on your system.
3. Install any required dependencies (if applicable).

## Usage
1. Run the script from the command line:
   ```bash
   python ASS-9.4(10B7).py
```

2. Enter the name of the Python file you want to process when prompted.
3. The script will parse the file, generate missing docstrings, and save the updated file.

Examples

Input File (Before Running the Script)

```
def add(a, b):
    return a + b

class Calculator:
    def multiply(self, x, y):
        return x * y
```

Output File (After Running the Script)

```
def add(a, b):
    """
    Adds two numbers.

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

    Returns:
        int or float: The sum of the two numbers.
    """
    return a + b

class Calculator:
    """
    A simple calculator class.

    Methods:
        multiply(x, y): Multiplies two numbers.
    """
    def multiply(self, x, y):
        """
        Multiplies two numbers.

        Args:
            x (int or float): The first number.
            y (int or float): The second number.

        Returns:
            int or float: The product of the two numbers.
        """
        return x * y
```
```