

AI-ASSITED

ASSIGNMENT -9.2

ROLLNO:2403A52084

BATCH:04

TASK01:

PROMPT:

(Documentation – Google-Style Docstrings for Python Functions)

- Task: Use AI to add Google-style docstrings to all functions in a given Python script.

- Instructions:

- o Prompt AI to generate docstrings without providing any input-output examples.

- o Ensure each docstring includes:

- Function description
 - Parameters with type hints
 - Return values with type hints
 - Example usage

- o Review the generated docstrings for accuracy and formatting.

- Expected Output #1:

- o A Python script with all functions documented using correctly formatted Google-style docstrings

CODE:

```
def square(n: int) -> int:
    """
    Returns the square of an integer.

    Args:
        n (int): The integer to be squared.

    Returns:
        int: The square of the input integer.

    Example:
        result = square(5)
    """
    return n * n

if __name__ == "__main__":
    print(square(5))
```

OUTPUT:

```
PS C:\Users\NIKHITHA\OneDrive\Desktop\AI> python -u "c:\Users\NIKHITHA\OneDrive\Desktop\AI\LAB-9.4\task1"
25
PS C:\Users\NIKHITHA\OneDrive\Desktop\AI> python -u "c:\Users\NIKHITHA\OneDrive\Desktop\AI\LAB-9.4\task1"
```

OBSERVATOIN:

- The script defines a function `square` that takes an integer `n` and returns its square (`n * n`).
- The function is documented with a Google-style docstring, clearly describing its purpose, parameters, return value, and an example usage.
- In the main block (`if __name__ == "__main__":`), the function is called with the argument 5, and the result (25) is printed.
- The code is simple, readable, and demonstrates good documentation practices for small utility functions.

TASK02:

PROMPT: Add meaningful inline comments to the following Python program, focusing only on parts where the logic may not be immediately obvious.

Skip basic syntax explanations and comment only on sections that could be tricky or non-intuitive for someone reading the code.

Ensure the comments help improve readability and understanding of the code.

```
def find_unique_numbers(numbers: list[int]) -> list[int]:  
    """  
    Returns a list of unique integers from the input list.  
  
    Args:  
    | numbers (list[int]): List of integers.  
  
    Returns:  
    | list[int]: List of unique integers.  
  
    Example:  
    | result = find_unique_numbers([1, 2, 2, 3])  
    """  
    result = []  
    seen = set()  
    for num in numbers:  
        if num not in seen:  
            result.append(num)  
            seen.add(num)  
    return result  
  
if __name__ == "__main__":  
    print(find_unique_numbers([1, 2, 2, 3, 4, 4, 5]))
```

OUTPUT:

```
PS C:\Users\WIKHITHA\OneDrive\Desktop\AI> python -u "c:\Users\WIKHITHA\OneDrive\Desktop\AI\LAB-9.4\task2"  
[1, 2, 3, 4, 5]  
PS C:\Users\WIKHITHA\OneDrive\Desktop\AI>
```

OBSERVATION: The `find_unique_numbers` function efficiently extracts unique integers from a list while preserving their order using a set for fast lookups. The code is

concise,readable, and demonstrates a common pattern for deduplication in Python.

TASK03:

PROMPT:

Write a concise module-level docstring for the following Python file, summarizing its purpose, dependencies, and main functions.

Place the docstring at the top of the file and do not rewrite the code.

CODE:

```
4 / ... task3 / ...
✓ """
  This module provides a simple utility function for mathematical operations.

  Main Functions:
  | - square: Returns the square of an integer.

  Dependencies:
  | - No external dependencies; uses standard Python.

  Usage:
  | Call the square function with an integer argument to get its square.
  """

✓ def square(n: int) -> int:
  |     return n * n

✓ if __name__ == "__main__":
  |     print(square(5))
```

OUTPUT:

```
PS C:\Users\WIKHITHA\OneDrive\Desktop\AI> python -u "c:\Users\WIKHITHA\OneDrive\Desktop\AI\LAB-9.4\task3"
25
PS C:\Users\WIKHITHA\OneDrive\Desktop\AI> █
```

OBSERVATION: This code defines a well-documented module with a single function, `square`, that calculates the square of an integer. It is simple, easy to use, and demonstrates good practice by including a module-level docstring and a main block for direct execution.

TASK04:

PROMPT:

Transform the following Python code by moving all relevant inline comments into structured Google-style function docstrings. Replace the comments with clear, standardized docstrings that documentation quality.

CODE:

```
def factorial(n: int) -> int:
    """
    Calculates the factorial of a non-negative integer using recursion.

    Args:
        n (int): The non-negative integer to calculate the factorial of.

    Returns:
        int: The factorial of the input integer. Returns 1 if n is 0 or 1.

    Example:
        result = factorial(5)
    """
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

def is_even(number: int) -> bool:
    """
    Checks if the given number is even.

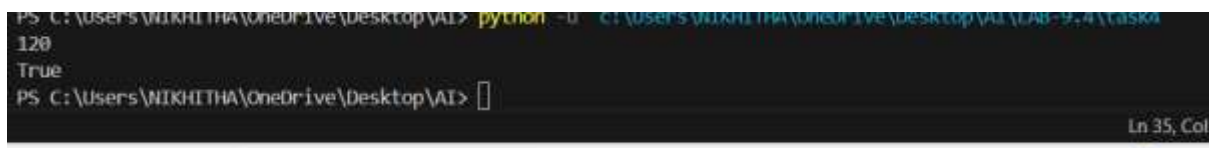
    Args:
        number (int): The integer to check.

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

    Example:
        result = is_even(4)
    """
    return number % 2 == 0

if __name__ == "__main__":
    print(factorial(5))
    print(is_even(4))
```

OUTPUT:



```
PS C:\Users\WIKHITHA\OneDrive\Desktop\AI> python -u C:\Users\WIKHITHA\OneDrive\Desktop\AI\QAB-9-4\task4.py
120
True
PS C:\Users\WIKHITHA\OneDrive\Desktop\AI> 
```

OBSEVATOIN:

This code defines two utility functions: [factorial](#) for calculating the factorial of a non-negative integer using recursion, and [is_even](#) for checking if a number is even.

Both functions are well-documented with Google-style docstrings, and the script demonstrates their usage in the main block.

TASK5:

PROMPT:

Review the following Python code and correct any outdated or inaccurate docstrings.

Rewrite each docstring to accurately reflect the current code behavior, using Google-style formatting.

CODE:

```
def add(a: int, b: int) -> int:
    """
    Adds two integers and returns the result.

    Args:
        a (int): The first integer.
        b (int): The second integer.

    Returns:
        int: The sum of a and b.

    Example:
        result = add(2, 3)
    """
    return a + b

if __name__ == "__main__":
    print(add(2, 3))
```

OUTPUT:

```
PS C:\Users\WIKHITHA\OneDrive\Desktop\AI> python -u "c:\Users\WIKHITHA\OneDrive\Desktop\AI\LAB-9.4\task5"
5
PS C:\Users\WIKHITHA\OneDrive\Desktop\AI> 
```

OBSERVATION:

This code defines a simple, well-documented function `add` that returns the sum of two integers.

It follows Google-style docstring conventions and includes a main block to demonstrate usage by printing the result of `add(2, 3)`.

TASK:06

PROMPT: Compare documentation output from a vague prompt ("Add comments to this function") and a detailed prompt ("Add Google-style docstrings with parameters, return types, and examples") for the same Python function.

Show the results and analyze the differences in a comparison table.

CODE:

```
# task6 > ...

# Vague prompt output
def add(a: int, b: int) -> int:
    # Adds two numbers
    return a + b

# Detailed prompt output
def add(a: int, b: int) -> int:
    """
    Adds two integers and returns the result.

    Args:
        a (int): The first integer.
        b (int): The second integer.

    Returns:
        int: The sum of a and b.

    Example:
        result = add(2, 3)
    """
    return a + b

# Comparison Table
print("| Prompt      | Output                                     | Observations")
print("|-----|-----|-----|")
print("| Vague      | # Adds two numbers          | Brief, lacks detail and structure |")
print("| Detailed   | Google-style docstring      | Complete, clear, standardized    |")
```


OUTPUT:

```
PS C:\Users\NIKHITHA\OneDrive\Desktop\AI> python -u "c:\Users\NIKHITHA\OneDrive\Desktop\AI\LAB-9.4\task6"
| Prompt | Output | Observations |
|-----|-----|-----|
| Vague | # Adds two numbers | Brief, lacks detail and structure |
| Detailed | Google-style docstring | Complete, clear, standardized |
PS C:\Users\NIKHITHA\OneDrive\Desktop\AI>
```

OBSERVATION:

This code demonstrates the difference between a vague comment and a detailed Google-style docstring for the same function. The comparison table clearly shows that the detailed prompt produces more complete, clear, and standardized documentation than the vague prompt.