ASSIGNMENT – 9.2

NAME: NALLALA SIRI

HT.NO: 2403A52037

BATCH: 03

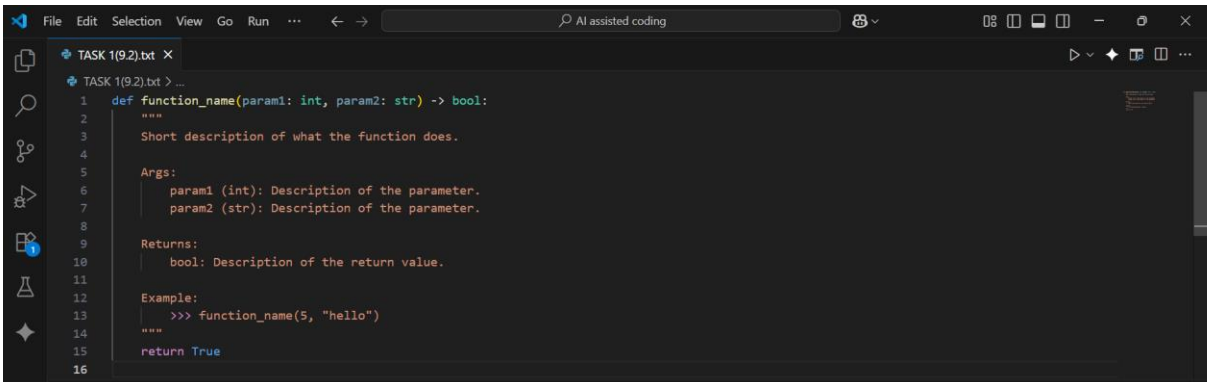
TASK 1:

(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:

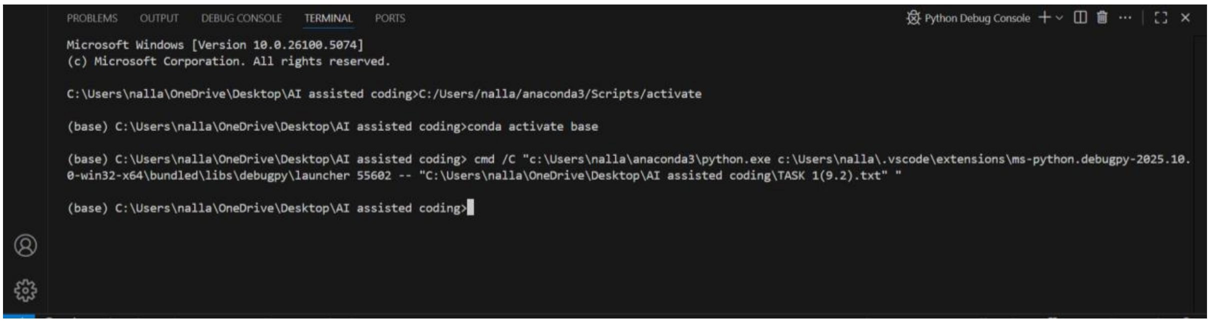
Add Google-style docstrings to all functions in a given Python script.

CODE:



```python
def function_name(param1: int, param2: str) -> bool:
    """
    Short description of what the function does.

    Args:
        param1 (int): Description of the parameter.
        param2 (str): Description of the parameter.

    Returns:
        bool: Description of the return value.

    Example:
        >>> function_name(5, "hello")
    """
    return True

```

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 55602 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\TASK 1(9.2).txt" "

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

OBSERVATION:

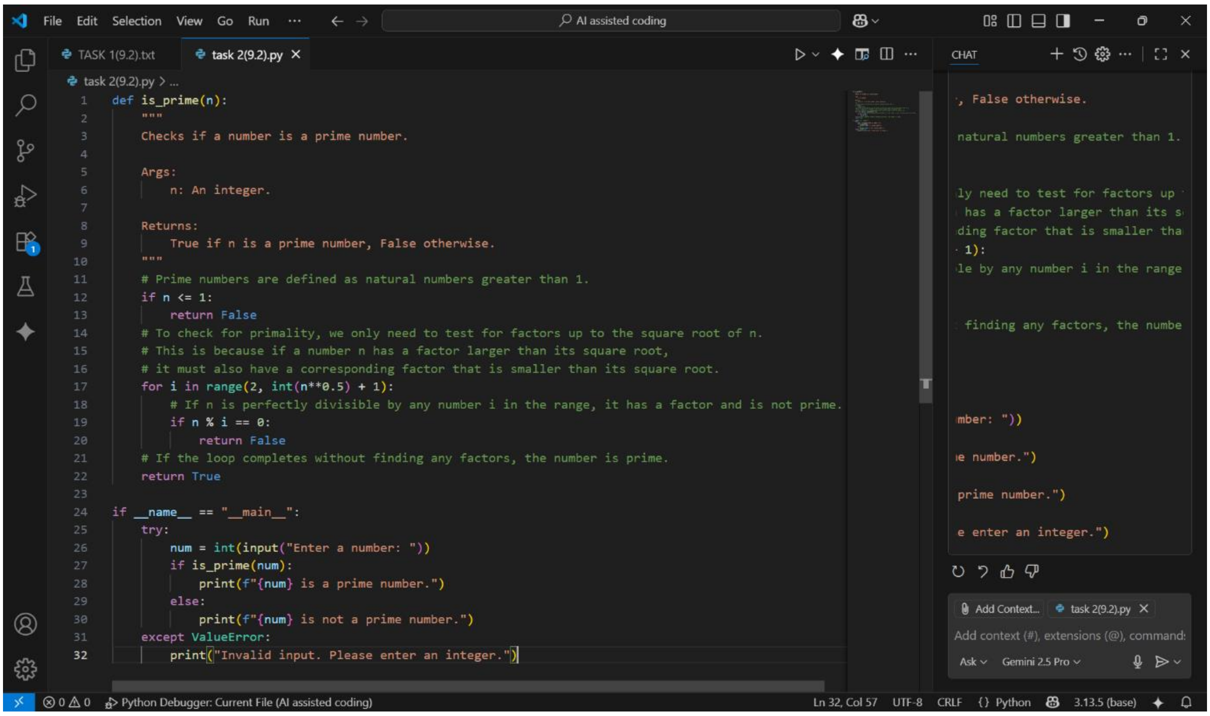
In Python, def function_name(param1: type, param2: type) -> return_type: is just a template for making a function. The name comes after def, the inputs go inside parentheses, : type shows each input's expected data type, and -> return_type shows what the function will give back. Since type and return_type 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:

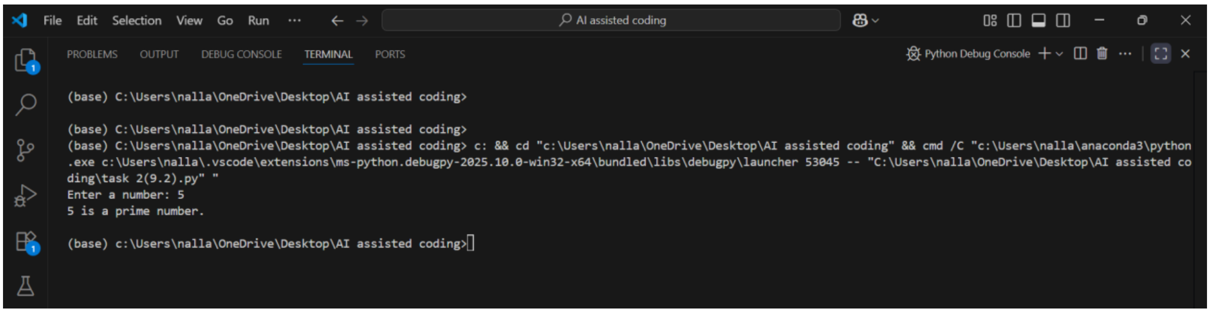
Add meaningful inline comments to a Python
program explaining only complex logic parts.
CODE:



```python
def is_prime(n):
    """
    Checks if a number is a prime number.

    Args:
        n: An integer.

    Returns:
        True if n is a prime number, False otherwise.
    """
    # Prime numbers are defined as natural numbers greater than 1.
    if n <= 1:
        return False
    # To check for primality, we only 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.
    for i in range(2, int(n**0.5) + 1):
        # If n is perfectly divisible by any number i in the range, it has a factor and is not prime.
        if n % i == 0:
            return False
    # If the loop completes without finding any factors, the number is prime.
    return True

if __name__ == "__main__":
    try:
        num = int(input("Enter a number: "))
        if is_prime(num):
            print(f"{num} is a prime number.")
        else:
            print(f"{num} is not a prime number.")
    except ValueError:
        print("Invalid input. Please enter an integer.")
```

```
', False otherwise.

natural numbers greater than 1.


ly need to test for factors up
 has a factor larger than its s
ding factor that is smaller tha
 1):
le by any number i in the range


 finding any factors, the numbe




mber: "))

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 co
ding\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.

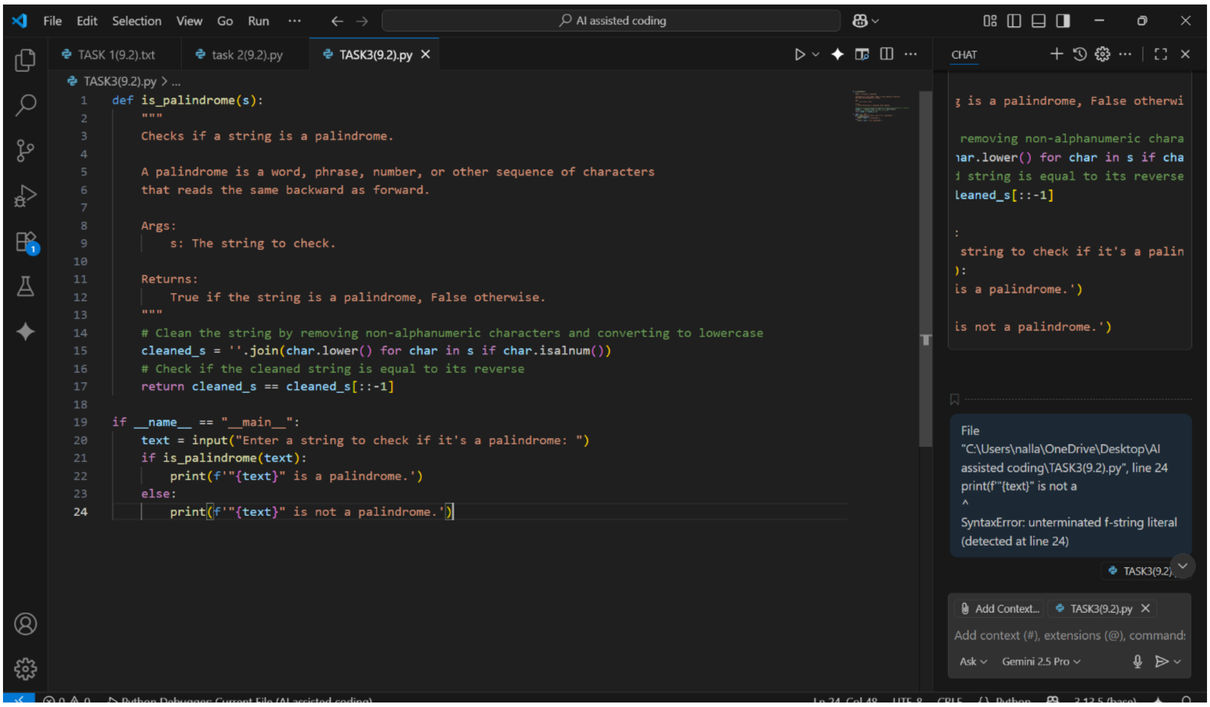
o Instruct AI to write a single multi-line docstring at the top of the file.

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:

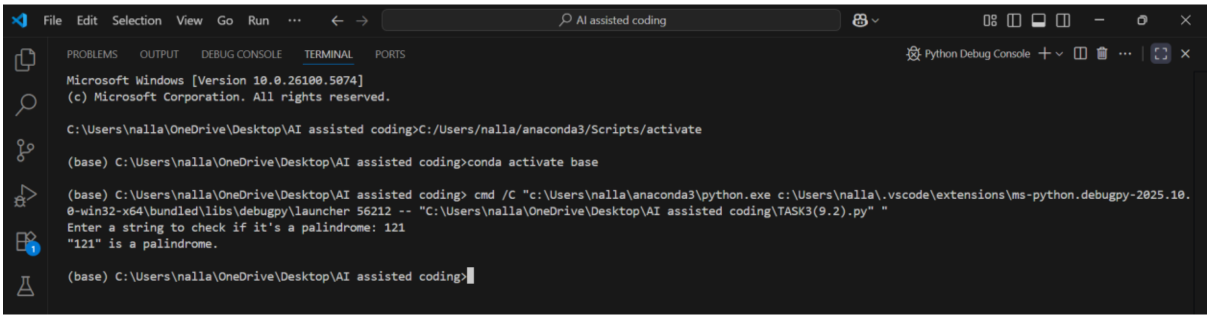


```python
def is_palindrome(s):
    """
    Checks if a string is a palindrome.

    A palindrome is a word, phrase, number, or other sequence of characters
    that reads the same backward as forward.

    Args:
        s: The string to check.

    Returns:
        True if the string is a palindrome, False otherwise.
    """
    # Clean the string by removing non-alphanumeric characters and converting to lowercase
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
    # Check if the cleaned string is equal to its reverse
    return cleaned_s == cleaned_s[::-1]

if __name__ == "__main__":
    text = input("Enter a string to check if it's a palindrome: ")
    if is_palindrome(text):
        print(f'"{text}" is a palindrome.')
    else:
        print(f'"{text}" is not a palindrome.')
```

```
g is a palindrome, False otherwi

 removing non-alphanumeric chara
ıar.lower() for char in s if cha
i string is equal to its reverse
leaned_s[::-1]

:
 string to check if it's a palin
):
is a palindrome.')

is not a palindrome.')
```

```
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:



```
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:

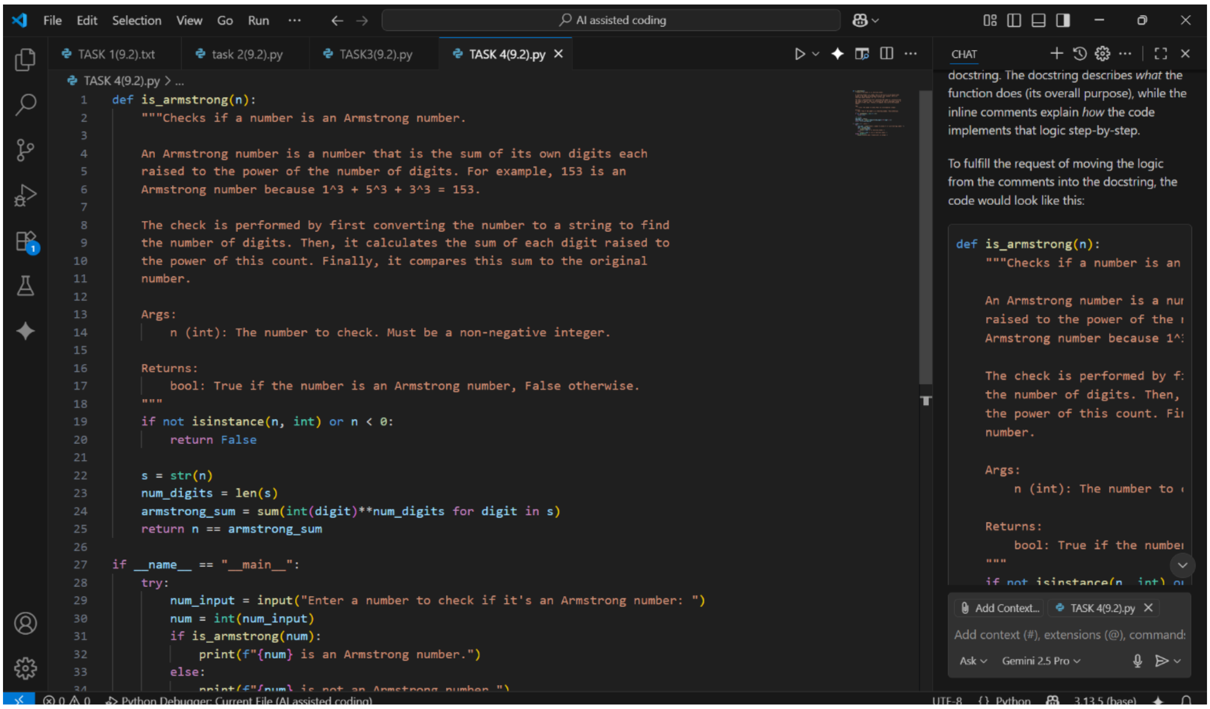
A module-level docstring is a documentation string placed at the very top of a Python file to provide a high-level summary of its contents. It briefly explains the module's purpose, lists any external libraries it depends on, and outlines the main functions or classes it offers. This allows other developers 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:



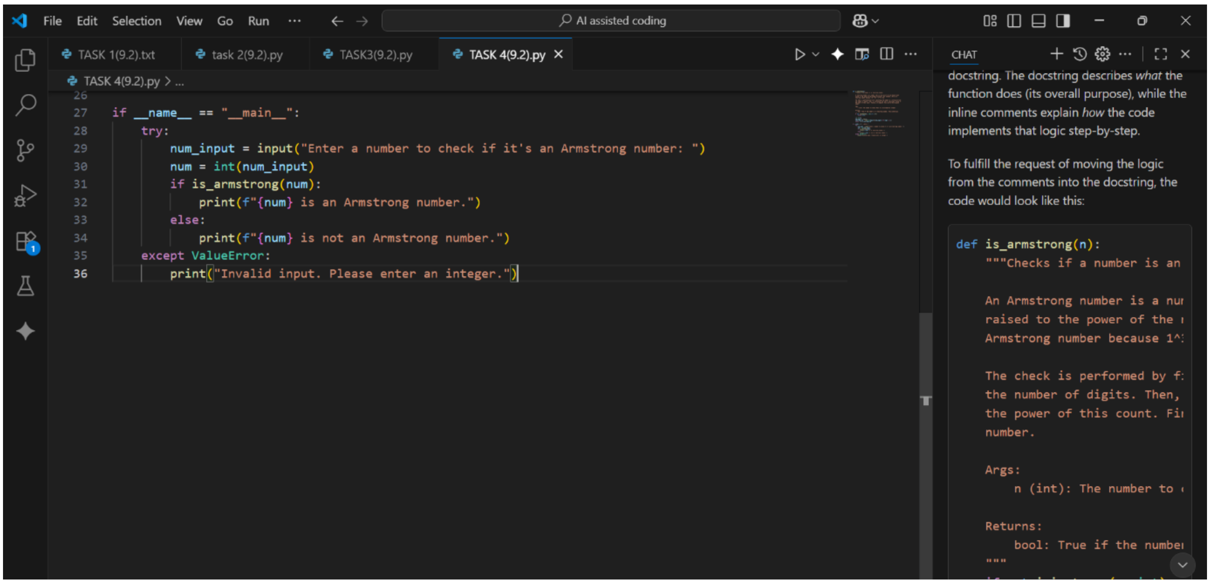
```python
def is_armstrong(n):
    """Checks if a number is an Armstrong number.

    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.")
```

docstring. The docstring describes *what* the function does (its overall purpose), while the inline comments explain *how* the code implements that logic step-by-step.

To fulfill the request of moving the logic from the comments into the docstring, the code would look like this:

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

    An Armstrong number is a nu
    raised to the power of the
    Armstrong number because 1^

    The check is performed by f
    the number of digits. Then,
    the power of this count. Fi
    number.

    Args:
        n (int): The number to

    Returns:
        bool: True if the numbe
    """
    if not isinstance(n, int) o
```

Add Context...
TASK 4(9.2).py
Add context (#), extensions (@), command
Ask
Gemini 2.5 Pro
⊗ 0 ⚠ 0
Python Debugger: Current File (AI assisted coding)
UTF-8
{} Python
3.13.5 (base)



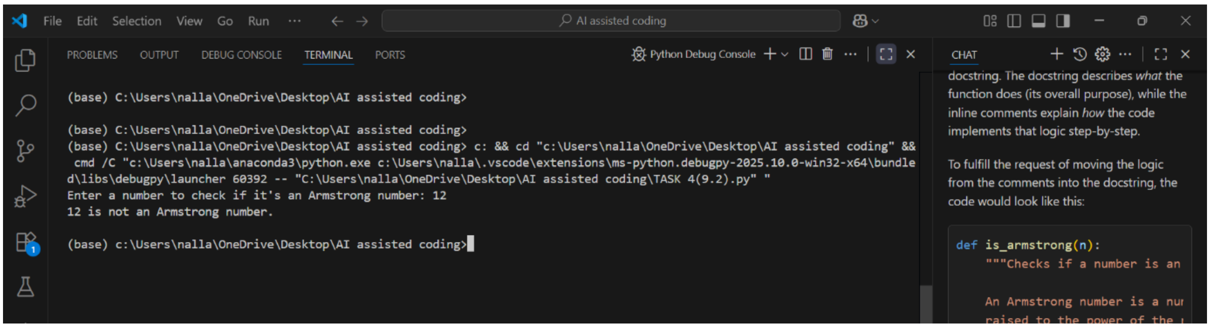
```python
if __name__ == "__main__":
    try:
        num_input = input("Enter a number to check if it's an Armstrong number: ")
        num = int(num_input)
        if is_armstrong(num):
            print(f"{num} is an Armstrong number.")
        else:
            print(f"{num} is not an Armstrong number.")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

docstring. The docstring describes *what* the function does (its overall purpose), while the inline comments explain *how* the code implements that logic step-by-step.

To fulfill the request of moving the logic from the comments into the docstring, the code would look like this:

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

    An Armstrong number is a nur
    raised to the power of the ı
    Armstrong number because 1^:

    The check is performed by f:
    the number of digits. Then,
    the power of this count. Fiı
    number.

    Args:
        n (int): The number to ‹

    Returns:
        bool: True if the numbeı
    """
```

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\bundle
d\libs\debugpy\launcher 60392 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\TASK 4(9.2).py" "
Enter a number to check if it's an Armstrong number: 12
12 is not an Armstrong number.

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

docstring. The docstring describes *what* the function does (its overall purpose), while the inline comments explain *how* the code implements that logic step-by-step.

To fulfill the request of moving the logic from the comments into the docstring, the code would look like this:

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

    An Armstrong number is a nur
    raised to the power of the ı
```

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.

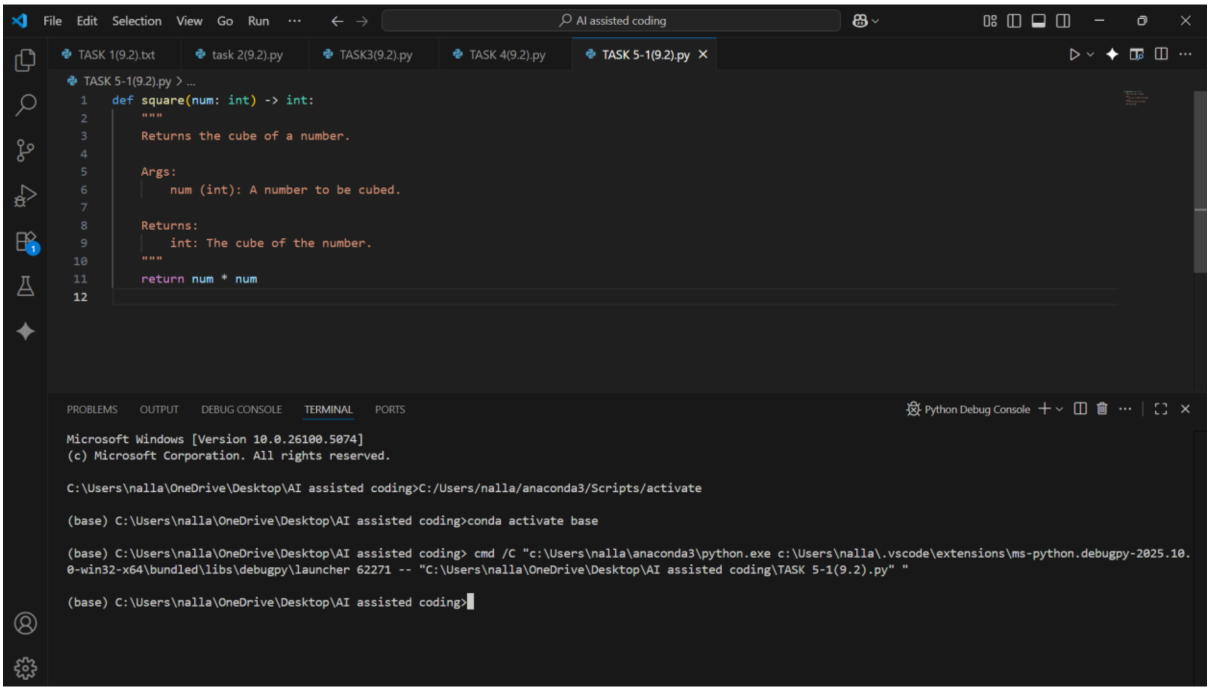
o Ensure corrections follow Google-style formatting.

PROMPT:

Identify and correct inaccuracies in existing docstrings.

CODE:

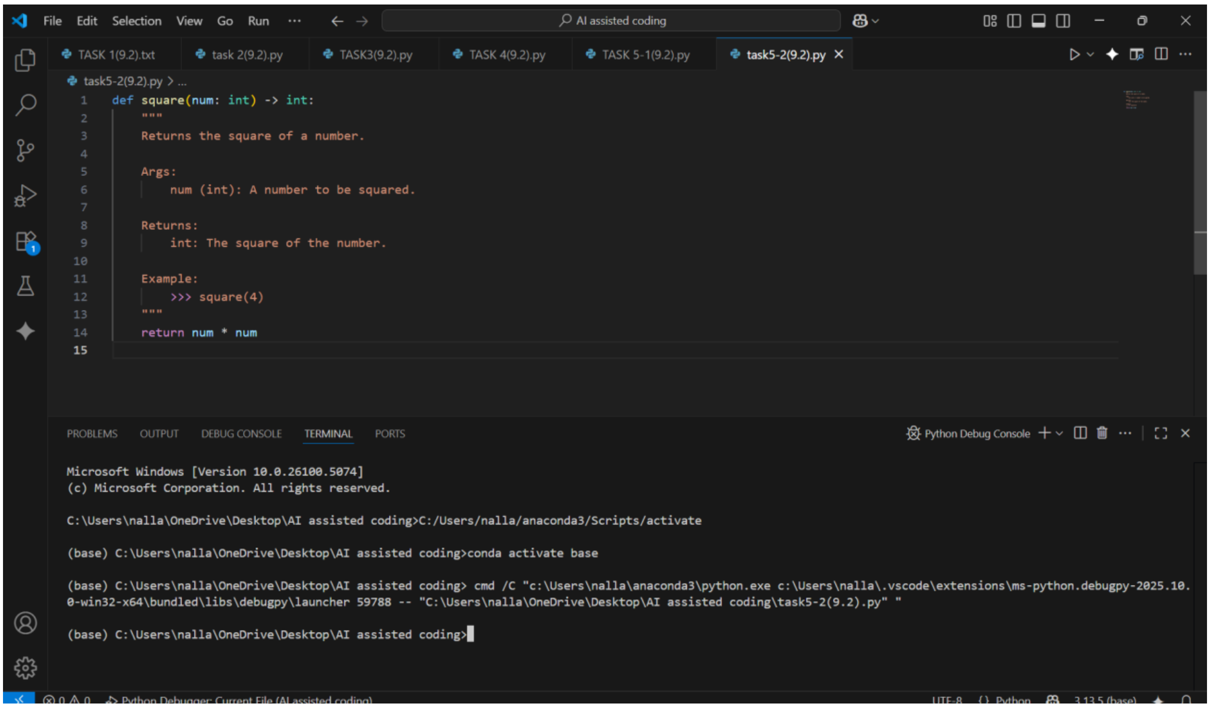
OUTDATED CODE OR INCORRECT CODE:



```python
def square(num: int) -> int:
    """
    Returns the cube of a number.

    Args:
        num (int): A number to be cubed.

    Returns:
        int: The cube of the number.
    """
    return num * num
```

```
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:



```python
def square(num: int) -> int:
    """
    Returns the square of a number.

    Args:
        num (int): A number to be squared.

    Returns:
        int: The square of the number.

    Example:
        >>> square(4)
    """
    return num * num
```

```
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" "

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

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:

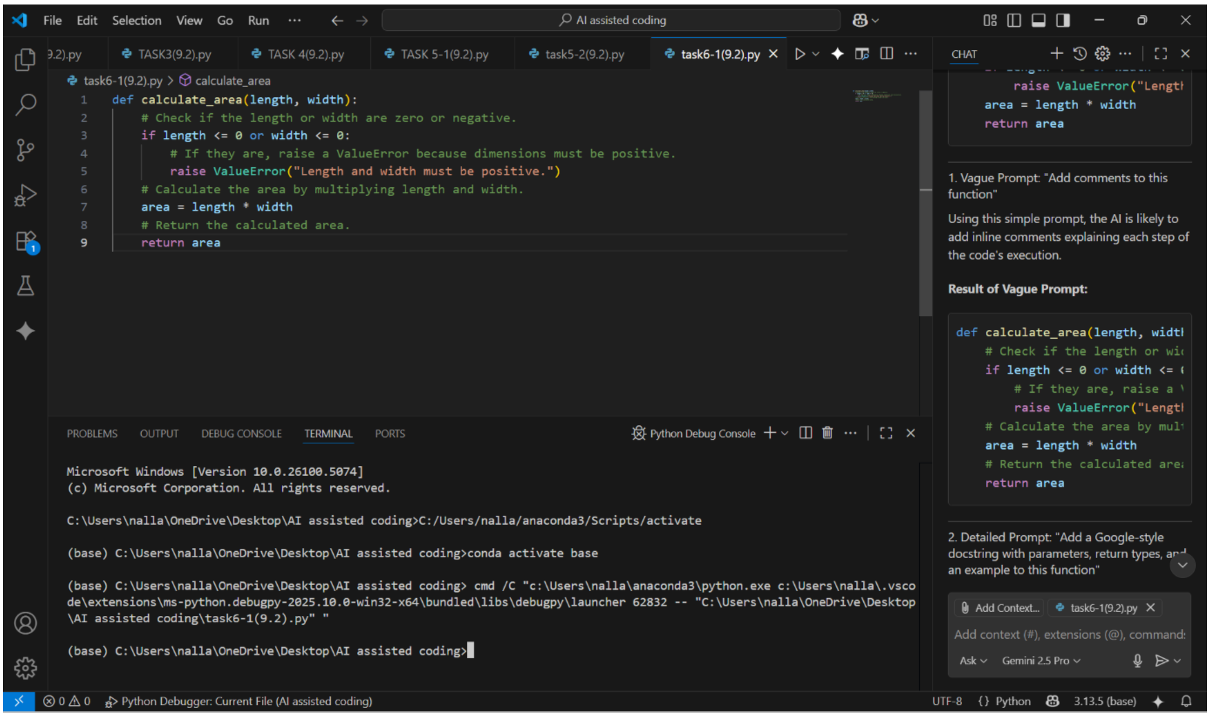
Compare documentation output from a vague prompt and a detailed prompt for the same Python function.
- Instructions:
o Create two prompts: one simple ("Add comments to this function") and one detailed ("Add Google-style docstrings with parameters, return types, and examples").
o Use AI to process the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



```python
def calculate_area(length, width):
    # Check if the length or width are zero or negative.
    if length <= 0 or width <= 0:
        # If they are, raise a ValueError because dimensions must be positive.
        raise ValueError("Length and width must be positive.")
    # Calculate the area by multiplying length and width.
    area = length * width
    # Return the calculated area.
    return area
```

```
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\.vsco
de\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 62832 -- "C:\Users\nalla\OneDrive\Desktop
\AI assisted coding\task6-1(9.2).py" "

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

```
        raise ValueError("Lengtl
    area = length * width
    return area
```

1. Vague Prompt: "Add comments to this function"

Using this simple prompt, the AI is likely to add inline comments explaining each step of the code's execution.

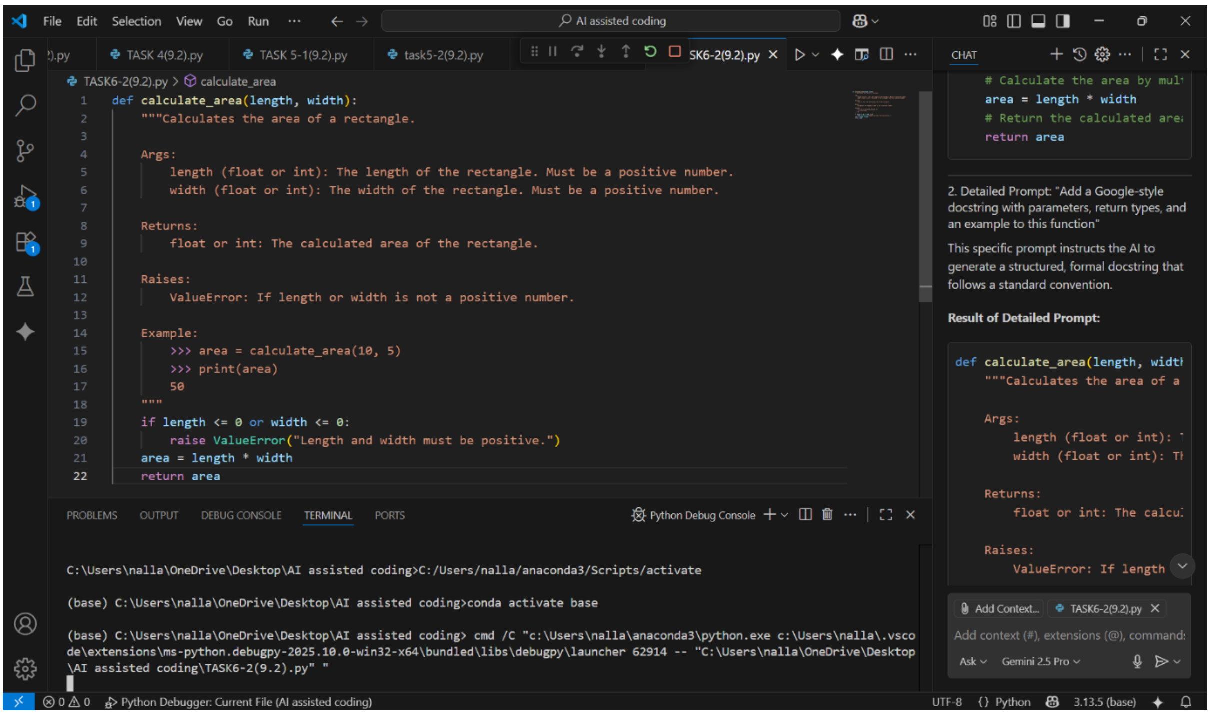
Result of Vague Prompt:

```
def calculate_area(length, widtl
    # Check if the length or wic
    if length <= 0 or width <= (
        # If they are, raise a \
        raise ValueError("Lengtl
    # Calculate the area by mul†
    area = length * width
    # Return the calculated are:
    return area
```

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

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

CODE:



```python
def calculate_area(length, width):
    """Calculates the area of a rectangle.

    Args:
        length (float or int): The length of the rectangle. Must be a positive number.
        width (float or int): The width of the rectangle. Must be a positive number.

    Returns:
        float or int: The calculated area of the rectangle.

    Raises:
        ValueError: If length or width is not a positive number.

    Example:
        >>> area = calculate_area(10, 5)
        >>> print(area)
        50
    """
    if length <= 0 or width <= 0:
        raise ValueError("Length and width must be positive.")
    area = length * width
    return area
```

```
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\.vsco
de\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 62914 -- "C:\Users\nalla\OneDrive\Desktop
\AI assisted coding\TASK6-2(9.2).py" "
```

```
    # Calculate the area by mul†
    area = length * width
    # Return the calculated are:
    return area
```

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

This specific prompt instructs the AI to generate a structured, formal docstring that follows a standard convention.

Result of Detailed Prompt:

```
def calculate_area(length, widtl
    """Calculates the area of a

    Args:
        length (float or int): 1
        width (float or int): Tl

    Returns:
        float or int: The calcu:

    Raises:
        ValueError: If length
```

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.