

# 2303A51797

## Batch : 26

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

Codebase

#### Scenario

You have joined a development team where several utility functions are already implemented, but the code lacks proper documentation. New team members are struggling to understand how these functions should be used.

#### Task Description

You are given a Python script containing multiple functions without any docstrings.

Using an AI-assisted coding tool:

- Ask the AI to automatically generate Google-style function docstrings for each function
- Each docstring should include:
  - A brief description of the function
  - Parameters with data types
  - Return values
  - At least one example usage (if applicable)

Experiment with different prompting styles (zero-shot or context-based) to observe quality differences.

#### Expected Outcome

- A Python script with well-structured Google-style docstrings
  - Docstrings that clearly explain function behavior and usage
- Improved readability and usability of the codebase give code for this question for vs code

```

File Edit Selection View Go Run Terminal Help ← → SOURCE CONTROL
new.py •
C: > Users > Chinnari > Downloads > new.py > reverse_string
1 def add(a, b):
2     return a + b
3 def divide(a, b):
4     if b == 0:
5         raise ValueError("Division by zero is not allowed")
6     return a / b
7
8 def is_even(number):
9     return number % 2 == 0
10
11 def factorial(n):
12     if n < 0:
13         raise ValueError("Negative numbers are not allowed")
14     if n == 0 or n == 1:
15         return 1
16     return n * factorial(n - 1)
17 def reverse_string(text):
18     return text[::-1]
19
20 print("Addition:", add(5, 3))
21 print("Division:", divide(10, 2))
22 print("Is 4 even?", is_even(4))
23 print("Factorial of 5:", factorial(5))
24 print("Reverse of 'hello':", reverse_string("hello"))
25

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Chinnari\Downloads\Devops> **Addition:** 8  
**>> Division:** 5.0  
**>> Is 4 even?** True  
**>> Factorial of 5:** 120  
**>> Reverse of 'hello':** olleh  
**>> [ ]**

Ln 18, Col 22 Spaces: 4 UTF-8 CRLF { } Python powershell 3.13.12 (Microsoft Store) Launch Program (Devops)

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

### Comments

#### Scenario

A Python program contains complex logic that works correctly but is difficult to understand at first glance. Future maintainers may find it hard to debug or extend this code.

#### Task Description

You are provided with a Python script containing:

- Loops
- Conditional logic
- Algorithms (such as Fibonacci sequence, sorting, or searching)

Use AI assistance to:

- 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.

#### Expected Outcome

- A Python script with concise, meaningful inline comments
- Comments that explain why the logic exists, not what Python syntax does

- Noticeable improvement in code readability i want code for this with output

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows a folder structure with a file named "new.py".
- Source Control:** Shows a message about initializing a Git repository.
- Editor:** Displays Python code for Fibonacci sequence generation and bubble sort, along with a binary search function.
- Terminal:** Shows the output of running the script, including the Fibonacci sequence [0, 1, 1, 2, 3, 5, 8] and the sorted bubble sort array [64, 34, 25, 12, 22, 11, 98].
- Status Bar:** Shows file statistics (0△0), Python debugger status, and other settings.

```

1 def fibonacci(n):
2     sequence = []
3     a, b = 0, 1
4
5     for _ in range(n):
6         sequence.append(a)
7         a, b = b, a + b
8
9     return sequence
10 def bubble_sort(arr):
11     n = len(arr)
12
13     for i in range(n):
14         for j in range(0, n - i - 1):
15             if arr[j] > arr[j + 1]:
16                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
17
18     return arr
19 def binary_search(arr, target):
20     left, right = 0, len(arr) - 1
21
22     while left <= right:
23         mid = (left + right) // 2
24         if arr[mid] == target:
25             return mid
26         elif arr[mid] < target:
27             left = mid + 1
28         else:
29             right = mid - 1
30
31     return -1
32
33 if __name__ == "__main__":
34
35     print("\nFibonacci (first 7 numbers):")
36     print(fibonacci(7))
37
38     print("\nBubble Sort:")
39     numbers = [64, 34, 25, 12, 22, 11, 98]
40     print("Before sorting:", numbers)
41     sorted_numbers = bubble_sort(numbers.copy())
42     print("After sorting:", sorted_numbers)
43
44     print("\nBinary Search:")
45     index = binary_search(sorted_numbers, 25)
46     print("Element 25 found at index:", index)

```

This screenshot shows the same setup as the first one, but with additional print statements added to the binary search section:

```

1 def fibonacci(n):
2     sequence = []
3     a, b = 0, 1
4
5     for _ in range(n):
6         sequence.append(a)
7         a, b = b, a + b
8
9     return sequence
10 def bubble_sort(arr):
11     n = len(arr)
12
13     for i in range(n):
14         for j in range(0, n - i - 1):
15             if arr[j] > arr[j + 1]:
16                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
17
18     return arr
19 def binary_search(arr, target):
20     left, right = 0, len(arr) - 1
21
22     while left <= right:
23         mid = (left + right) // 2
24         if arr[mid] == target:
25             return mid
26         elif arr[mid] < target:
27             left = mid + 1
28         else:
29             right = mid - 1
30
31     return -1
32
33 if __name__ == "__main__":
34
35     print("\nFibonacci (first 7 numbers):")
36     print(fibonacci(7))
37
38     print("\nBubble Sort:")
39     numbers = [64, 34, 25, 12, 22, 11, 98]
40     print("Before sorting:", numbers)
41     sorted_numbers = bubble_sort(numbers.copy())
42     print("After sorting:", sorted_numbers)
43
44     print("\nBinary Search:")
45     index = binary_search(sorted_numbers, 25)
46     print("Element 25 found at index:", index)

```

The terminal output now includes the binary search result: "Element 25 found at index: 3".

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

#### Package

#### Scenario

Your team is preparing a Python module to be shared internally (or uploaded to a repository). Anyone opening the file should immediately understand its purpose and structure.

#### Task Description

Provide a complete Python module to an AI tool and instruct it to automatically generate a module-level docstring at the top of the file that includes:

- The purpose of the module
- Required libraries or dependencies
- A brief description of key functions and classes
- A short example of how the module can be used

Focus on clarity and professional tone

#### Expected Outcome

- A well-written multi-line module-level docstring
- Clear overview of what the module does and how to use it
- Documentation suitable for real-world projects or repositories

```
newpy
C:\Users\Chinnari\Downloads> newpy > ...
1 class BankAccount:
2     """Represents a simple bank account."""
3
4     def __init__(self, owner, balance=0):
5         self.owner = owner
6         self.balance = balance
7
8     def deposit(self, amount):
9         if amount < 0:
10             self.balance += amount
11             return f"({amount}) deposited successfully."
12         return "Invalid deposit amount."
13
14     def withdraw(self, amount):
15         if 0 < amount <= self.balance:
16             self.balance -= amount
17             return f"({amount}) withdrawn successfully."
18         return "Insufficient balance or invalid amount."
19
20     def get_balance(self):
21         return self.balance
22
23 if __name__ == "__main__":
24     account = BankAccount("Soumya", 1000)
25
26     print("Initial Balance:", account.get_balance())
27     print(account.deposit(500))
28     print("Balance after deposit:", account.get_balance())
29     print(account.withdraw(300))
30     print("Final Balance:", account.get_balance())
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Chinnari\Downloads\Devops> c;; cd 'c:\Users\Chinnari\Downloads\Devops' & 'c:\Users\Chinnari\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\Chinnari\Downloads\newpy'
Initial Balance: 1000
500 deposited successfully.
Balance after deposit: 1500
300 withdrawn successfully.
Final Balance: 1200
PS C:\Users\Chinnari\Downloads\Devops>
```

#### Task 4: Converting Developer Comments into Structured Docstrings

##### Scenario

In a legacy project, developers have written long explanatory comments inside functions instead of proper docstrings. The team now wants to standardize documentation.

##### Task Description

You are given a Python script where functions contain detailed inline comments explaining their logic.

Use AI to:

- Automatically convert these comments into structured Google-style or NumPy-style docstrings
- Preserve the original meaning and intent of the comments
- Remove redundant inline comments after conversion

Expected Outcome

- Functions with clean, standardized docstrings
- Reduced clutter inside function bodies
- Improved consistency across the codebase i want code for this with output

The screenshot shows the VS Code interface with a Python file named `new.py` open. The code contains functions for calculating discounts and finding the largest number in a list. The terminal below shows the execution of the script, including the calculation of a discounted price and the determination of the largest number from a list of three integers.

```
C:\> Users> Chinnari> Downloads > new.py >...
C:\Users\Chinnari\Downloads> new.py
  def calculate_discount(price, discount_percent):
    if price < 0 or discount_percent < 0:
        raise ValueError("Price and discount must be non-negative")
    discount_amount = (price * discount_percent) / 100
    final_price = price - discount_amount
    return final_price

  def find_largest(numbers):
    if not numbers:
        raise ValueError("List cannot be empty")
    largest = numbers[0]
    for num in numbers:
        if num > largest:
            largest = num
    return largest

if __name__ == "__main__":
    print("Discounted Price:", calculate_discount(1000, 10))
    print("Largest Number:", find_largest([3, 7, 2, 9]))
```

```
problems output debug console terminal ports
r1\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60716' ... 'C:\Users\Chinnari\Downloads\new.py'
Info: Starting process: C:\Users\Chinnari\Downloads\new.py
● 300 withdraw successfully.
Final Balance: 1000
PS C:\Users\Chinnari\Downloads\Devops> c:\ cd 'c:\Users\Chinnari\Downloads\Devops' & 'c:\Users\Chinnari\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Chinnari\Downloads\new.py'
r1\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63528' ... 'C:\Users\Chinnari\Downloads\new.py'
○ Discounted Price: 900.0
Largest Number: 9
PS C:\Users\Chinnari\Downloads\Devops>
```

## Task 5: Building a Mini Automatic Documentation Generator

Scenario

Your team wants a simple internal tool that helps developers start documenting new Python files quickly, without writing documentation from scratch.

Task Description

Design a small Python utility that:

- Reads a given .py file
- Automatically detects:

o Functions

o Classes

- Inserts placeholder Google-style docstrings for each detected function or class

AI tools may be used to assist in generating or refining this utility.

Note: The goal is documentation scaffolding, not perfect

documentation.

## Expected Outcome

- A working Python script that processes another .py file
  - Automatically inserted placeholder docstrings
  - Clear demonstration of how AI can assist in documentation

automation i want code for this with output

A screenshot of the Microsoft DevOps IDE interface. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar for 'DevOps'. The left sidebar has a 'SOURCE CONTROL' section with a note about initializing a repository and a 'Initialize Repository' button. Below it is a note about publishing code to GitHub. The main area shows a code editor with Python code for a 'calculator' class. The terminal at the bottom shows the command to run the code and the output of the documentation scaffolding. The status bar at the bottom right shows the file path as 'C:\Users\Chilemar\Downloads\DevOps\calculator.py' and other system information.