2303A51940

Batch : 25

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:

o A brief description of the function

o Parameters with data types

o Return values

o At least one example usage (if applicable)

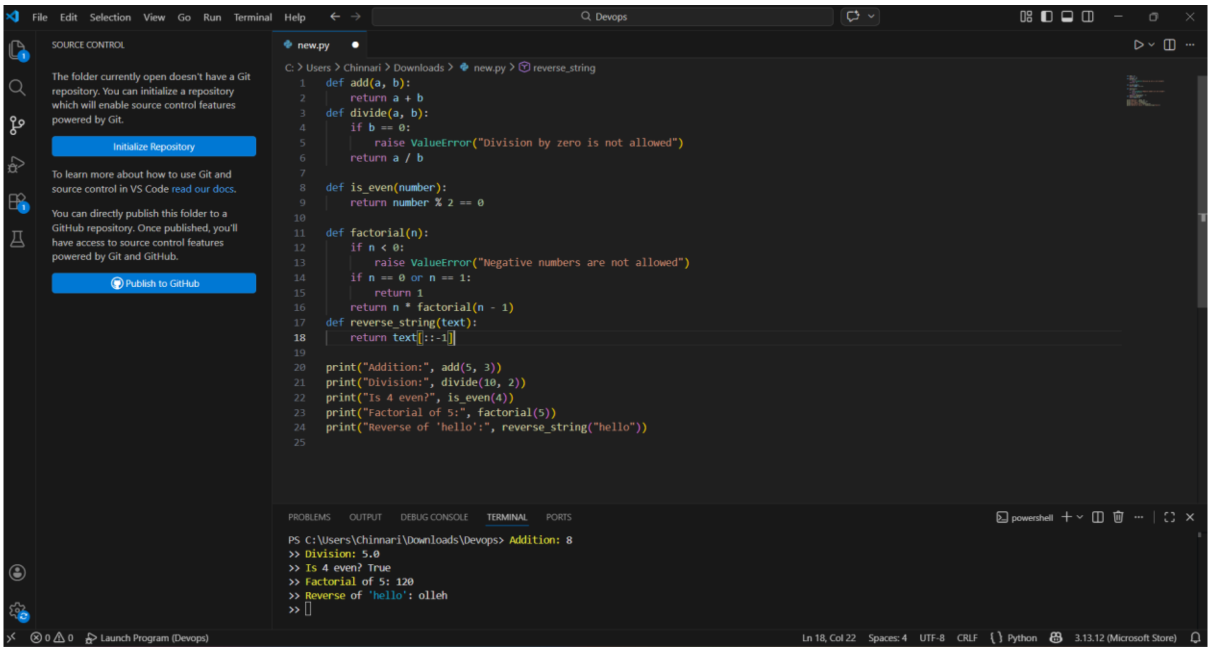
Experiment with different prompting styles (zero-shot or context-based)

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 ths question for vs code


File Edit Selection View Go Run Terminal Help
Devops
SOURCE CONTROL
Initialize Repository
Publish to GitHub
new.py
C: > Users > Chinnari > Downloads > new.py > reverse_string
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Launch Program (Devops)
Ln 18, Col 22 Spaces: 4 UTF-8 CRLF Python 3.13.12 (Microsoft Store)

The folder currently open doesn't have a Git repository. You can initialize a repository which will enable source control features powered by Git.

To learn more about how to use Git and source control in VS Code read our docs.

You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by Git and GitHub.

```python
def add(a, b):
    return a + b
def divide(a, b):
    if b == 0:
        raise ValueError("Division by zero is not allowed")
    return a / b

def is_even(number):
    return number % 2 == 0

def factorial(n):
    if n < 0:
        raise ValueError("Negative numbers are not allowed")
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)
def reverse_string(text):
    return text[::-1]

print("Addition:", add(5, 3))
print("Division:", divide(10, 2))
print("Is 4 even?", is_even(4))
print("Factorial of 5:", factorial(5))
print("Reverse of 'hello':", reverse_string("hello"))
```

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

Task 2: Enhancing Readability Through AI-Generated Inline

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

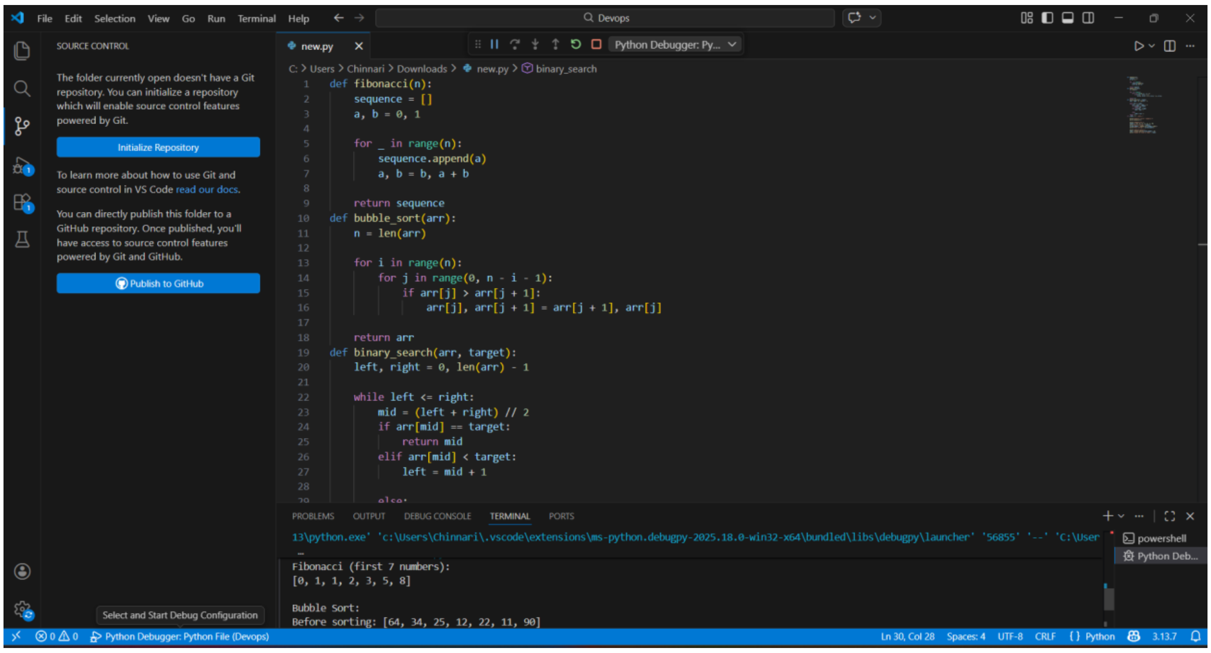
The goal is to improve clarity without cluttering the code.

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



SOURCE CONTROL

The folder currently open doesn't have a Git repository. You can initialize a repository which will enable source control features powered by Git.

Initialize Repository

To learn more about how to use Git and source control in VS Code read our docs.

You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by Git and GitHub.

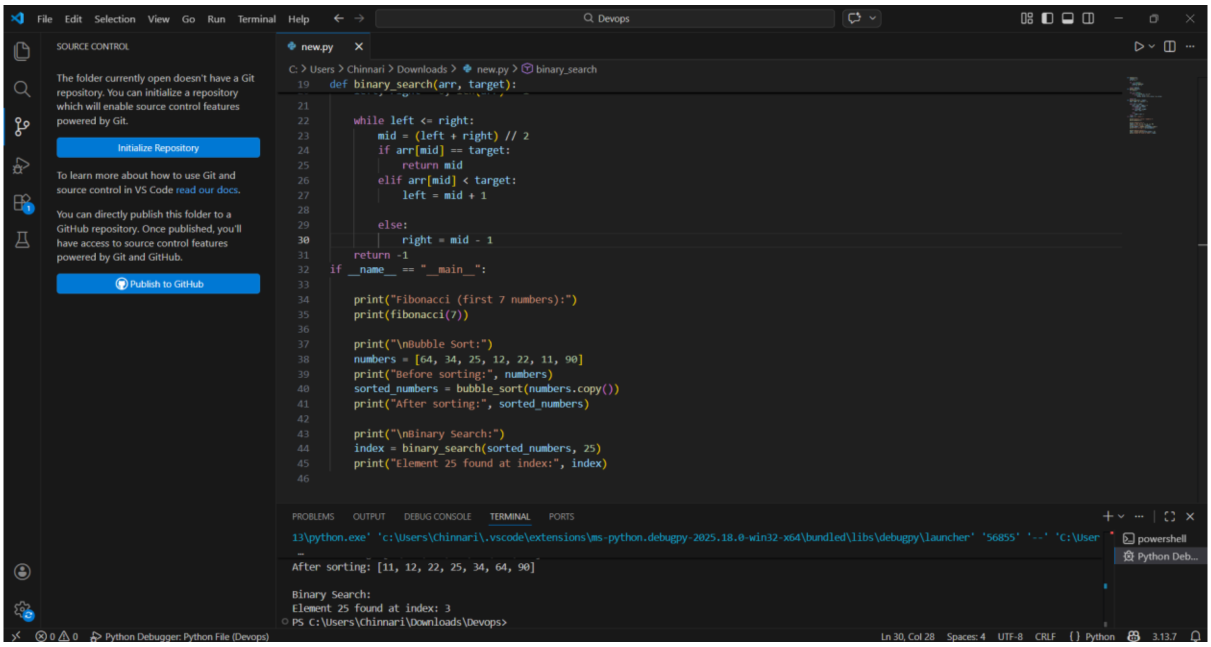
Publish to GitHub

C: > Users > Chinnari > Downloads > new.py > binary_search

```python
def fibonacci(n):
    sequence = []
    a, b = 0, 1

    for _ in range(n):
        sequence.append(a)
        a, b = b, a + b

    return sequence
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
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1

        else:
```

```
13\python.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56855' '--' 'C:\User
...
Fibonacci (first 7 numbers):
[0, 1, 1, 2, 3, 5, 8]

Bubble Sort:
Before sorting: [64, 34, 25, 12, 22, 11, 90]
```

File Edit Selection View Go Run Terminal Help
Devops
new.py
Python Debugger: Py...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
Select and Start Debug Configuration
Python Debugger: Python File (Devops)
Ln 30, Col 28 Spaces: 4 UTF-8 CRLF { } Python 3.13.7



SOURCE CONTROL

The folder currently open doesn't have a Git repository. You can initialize a repository which will enable source control features powered by Git.

Initialize Repository

To learn more about how to use Git and source control in VS Code read our docs.

You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by Git and GitHub.

Publish to GitHub

C: > Users > Chinnari > Downloads > new.py > binary_search

```python
def binary_search(arr, target):

    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1

        else:
            right = mid - 1
    return -1
if __name__ == "__main__":

    print("Fibonacci (first 7 numbers):")
    print(fibonacci(7))

    print("\nBubble Sort:")
    numbers = [64, 34, 25, 12, 22, 11, 90]
    print("Before sorting:", numbers)
    sorted_numbers = bubble_sort(numbers.copy())
    print("After sorting:", sorted_numbers)

    print("\nBinary Search:")
    index = binary_search(sorted_numbers, 25)
    print("Element 25 found at index:", index)
```

```
13\python.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56855' '--' 'C:\User
...
After sorting: [11, 12, 22, 25, 34, 64, 90]

Binary Search:
Element 25 found at index: 3
PS C:\Users\Chinnari\Downloads\Devops>
```

File Edit Selection View Go Run Terminal Help
Devops
new.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
Python Debugger: Python File (Devops)
Ln 30, Col 28 Spaces: 4 UTF-8 CRLF { } Python 3.13.7

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

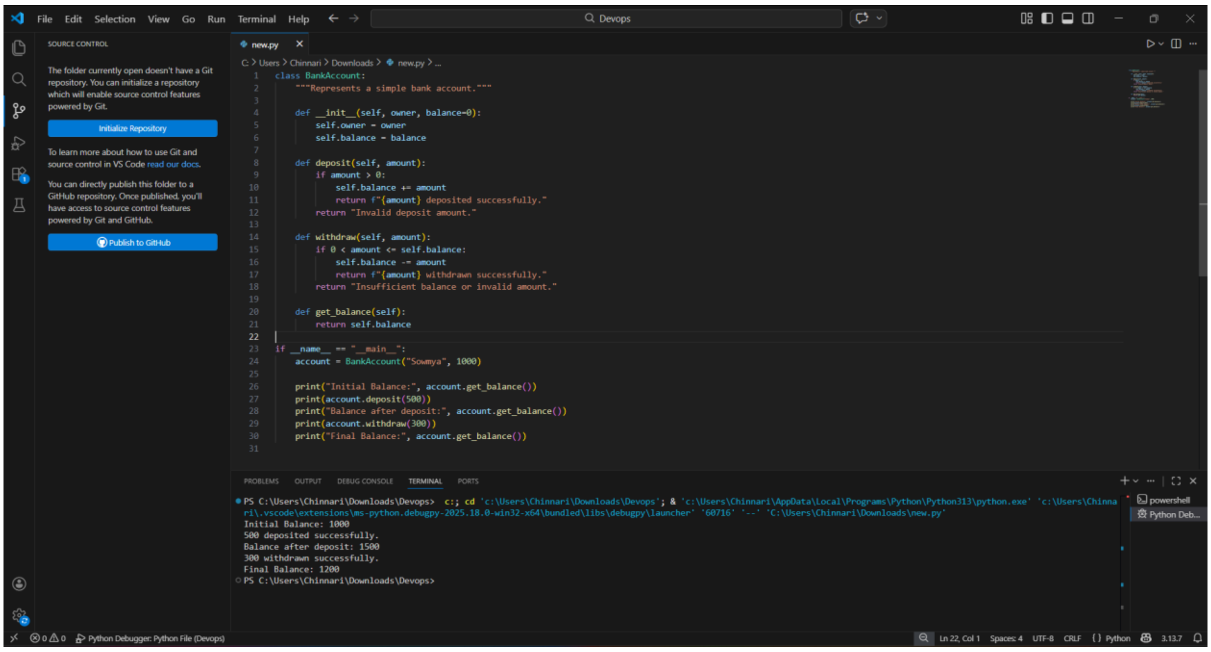
Focus on clarity and professional tone

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



```python
class BankAccount:
    """Represents a simple bank account."""

    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return f"{amount} deposited successfully."
        return "Invalid deposit amount."

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            return f"{amount} withdrawn successfully."
        return "Insufficient balance or invalid amount."

    def get_balance(self):
        return self.balance

if __name__ == "__main__":
    account = BankAccount("Sowmya", 1000)

    print("Initial Balance:", account.get_balance())
    print(account.deposit(500))
    print("Balance after deposit:", account.get_balance())
    print(account.withdraw(300))
    print("Final Balance:", account.get_balance())
```

```
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\Chinna
ri\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60716' '--' 'C:\Users\Chinnari\Downloads\new.py'
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

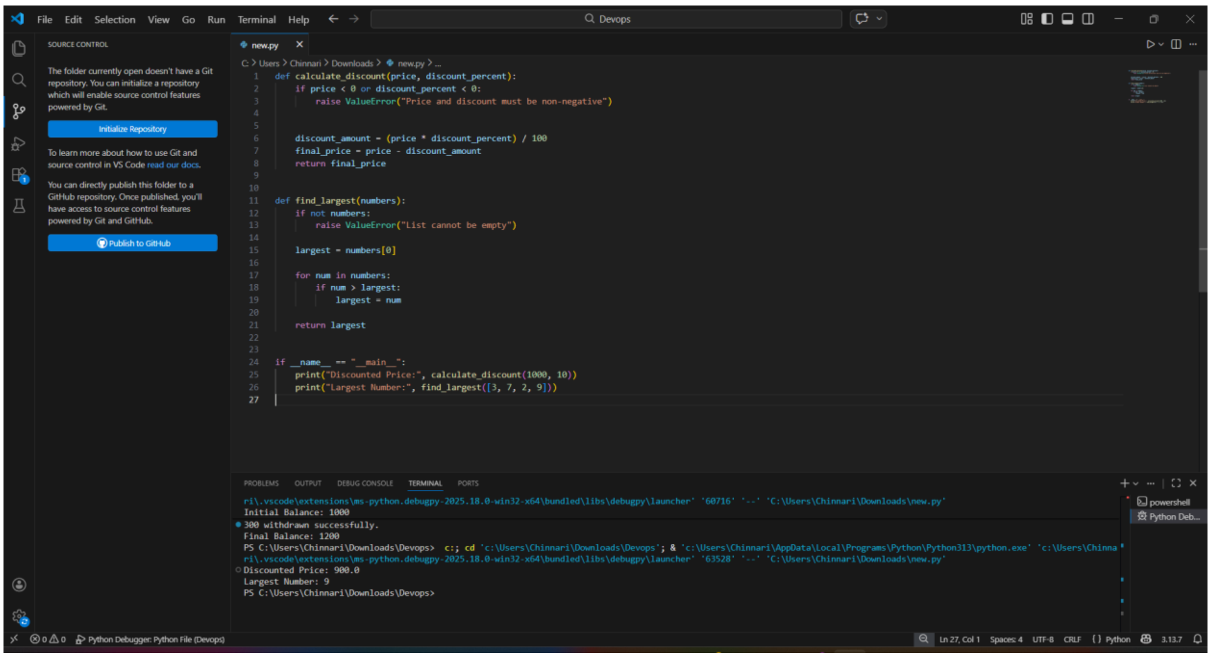
• Remove redundant inline comments after conversion

Expected Outcome

• Functions with clean, standardized docstrings

• Reduced clutter inside function bodies

• Improved consistency across the codebase i want code for this with output



```python
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]))

```

```
ri\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60716' '--' 'C:\Users\Chinnari\Downloads\new.py'
Initial Balance: 1000
300 withdrawn successfully.
Final Balance: 1200
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\Chinna
ri\.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.

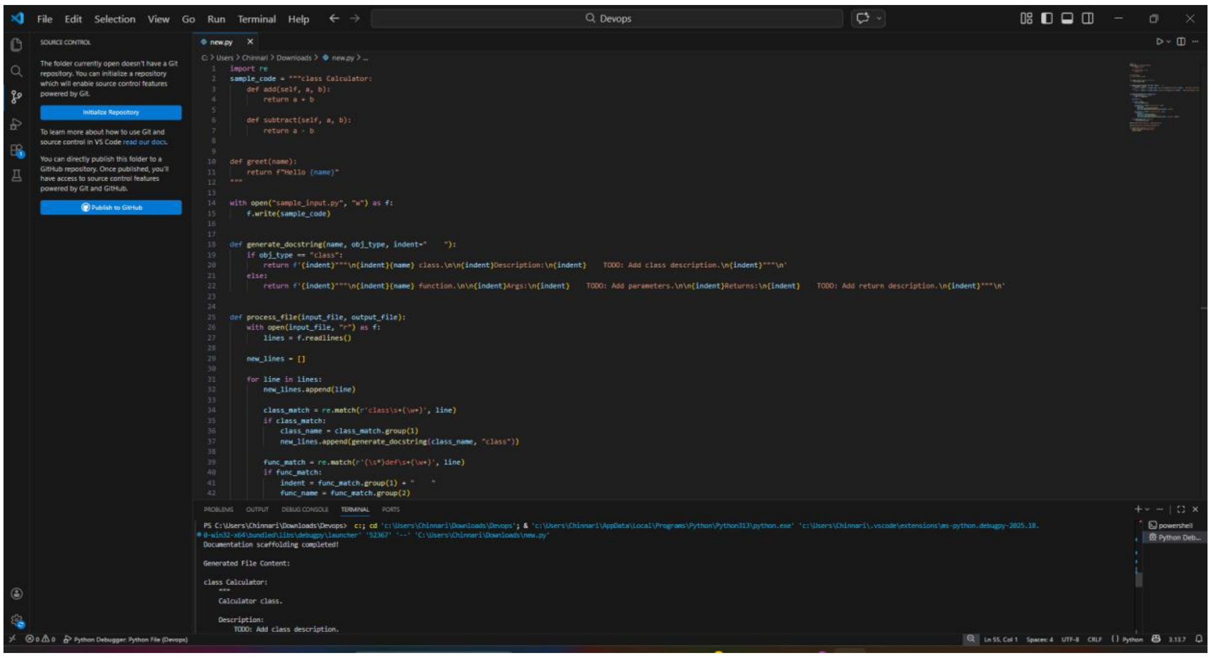
Expected Outcome

• 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



The folder currently open doesn't have a Git repository. You can initialize a repository which will enable source control features powered by Git.

To learn more about how to use Git and source control in VS Code read our docs.

You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by Git and GitHub.

```python
import re
sample_code = """class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b


def greet(name):
    return f"Hello {name}"
"""

with open("sample_input.py", "w") as f:
    f.write(sample_code)


def generate_docstring(name, obj_type, indent="    "):
    if obj_type == "class":
        return f'{indent}"""\n{indent}{name} class.\n\n{indent}Description:\n{indent}    TODO: Add class description.\n{indent}"""\n'
    else:
        return f'{indent}"""\n{indent}{name} function.\n\n{indent}Args:\n{indent}    TODO: Add parameters.\n\n{indent}Returns:\n{indent}    TODO: Add return description.\n{indent}"""\n'


def process_file(input_file, output_file):
    with open(input_file, "r") as f:
        lines = f.readlines()

    new_lines = []

    for line in lines:
        new_lines.append(line)

        class_match = re.match(r'class\s+(\w+)', line)
        if class_match:
            class_name = class_match.group(1)
            new_lines.append(generate_docstring(class_name, "class"))

        func_match = re.match(r'(\s*)def\s+(\w+)', line)
        if func_match:
            indent = func_match.group(1) + "    "
            func_name = func_match.group(2)
```

```
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\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52367' '--' 'C:\Users\Chinnari\Downloads\new.py'
Documentation scaffolding completed!

Generated File Content:

class Calculator:
    """
    Calculator class.

    Description:
        TODO: Add class description.
```