

# ASSIGNMENT – 9.2

**NAME: A. Manideepika**

**HT.NO: 2403A52052**

**BATCH: 03**

## TASK 1:

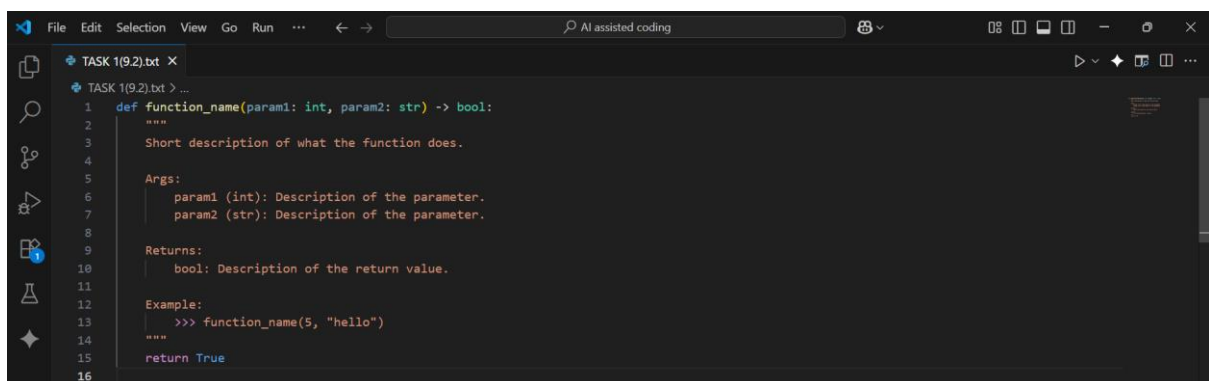
(Documentation – Google-Style Docstrings for Python Functions)

- 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.

## PROMPT:

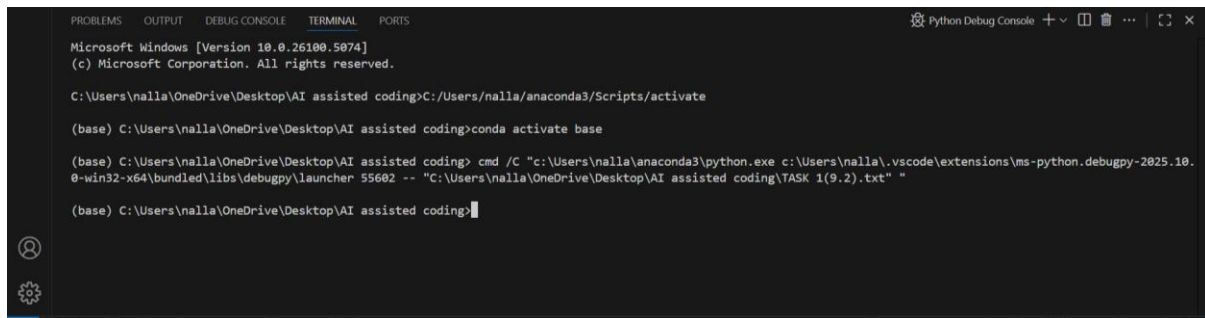
Add Google-style docstrings to all functions in a given Python script.

## CODE:

A screenshot of a code editor window with a dark theme. The title bar shows 'TASK 1(9.2).txt'. The editor contains a Python function definition. The function has two parameters, 'param1' of type 'int' and 'param2' of type 'str', and returns a 'bool'. The docstring is multi-line, starting with a short description, followed by an 'Args:' section listing the parameters with their descriptions, a 'Returns:' section with the return value description, and an 'Example:' section showing a function call. The function ends with a 'return True' statement.

```
1 def function_name(param1: int, param2: str) -> bool:
2     """
3     Short description of what the function does.
4
5     Args:
6         param1 (int): Description of the parameter.
7         param2 (str): Description of the parameter.
8
9     Returns:
10        bool: Description of the return value.
11
12     Example:
13        >>> function_name(5, "hello")
14        """
15
16     return True
```

## OUTPUT:

A screenshot of a Python Debug Console window. The window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, showing a Windows command prompt. The prompt shows the user is in a directory C:\Users\nalla\OneDrive\Desktop\AI assisted coding. They run 'C:\Users\nalla\anaconda3\Scripts\activate', then '(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base'. Then they run a command to run a Python script: '(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding> cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundle\libs\debugpy\launcher 55602 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\TASK 1(9.2).txt" "'. The prompt returns '(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>'.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Debug Console + - [ ] ... [ ] x

Microsoft Windows [Version 10.0.26100.5874]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:\Users\nalla\anaconda3\Scripts\activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding> cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundle\libs\debugpy\launcher 55602 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\TASK 1(9.2).txt" "

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
```

## OBSERVATION:

In Python, `def function_name(param1: type, param2: type) -> return_type`: is just a template for making a function. The name comes after `def`, the inputs go inside parentheses, `:` type shows each input's expected data type, and `-> return_type` shows what the function will give back. Since `type` and `return_type` were only placeholders, Python gave an error. Replacing them with real types, like `def add(a: int, b: int) -> int:`, makes it a working function that adds two integers and returns an integer.

## TASK 2:

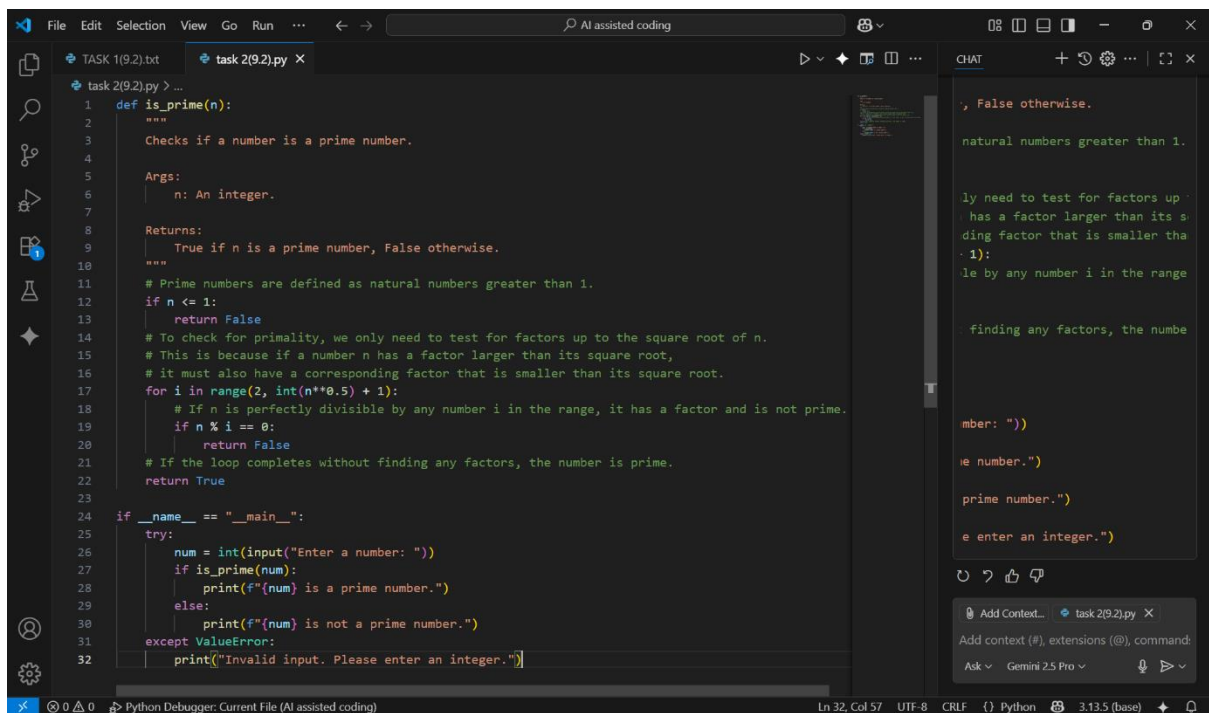
(Documentation – Inline Comments for Complex Logic)

- Task: Use AI to add meaningful inline comments to a Python program explaining only complex logic parts.
- Instructions:
  - o Provide a Python script without comments to the AI.
  - o Instruct AI to skip obvious syntax explanations and focus only on tricky or non-intuitive code sections.
  - o Verify that comments improve code readability and maintainability.

## PROMPT:

Add meaningful inline comments to a Python program explaining only complex logic parts.

## CODE:



```
1 def is_prime(n):
2     """
3     Checks if a number is a prime number.
4
5     Args:
6         n: An integer.
7
8     Returns:
9         True if n is a prime number, False otherwise.
10    """
11    # Prime numbers are defined as natural numbers greater than 1.
12    if n <= 1:
13        return False
14    # To check for primality, we only need to test for factors up to the square root of n.
15    # This is because if a number n has a factor larger than its square root,
16    # it must also have a corresponding factor that is smaller than its square root.
17    for i in range(2, int(n**0.5) + 1):
18        # If n is perfectly divisible by any number i in the range, it has a factor and is not prime.
19        if n % i == 0:
20            return False
21    # If the loop completes without finding any factors, the number is prime.
22    return True
23
24 if __name__ == "__main__":
25     try:
26         num = int(input("Enter a number: "))
27         if is_prime(num):
28             print(f"{num} is a prime number.")
29         else:
30             print(f"{num} is not a prime number.")
31     except ValueError:
32         print("Invalid input. Please enter an integer.")
```

AI Chat Window:

..., False otherwise.

natural numbers greater than 1.

ly need to test for factors up to the square root of n. This is because if a number n has a factor larger than its square root, it must also have a corresponding factor that is smaller than its square root.

finding any factors, the number is not prime.

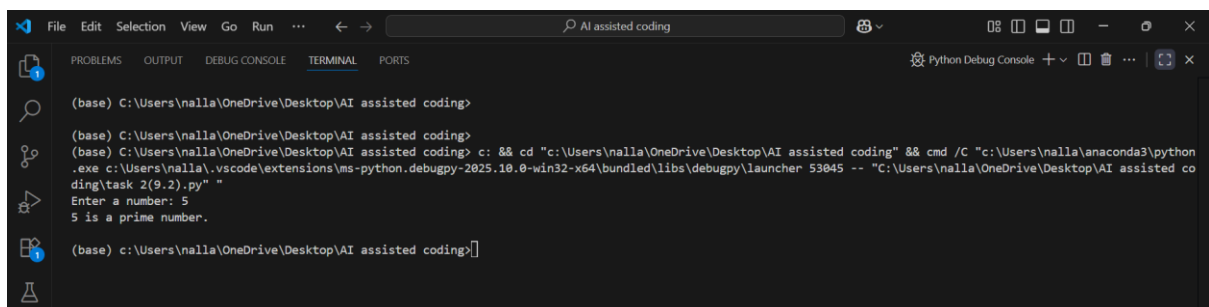
number: ")

ie number.")

prime number.")

e enter an integer.")

## OUTPUT:



```
(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding> c: && cd "c:\Users\nalla\OneDrive\Desktop\AI assisted coding" && cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 53045 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\task 2(9.2).py" "
Enter a number: 5
5 is a prime number.
(base) c:\Users\nalla\OneDrive\Desktop\AI assisted coding>]
```

## OBSERVATION:

This Python script determines if a number is prime. It defines a function `is_prime` that takes an integer `n` and first confirms it's greater than 1. The function then efficiently checks for factors by iterating only up to the square root of `n`; if any factor is found, the number is not prime. If the loop completes without finding factors, the number is confirmed to be prime. The main part of the script prompts the user for an integer, calls this function to perform the check, and prints whether the number is prime or not, while also handling cases where the input is not a valid integer. By the inline comments we easily understand the above code.

## TASK 3:

(Documentation – Module-Level Documentation)

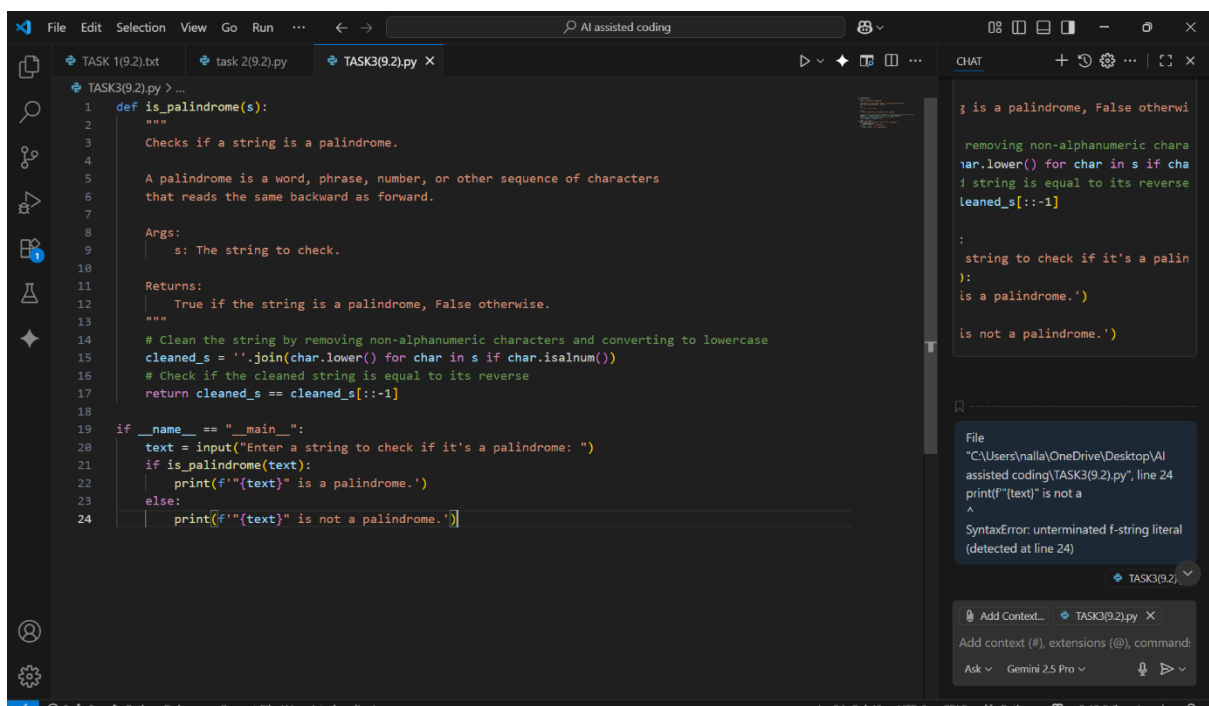
- Task: Use AI to create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file.

- Instructions:
  - o Supply the entire Python file to AI.
  - o Instruct AI to write a single multi-line docstring at the top of the file.
  - o Ensure the docstring clearly describes functionality and usage without rewriting the entire code.

## PROMPT:

Create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file.

## CODE:



The screenshot shows a VS Code editor with a Python file named `TASK3(9.2).py`. The file contains a function `is_palindrome(s)` with a multi-line docstring. The docstring describes the function's purpose, arguments, and return value. The code also includes a main block that prompts the user for a string and checks if it's a palindrome.

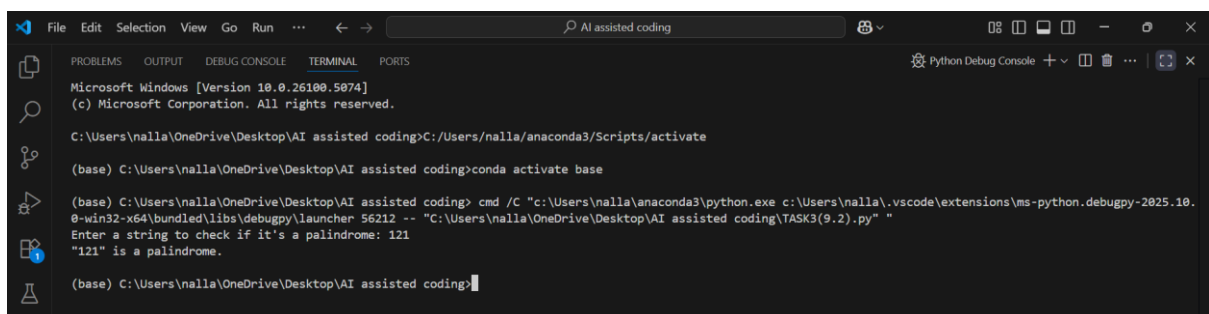
```

1 def is_palindrome(s):
2     """
3     Checks if a string is a palindrome.
4
5     A palindrome is a word, phrase, number, or other sequence of characters
6     that reads the same backward as forward.
7
8     Args:
9         s: The string to check.
10
11     Returns:
12         True if the string is a palindrome, False otherwise.
13     """
14     # Clean the string by removing non-alphanumeric characters and converting to lowercase
15     cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
16     # Check if the cleaned string is equal to its reverse
17     return cleaned_s == cleaned_s[::-1]
18
19 if __name__ == "__main__":
20     text = input("Enter a string to check if it's a palindrome: ")
21     if is_palindrome(text):
22         print(f"{text} is a palindrome.")
23     else:
24         print(f"{text} is not a palindrome.")
  
```

The chat window on the right shows the AI assistant's response to the prompt. It provides a summary of the code and identifies a syntax error in the main block of the code.

File  
 "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\TASK3(9.2).py", line 24  
 print(f"{text}") is not a  
 ^  
 SyntaxError: unterminated f-string literal  
 (detected at line 24)

## OUTPUT:



The screenshot shows a terminal window with the following output:

```

Microsoft Windows [Version 10.0.26100.5074]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:\Users\nalla\anaconda3\Scripts\activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>cmd /c "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 56212 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\TASK3(9.2).py" "
Enter a string to check if it's a palindrome: 121
"121" is a palindrome.

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
  
```

## OBSERVATION:

A module-level docstring is a documentation string placed at the very top of a Python file to provide a high-level summary of its contents. It briefly explains the module's purpose, lists any external libraries it depends on, and outlines the main functions or classes it offers. This allows other developers to quickly grasp the file's functionality and usage without reading the entire code.

## TASK 4:

(Documentation – Convert Comments to Structured Doc strings)

- Task: Use AI to transform existing inline comments into structured function docstrings following Google style.
- Instructions:
  - o Provide AI with Python code containing inline comments.
  - o Ask AI to move relevant details from comments into function docstrings.
  - o Verify that the new docstrings keep the meaning intact while improving structure.

## PROMPT:

Transform existing inline comments into structured function docstrings following Google style.

## CODE:

The screenshot shows a code editor with a Python file named 'TASK 4(9.2).py'. The code defines a function `is_armstrong(n)` that checks if a number is an Armstrong number. The original code has inline comments explaining the logic. An AI assistant (Gemini 2.5 Pro) is shown on the right, providing a prompt to transform these comments into a structured docstring. The transformed code is shown below:

```
def is_armstrong(n):
    """Checks if a number is an Armstrong number.

    An Armstrong number is a number that is the sum of its own digits each
    raised to the power of the number of digits. For example, 153 is an
    Armstrong number because 1^3 + 5^3 + 3^3 = 153.

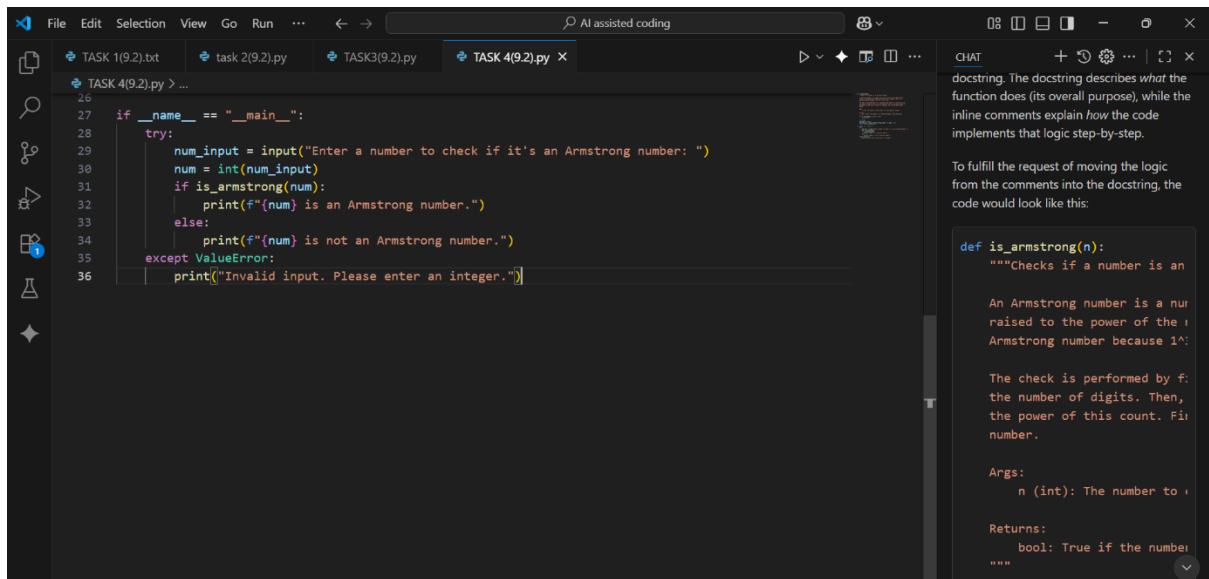
    The check is performed by first converting the number to a string to find
    the number of digits. Then, it calculates the sum of each digit raised to
    the power of this count. Finally, it compares this sum to the original
    number.

    Args:
        n (int): The number to check. Must be a non-negative integer.

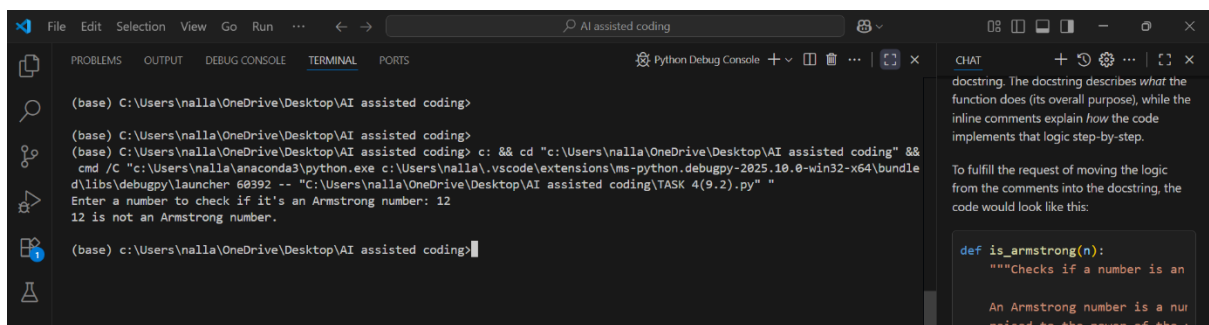
    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(n, int) or n < 0:
        return False

    s = str(n)
    num_digits = len(s)
    armstrong_sum = sum(int(digit)**num_digits for digit in s)
    return n == armstrong_sum

if __name__ == "__main__":
    try:
        num_input = input("Enter a number to check if it's an Armstrong number: ")
        num = int(num_input)
        if is_armstrong(num):
            print(f"{num} is an Armstrong number.")
        else:
            print(f"{num} is not an Armstrong number.")
```



## OUTPUT:



## OBSERVATION:

By moving the explanations from inline comments into the docstring, the code becomes more organized and adheres to standard Python documentation practices. The function's implementation is now clearly explained in one central place, improving readability and maintainability.

## TASK 5:

Task: Use AI to identify and correct inaccuracies in existing docstrings.

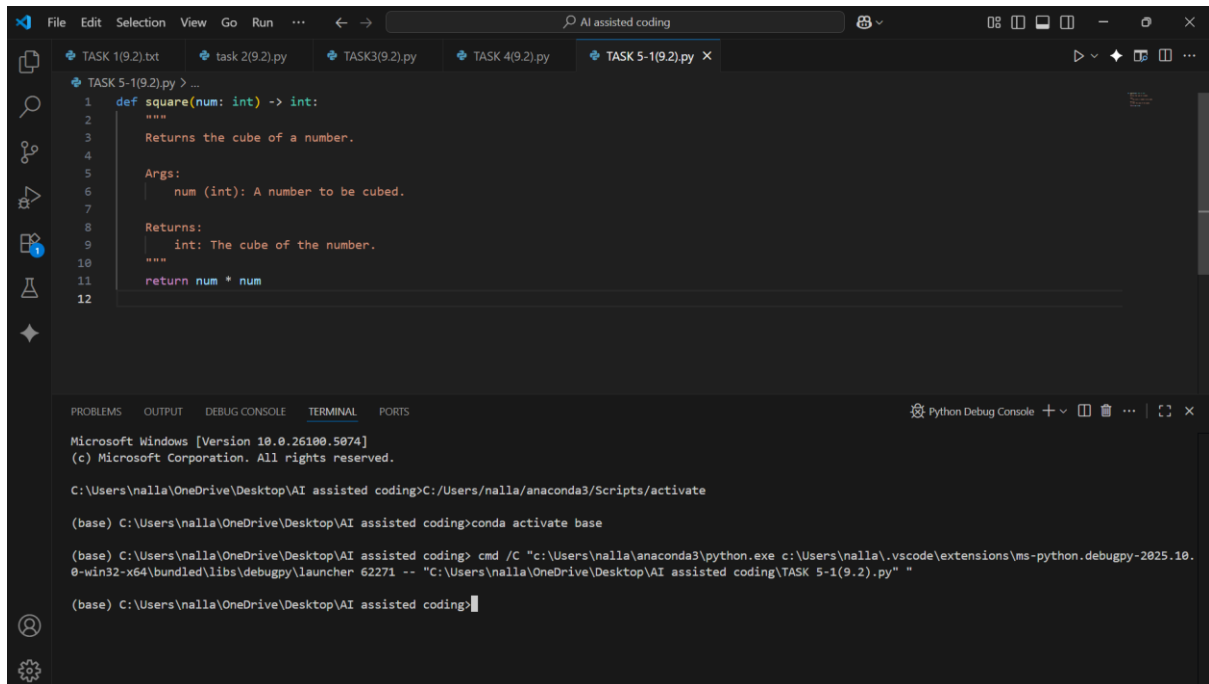
- Instructions:
  - o Provide Python code with outdated or incorrect docstrings.
  - o Instruct AI to rewrite each docstring to match the current code behavior.
  - o Ensure corrections follow Google-style formatting.

## PROMPT:

Identify and correct inaccuracies in existing docstrings.

**CODE:**

**OUTDATED CODE OR INCORRECT CODE:**



The screenshot shows a Visual Studio Code editor window with a Python file named 'TASK 5-1(9.2).py'. The code defines a function 'square' with a docstring that contains several inaccuracies. The terminal window at the bottom shows the command-line execution of the script, including the activation of the conda environment and the running of the Python script.

```
1 def square(num: int) -> int:
2     """
3     Returns the cube of a number.
4
5     Args:
6     |   num (int): A number to be cubed.
7
8     Returns:
9     |   int: The cube of the number.
10    """
11    return num * num
12
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console

Microsoft Windows [Version 10.0.26100.5074]  
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:/Users/nalla/anaconda3/Scripts/activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 62271 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\TASK 5-1(9.2).py" "

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>

**CORRECTED CODE:**

```
File Edit Selection View Go Run ... AI assisted coding
TASK 1(9.2).txt task 2(9.2).py TASK3(9.2).py TASK 4(9.2).py TASK 5-1(9.2).py task5-2(9.2).py X
task5-2(9.2).py > ...
1 def square(num: int) -> int:
2     """
3     Returns the square of a number.
4
5     Args:
6     |   num (int): A number to be squared.
7
8     Returns:
9     |   int: The square of the number.
10
11     Example:
12     |   >>> square(4)
13     |   16
14     """
15     return num * num

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console
Microsoft Windows [Version 10.0.26100.5074]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:/Users/nalla/anaconda3/Scripts/activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 59788 -- "c:\Users\nalla\OneDrive\Desktop\AI assisted coding\task5-2(9.2).py" "
```

## OBSERVATION:

The main issue is docstring drift—the code changes but the documentation doesn't. Correcting the docstrings to Google style makes the functions clearer, accurate, and easier to maintain.

## TASK 6:

Compare documentation output from a vague prompt and a detailed prompt for the same Python function.

- Instructions:
  - o Create two prompts: one simple ("Add comments to this function") and one detailed ("Add Google-style docstrings with parameters, return types, and examples").
  - o Use AI to process the same Python function with both prompts.
  - o Analyze and record differences in quality, accuracy, and completeness.

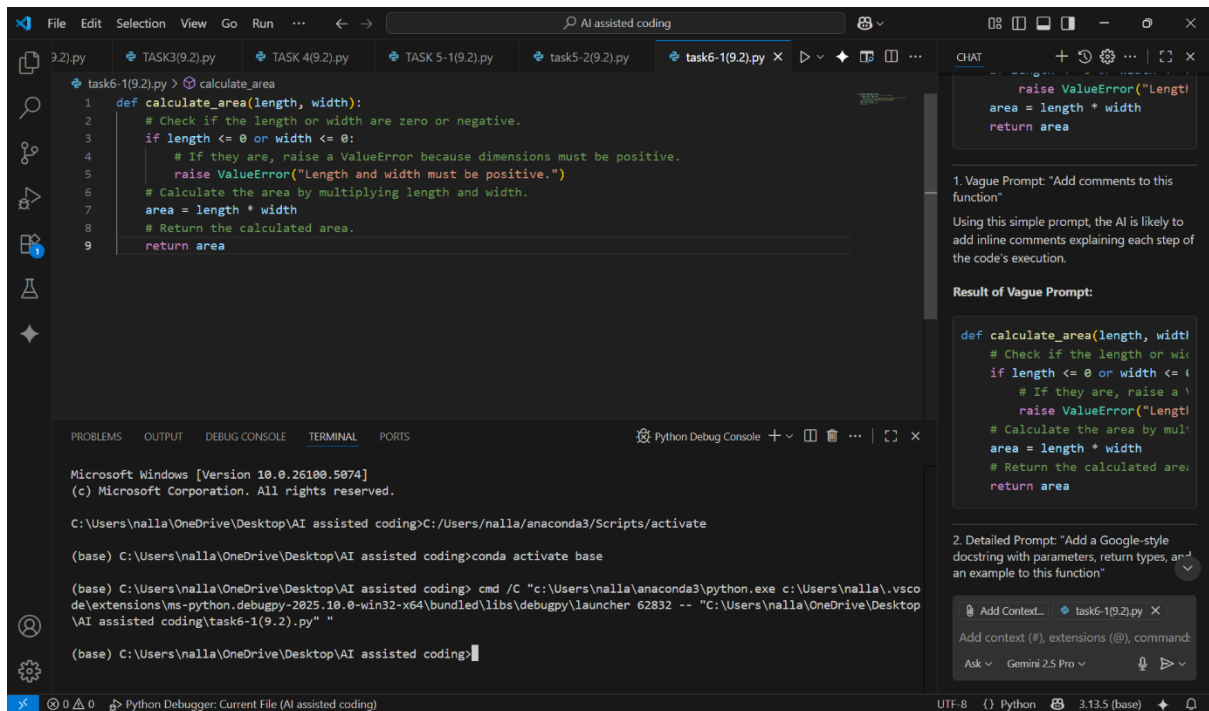
## PROMPT:

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

**Vague Prompt: "Add comments to this function"**

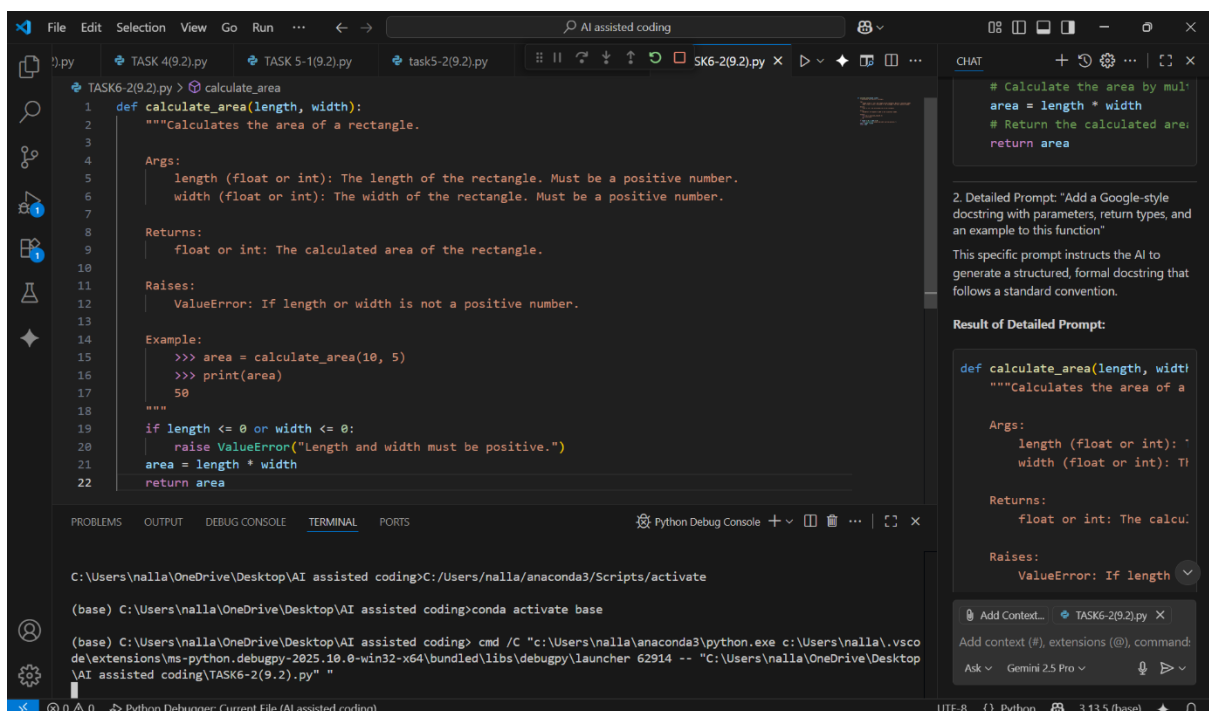


## CODE:



## Detailed Prompt: "Add a Google-style docstring with parameters, return types, and an example to this function"

## CODE:



**OBSERVATION:**

A detailed and specific prompt yields a vastly superior documentation result. It moves beyond simple line-by-line explanations to create structured, comprehensive, and professional documentation that significantly improves code maintainability and usability.