AI Assisted Coding-9.4

A.Shashank || Batch-09 || 2303A51639

Task 1: Auto-Generating Function Documentation in a Shared Codebase

Code:

```
#Task-01: Auto-Generating Function Documentation in a Shared Codebase
"""Adds two numbers together.
Args:
    a (int or float): The first number to add.
    b (int or float): The second number to add.
Returns:
    int or float: The sum of a and b.
Example:
    >>> add_numbers(5, 3)
    8
    >>> add_numbers(2.5, 1.5)
    4.0
"""
"""Calculates the factorial of a non-negative integer using recursion.
Args:
    n (int): A non-negative integer for which to calculate the factorial.
Returns:
    int: The factorial of n (n!).
Raises:
    RecursionError: If n is negative (will cause infinite recursion).
Example:
    >>> factorial(5)
    120
    >>> factorial(0)
    1
"""
"""Checks whether a given number is a prime number.
Args:
    num (int): The number to check for primality.
Returns:
    bool: True if num is a prime number, False otherwise.
Example:
    >>> is_prime(7)
    True
    >>> is_prime(10)
    False
    >>> is_prime(1)
    False
"""
def add_numbers(a, b):
    return a + b

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

def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True
```

Task 2: Enhancing Readability Through AI-Generated Inline Comments

Code:

```
#Task-02: Enhancing Readability Through AI-Generated Inline
Comments
def fibonacci(n):
    if n <= 0:
        return []  # Return empty list for invalid input

    elif n == 1:
        return [0]  # Base case: first Fibonacci number

    seq = [0, 1]  # Initialize sequence with first two numbers

    for i in range(2, n):
        # Each number is the sum of the previous two numbers
        seq.append(seq[i - 1] + seq[i - 2])

    return seq
```

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

Code:

```
#Task-03: Generating Module-Level Documentation for a Python
Package

import math

def area_circle(radius):
    return math.pi * radius * radius

def area_square(side):
    return side * side
    """
    Geometry Calculation Module

    This module provides utility functions for calculating the
areas of basic geometric shapes.
    It serves as a lightweight reference implementation for
fundamental geometry formulas.

    Dependencies:
        - math: Standard library module for mathematical
constants and operations.

    Key Functions:
        - area_circle(radius): Calculates the area of a circle
given its radius.
        - area_square(side): Calculates the area of a square
given its side length.

    Example Usage:
        >>> from lab_9_4 import area_circle, area_square
        >>> area_circle(5)
        78.53981633974483
        >>> area_square(4)
        16
    """
```

Task 4: Converting Developer Comments into Structured Docstrings

Code:

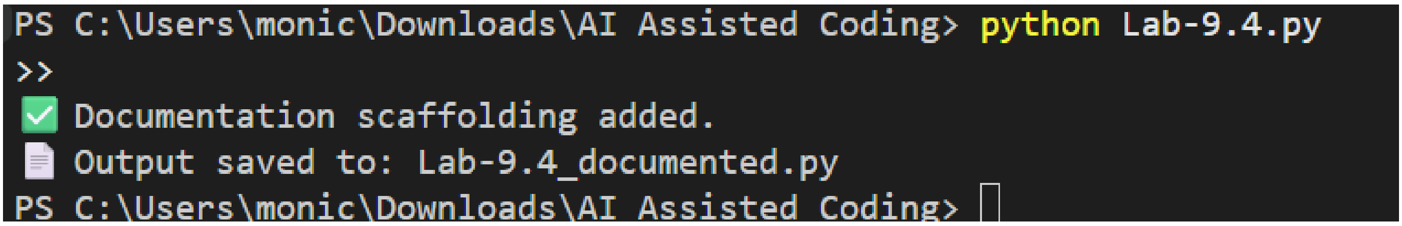
```
#Task-04: Converting Developer Comments into Structured
Docstrings

def calculate_discount(price, discount):
    """Calculates the final price after applying a discount.

    Args:
        price (int or float): The original price of the item.
        discount (int or float): The discount percentage to
apply.

    Returns:
        int or float: The final price after discount is
deducted.

    Example:
        >>> calculate_discount(100, 20)
        80.0
        >>> calculate_discount(50, 10)
        45.0
    """
    final_price = price - (price * discount / 100)
    return final_price


def simple_interest(principal, rate, time):
    """Calculates simple interest on a given principal amount.

    Args:
        principal (int or float): The initial amount of money.
        rate (int or float): The annual interest rate (as a
percentage).
        time (int or float): The time period in years.

    Returns:
        int or float: The calculated simple interest.

    Example:
        >>> simple_interest(1000, 5, 2)
        100.0
        >>> simple_interest(500, 10, 3)
        150.0
    """
    interest = (principal * rate * time) / 100
    return interest
```

Task-05: Building a Mini Automatic Documentation Generator

Code:

```
import ast

class DocstringGenerator(ast.NodeVisitor):
    def __init__(self):
        self.items = []

    def visit_FunctionDef(self, node):
        # Only add docstring if it does not already exist
        if not ast.get_docstring(node):
            self.items.append({
                'type': 'function',
                'name': node.name,
                'lineno': node.lineno,
                'args': [arg.arg for arg in node.args.args]
            })
        self.generic_visit(node)

    def visit_ClassDef(self, node):
        # Only add docstring if it does not already exist
        if not ast.get_docstring(node):
            self.items.append({
                'type': 'class',
                'name': node.name,
                'lineno': node.lineno
            })
        self.generic_visit(node)


def generate_google_docstring(item):
    if item['type'] == 'function':

        if item['args']:
            args_section = "\n".join(
                [f"        {arg} (type): Description." for arg
in item['args']]
            )
        else:
            args_section = "        None"

        docstring = f'''    """
    {item["name"]} description.

    Args:
{args_section}

    Returns:
        type: Description.
    """'''

    else:
        docstring = f'''    """
    {item["name"]} class description.
    Attributes:
        attr (type): Description.
    """'''

    return docstring


def insert_docstrings(input_file, output_file):
    with open(input_file, 'r') as f:
        content = f.read()

    tree = ast.parse(content)
    generator = DocstringGenerator()
    generator.visit(tree)

    lines = content.split('\n')
    offset = 0

    # Sort by line number to insert correctly
    for item in sorted(generator.items, key=lambda x:
x['lineno']):
        insert_line = item['lineno'] - 1 + offset
        docstring = generate_google_docstring(item)

        lines.insert(insert_line + 1, docstring)
        offset += docstring.count('\n') + 1

    with open(output_file, 'w') as f:
        f.write('\n'.join(lines))

    print(f"✅ Documentation scaffolding added.")
    print(f"📄 Output saved to: {output_file}")


if __name__ == "__main__":
    insert_docstrings('Lab-9.4.py', 'Lab-9.4_documented.py')
```

Output:



```
PS C:\Users\monic\Downloads\AI Assisted Coding> python Lab-9.4.py
>>
✅ Documentation scaffolding added.
📄 Output saved to: Lab-9.4_documented.py
PS C:\Users\monic\Downloads\AI Assisted Coding>
```

Lab-9.4_documented.py

```
#Task-01: Auto-Generating Function Documentation in a Shared
Codebase
"""Adds two numbers together.
Args:
    a (int or float): The first number to add.
    b (int or float): The second number to add.
Returns:
    int or float: The sum of a and b.
Example:
    >>> add_numbers(5, 3)
    8
    >>> add_numbers(2.5, 1.5)
    4.0
"""
"""Calculates the factorial of a non-negative integer using
recursion.
Args:
    n (int): A non-negative integer for which to calculate the
factorial.
Returns:
    int: The factorial of n (n!).
Raises:
    RecursionError: If n is negative (will cause infinite
recursion).
Example:
    >>> factorial(5)
    120
    >>> factorial(0)
    1
"""
"""Checks whether a given number is a prime number.
Args:
    num (int): The number to check for primality.
Returns:
    bool: True if num is a prime number, False otherwise.
Example:
    >>> is_prime(7)
    True
    >>> is_prime(10)
    False
    >>> is_prime(1)
    False
"""
def add_numbers(a, b):
    """
    add_numbers description.
    Args:
        a (type): Description.
        b (type): Description.

    Returns:
        type: Description.
    """
    return a + b

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

def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
    """
    factorial description.

    Args:
        n (type): Description.

    Returns:
        type: Description.
    """
            return False
    return True



#Task-02: Enhancing Readability Through AI-Generated Inline
Comments
def fibonacci(n):
    if n <= 0:
        return []  # Return empty list for invalid input

    elif n == 1:
        return [0]  # Base case: first Fibonacci number

    """
    is_prime description.

    Args:
```

```python
        num (type): Description.

    Returns:
        type: Description.
    """
    seq = [0, 1]  # Initialize sequence with first two numbers

    for i in range(2, n):
        # Each number is the sum of the previous two numbers
        seq.append(seq[i - 1] + seq[i - 2])

    return seq


#Task-03: Generating Module-Level Documentation for a Python Package
import math

def area_circle(radius):
    return math.pi * radius * radius

def area_square(side):
    return side * side
    """
    Geometry Calculation Module
    """
    fibonacci description.

    Args:
        n (type): Description.

    Returns:
        type: Description.
    """

    This module provides utility functions for calculating the areas of basic geometric shapes.
    It serves as a lightweight reference implementation for fundamental geometry formulas.

    Dependencies:
        - math: Standard library module for mathematical constants and operations.

    Key Functions:
        - area_circle(radius): Calculates the area of a circle given its radius.
        - area_square(side): Calculates the area of a square
given its side length.

    Example Usage:
        >>> from lab_9_4 import area_circle, area_square
        >>> area_circle(5)
        78.53981633974483
        >>> area_square(4)
        16
    """




#Task-04: Converting Developer Comments into Structured
Docstrings
def calculate_discount(price, discount):
    """Calculates the final price after applying a discount.

    Args:
    """
    area_circle description.

    Args:
        radius (type): Description.

    Returns:
        type: Description.
    """
        price (int or float): The original price of the item.
        discount (int or float): The discount percentage to
apply.

    Returns:
        int or float: The final price after discount is
deducted.

    Example:
        >>> calculate_discount(100, 20)
        80.0
        >>> calculate_discount(50, 10)
        45.0
    """
    area_square description.

    Args:
        side (type): Description.
    Returns:
        type: Description.
    """
    """
    final_price = price - (price * discount / 100)
    return final_price


def simple_interest(principal, rate, time):
    """Calculates simple interest on a given principal amount.

    Args:
        principal (int or float): The initial amount of money.
        rate (int or float): The annual interest rate (as a
percentage).
        time (int or float): The time period in years.

    Returns:
        int or float: The calculated simple interest.

    Example:
        >>> simple_interest(1000, 5, 2)
        100.0
        >>> simple_interest(500, 10, 3)
        150.0
    """
    interest = (principal * rate * time) / 100
    return interest



# Task-05: Building a Mini Automatic Documentation Generator

import ast


class DocstringGenerator(ast.NodeVisitor):
    def __init__(self):
        self.items = []

    def visit_FunctionDef(self, node):
        # Only add docstring if it does not already exist
        if not ast.get_docstring(node):
            self.items.append({
                'type': 'function',
                'name': node.name,
                'lineno': node.lineno,
                'args': [arg.arg for arg in node.args.args]
            })
        self.generic_visit(node)

    def visit_ClassDef(self, node):
        # Only add docstring if it does not already exist
        if not ast.get_docstring(node):
            self.items.append({
                'type': 'class',
                'name': node.name,
                'lineno': node.lineno
            })
        self.generic_visit(node)


def generate_google_docstring(item):
    if item['type'] == 'function':

        if item['args']:
            args_section = "\n".join(
                [f"        {arg} (type): Description." for arg
in item['args']]
            )
        else:
            args_section = "        None"

        docstring = f'''    """
    {item["name"]} description.

    Args:
{args_section}

    Returns:
        type: Description.
    """'''

    else:
        docstring = f'''    """
    {item["name"]} class description.

    """
    DocstringGenerator class description.

    Attributes:
        attr (type): Description.
    """
    Attributes:
        attr (type): Description.
    """'''

    return docstring

    """
    __init__ description.

    Args:
        self (type): Description.

    Returns:
        type: Description.
    """

def insert_docstrings(input_file, output_file):
    with open(input_file, 'r') as f:
        content = f.read()

    tree = ast.parse(content)
    generator = DocstringGenerator()
    generator.visit(tree)

    lines = content.split('\n')
    offset = 0
    """
    visit_FunctionDef description.

    Args:
        self (type): Description.
        node (type): Description.

    Returns:
        type: Description.
    """

    # Sort by line number to insert correctly
    for item in sorted(generator.items, key=lambda x:
x['lineno']):
        insert_line = item['lineno'] - 1 + offset
        docstring = generate_google_docstring(item)

        lines.insert(insert_line + 1, docstring)
        offset += docstring.count('\n') + 1

    with open(output_file, 'w') as f:
        f.write('\n'.join(lines))

    print(f"✅ Documentation scaffolding added.")
    print(f"📄 Output saved to: {output_file}")


if __name__ == "__main__":
    insert_docstrings('Lab-9.4.py', 'Lab-9.4_documented.py')

    """
    visit_ClassDef description.

    Args:
        self (type): Description.
        node (type): Description.

    Returns:
        type: Description.
    """
    """
    generate_google_docstring description.

    Args:
        item (type): Description.

    Returns:
        type: Description.
    """
    """
    insert_docstrings description.

    Args:
        input_file (type): Description.
        output_file (type): Description.

    Returns:
        type: Description.
    """
```