9.4-Assignment

N SOURISH

2303A52360

Batch – 44

Task 1: Auto-generating Function Documentation in a Shared Codebase.

Code:

```
def add_numbers(a, b):
    """
    Adds two numbers and returns the result.

    Args:
        a (int | float): First number.
        b (int | float): Second number.

    Returns:
        int | float: Sum of the two numbers.

    Example:
        >>> add_numbers(3, 4)
        7
    """
    return a + b


def is_even(n):
    """
    Checks whether a number is even.

    Args:
        n (int): The number to check.

    Returns:
        bool: True if the number is even, False otherwise.

    Example:
        >>> is_even(6)
        True
    """
    return n % 2 == 0
```

```python
def factorial(n):
    """
    Computes the factorial of a non-negative integer using recursion.

    Args:
        n (int): Non-negative integer.

    Returns:
        int: Factorial of n.

    Example:
        >>> factorial(5)
        120
    """
    if n == 0:
        return 1
    return n * factorial(n - 1)


def find_max(lst):
    """
    Finds the maximum value in a list.

    Args:
        lst (list[int | float]): List of numbers.

    Returns:
        int | float: Maximum value in the list.

    Example:
        >>> find_max([2, 9, 4, 1])
        9
    """
    max_val = lst[0]
    for i in lst:
        if i > max_val:
            max_val = i
    return max_val
```

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

Code:

```python
def fibonacci(n):
    sequence = [0, 1]
    for i in range(2, n):
        next_value = sequence[i - 1] + sequence[i - 2]
        sequence.append(next_value)
    return sequence
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] > target:
            right = mid - 1
        else:
            left = mid + 1
    return -1
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
if __name__ == "__main__":
    print("Fibonacci:", fibonacci(7))
    arr = [2, 4, 6, 8, 10, 12]
    print("Binary Search (8):", binary_search(arr, 8))
    print("Bubble Sort:", bubble_sort([5, 2, 9, 1, 3]))
```

Output:

```
Fibonacci: [0, 1, 1, 2, 3, 5, 8]
Binary Search (8): 3
Bubble Sort: [1, 2, 3, 5, 9]
```

Task 3: Generating Module-Level Document for a Python Package.

Code:

```python
import math
def calculate_mean(numbers):
    if not numbers:
        raise ValueError("List cannot be empty.")
    return sum(numbers) / len(numbers)
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] > target:
            right = mid - 1
        else:
            left = mid + 1
    return -1
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
```

```python
class Statistics:
    def __init__(self, data):
        if not data:
            raise ValueError("Data cannot be empty.")
        self.data = data
    def variance(self):
        mean = sum(self.data) / len(self.data)
        return sum((x - mean) ** 2 for x in self.data) / len(self.data)
    def standard_deviation(self):
        return math.sqrt(self.variance())

if __name__ == "__main__":
    numbers = [5, 2, 9, 1, 3]
    print("Original List:", numbers)
    sorted_list = bubble_sort(numbers.copy())
    print("Sorted List:", sorted_list)
    print("Mean:", calculate_mean(sorted_list))
    print("Binary Search (9):", binary_search(sorted_list, 9))
    stats = Statistics(sorted_list)
    print("Variance:", stats.variance())
    print("Standard Deviation:", stats.standard_deviation())
```

Output:

```
Original List: [5, 2, 9, 1, 3]
Sorted List: [1, 2, 3, 5, 9]
Mean: 4.0
Binary Search (9): 4
Variance: 8.0
Standard Deviation: 2.8284271247461903
```

Task 4: Converting Developer Comments into Structural Docstrings.

Code:

```
def calculate_discount(price, discount_percent):
    discount_amount = (price * discount_percent) / 100
    final_price = price - discount_amount
    return final_price
def check_even_odd(number):
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"
if __name__ == "__main__":
    print("---- Discount Calculator ----")
    price = float(input("Enter the original price: "))
    discount = float(input("Enter the discount percentage: "))
    print("\n---- Even or Odd Checker ----")
    number = int(input("Enter a number: "))
    final_price = calculate_discount(price, discount)
    result = check_even_odd(number)
    print("\n----- Results -----")
    print("Final Price after discount:", final_price)
    print("The number is:", result)
```

Input:

```
---- Discount Calculator ----
Enter the original price: 500
Enter the discount percentage: 2

---- Even or Odd Checker ----
Enter a number: 34
```

Output:

```
----- Results -----
Final Price after discount: 490.0
The number is: Even
```

Task 5: Building a Mini Automatic Documentation Generator.

Code:

Main.py

```
class InsertDocstrings(ast.NodeTransformer):

    def visit_FunctionDef(self, node):
        if ast.get_docstring(node) is None:
            doc = ast.Expr(value=ast.Constant(generate_function_doc(node)))
            node.body.insert(0, doc)
        return node

    def visit_ClassDef(self, node):
        if ast.get_docstring(node) is None:
            doc = ast.Expr(value=ast.Constant(generate_class_doc(node)))
            node.body.insert(0, doc)
        return node


def process_file(filename):
    with open(filename, "r", encoding="utf-8") as file:
        source = file.read()

    tree = ast.parse(source)
    tree = InsertDocstrings().visit(tree)
    ast.fix_missing_locations(tree)

    new_code = ast.unparse(tree)

    output_file = filename.replace(".py", "_documented.py")

    with open(output_file, "w", encoding="utf-8") as file:
        file.write(new_code)

    print("\nDocstrings added successfully!")
    print("Output file created:", output_file)


if __name__ == "__main__":
    fname = input("Enter python file name: ")
    process_file(fname)
```

Sample_input.py

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def average(self):
        return sum(self.marks) / len(self.marks)


def welcome(user):
    print("Welcome", user)
```

Input:

```
Enter python file name: sample_input.py
```

Output:

```
Docstrings added successfully!
Output file created: sample_input_documented.py
```