

# ASSIGNMENT – 9.2

NAME: KAVALI SAI KARTHIK

HT.NO: 2403A52043

BATCH: AIB03

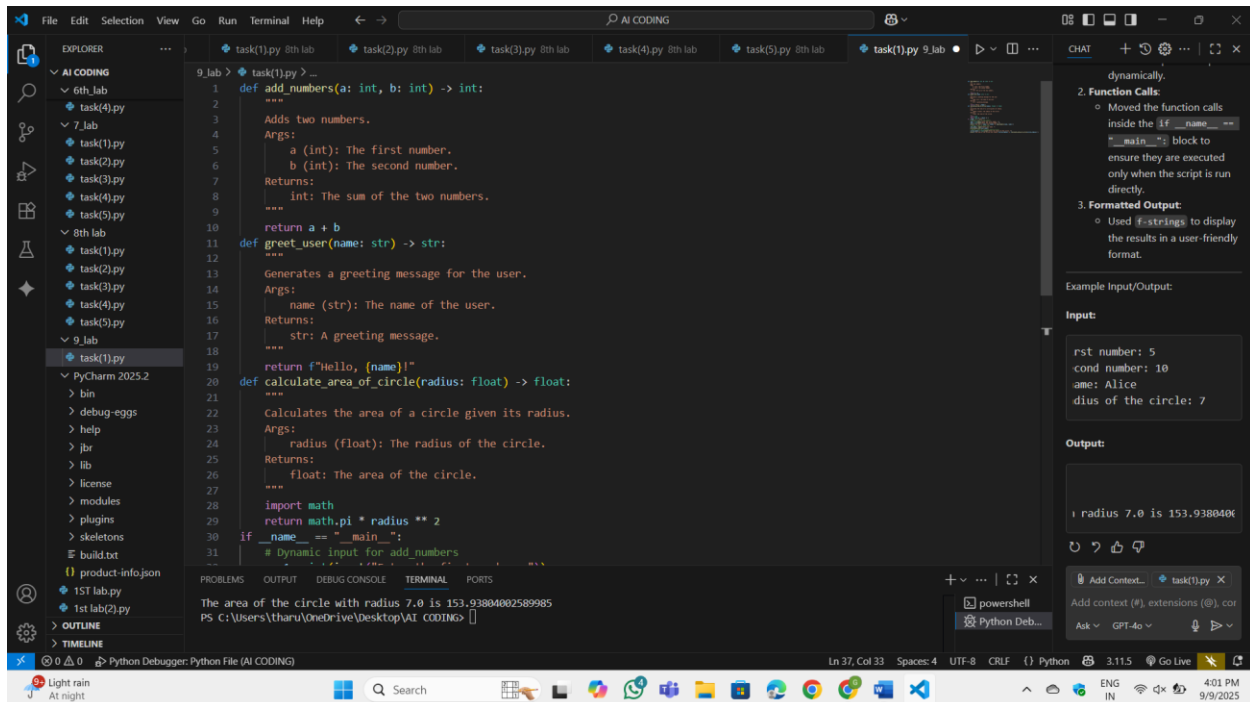
Task-1: (Documentation – Google-Style Docstrings for Python Functions)

- 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.
- Expected Output #1:
  - o A Python script with all functions documented using correctly formatted Google-style docstrings

Prompt: add Google-style docstrings to all functions in a given Python script.

Ensure each docstring includes: Function description, Parameters with type hints, Return values with type hints

Code:



```
9_lab > task(1).py > ...
1 def add_numbers(a: int, b: int) -> int:
2     """
3     Adds two numbers.
4     Args:
5         a (int): The first number.
6         b (int): The second number.
7     Returns:
8         int: The sum of the two numbers.
9     """
10    return a + b
11
12 def greet_user(name: str) -> str:
13    """
14    Generates a greeting message for the user.
15    Args:
16        name (str): The name of the user.
17    Returns:
18        str: A greeting message.
19    """
20    return f"Hello, {name}!"
21
22 def calculate_area_of_circle(radius: float) -> float:
23    """
24    Calculates the area of a circle given its radius.
25    Args:
26        radius (float): The radius of the circle.
27    Returns:
28        float: The area of the circle.
29    """
30    import math
31    return math.pi * radius ** 2
32
33 if __name__ == "__main__":
34    # Dynamic input for add_numbers
35    num1 = int(input("Enter the first number: "))
36    num2 = int(input("Enter the second number: "))
37    print(f"The sum of {num1} and {num2} is {add_numbers(num1, num2)}")
38    # Dynamic input for greet_user
39    user_name = input("Enter your name: ")
40    print(greet_user(user_name))
41    # Dynamic input for calculate_area_of_circle
42    circle_radius = float(input("Enter the radius of the circle: "))
43    print(f"The area of the circle with radius {circle_radius} is {calculate_area_of_circle(circle_radius)}")
```

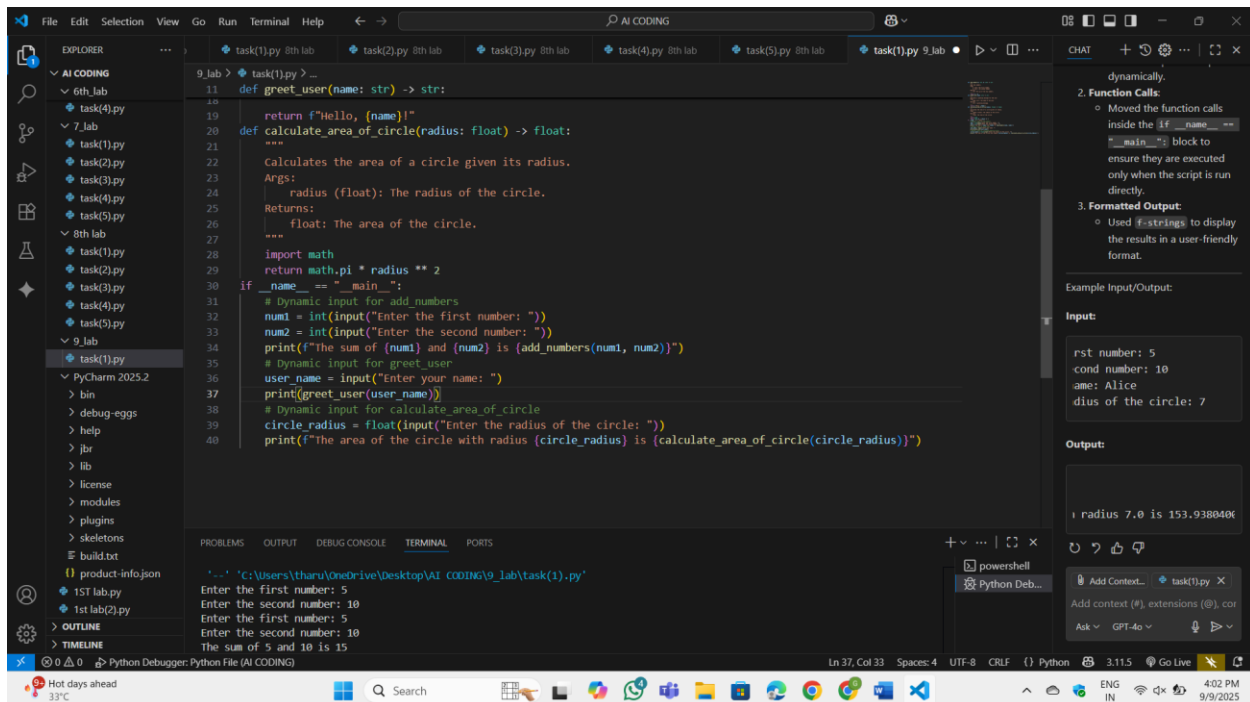
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

The area of the circle with radius 7.0 is 153.93804002589985  
PS C:\Users\tharu\OneDrive\Desktop\AI CODING> []

Light rain  
At night

Search

Ln 37, Col 33 Spaces: 4 UTF-8 CRLF Python 3.11.5 Go Live 4:01 PM 9/9/2025



```
9_lab > task(1).py > ...
11 def greet_user(name: str) -> str:
12     """
13     Generates a greeting message for the user.
14     Args:
15         name (str): The name of the user.
16     Returns:
17         str: A greeting message.
18     """
19     return f"Hello, {name}!"
20
21 def calculate_area_of_circle(radius: float) -> float:
22     """
23     Calculates the area of a circle given its radius.
24     Args:
25         radius (float): The radius of the circle.
26     Returns:
27         float: The area of the circle.
28     """
29     import math
30     return math.pi * radius ** 2
31
32 if __name__ == "__main__":
33    # Dynamic input for add_numbers
34    num1 = int(input("Enter the first number: "))
35    num2 = int(input("Enter the second number: "))
36    print(f"The sum of {num1} and {num2} is {add_numbers(num1, num2)}")
37    # Dynamic input for greet_user
38    user_name = input("Enter your name: ")
39    print(greet_user(user_name))
40    # Dynamic input for calculate_area_of_circle
41    circle_radius = float(input("Enter the radius of the circle: "))
42    print(f"The area of the circle with radius {circle_radius} is {calculate_area_of_circle(circle_radius)}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

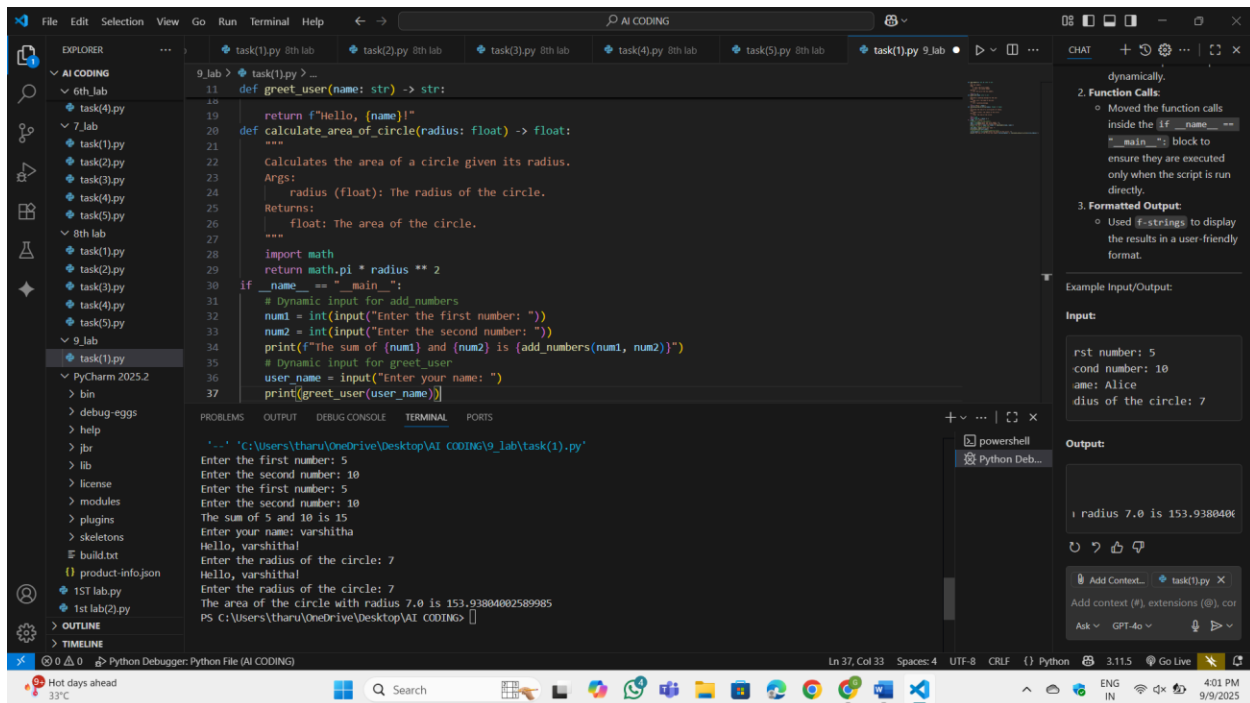
... 'C:\Users\tharu\OneDrive\Desktop\AI CODING\9\_lab\task(1).py'  
Enter the first number: 5  
Enter the second number: 10  
Enter the first number: 5  
Enter the second number: 10  
The sum of 5 and 10 is 15

Hot days ahead  
33°C

Search

Ln 37, Col 33 Spaces: 4 UTF-8 CRLF Python 3.11.5 Go Live 4:02 PM 9/9/2025

OP:



## Observation:

Added input() prompts for each function to allow the user to provide input dynamically. Moved the function calls inside the if `__name__ == "__main__"` block to ensure they are executed only when the script is run directly. Used f-strings to display the results in a user-friendly format.

## 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.

- Expected Output #2:

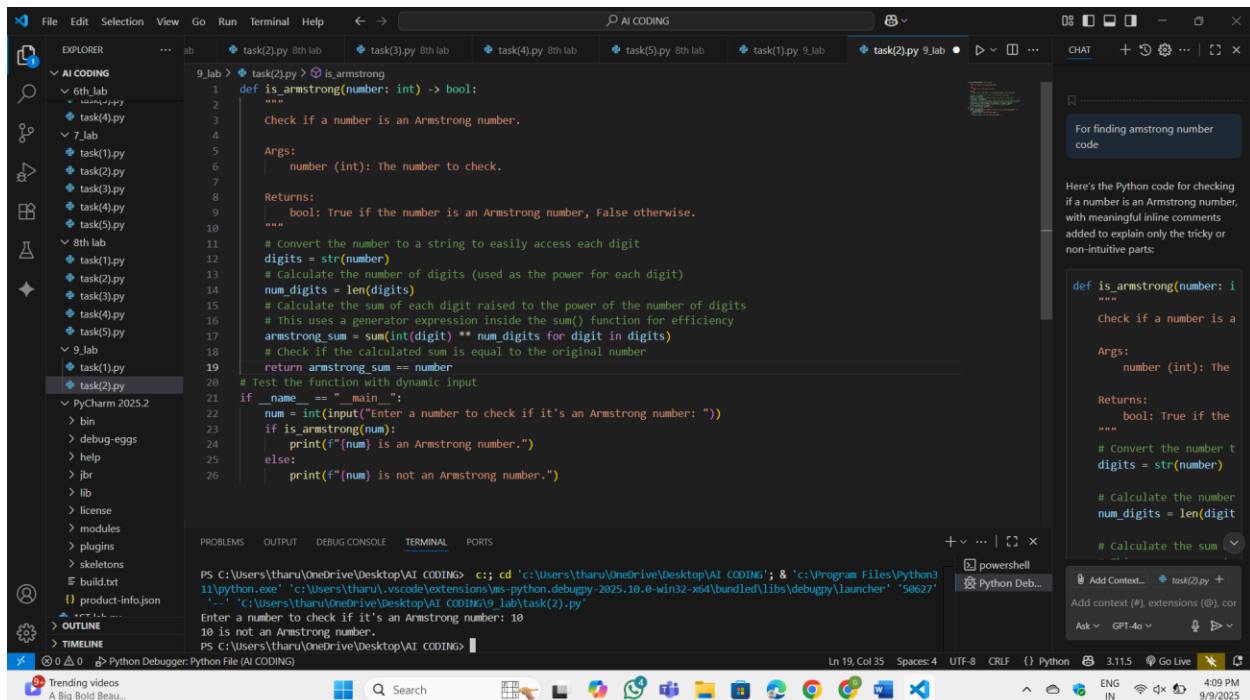
- o Python code with concise, context-aware inline comments for complex logic blocks

Prompt: for the code I provided focus only on tricky or non-intuitive code sections. And add meaningful inline comments to a Python program explaining only complex logic parts

Code:

```
1 def is_armstrong(number: int) -> bool:
2     """
3     Check if a number is an Armstrong number.
4
5     Args:
6         number (int): The number to check.
7
8     Returns:
9         bool: True if the number is an Armstrong number, False otherwise.
10    """
11    # Convert the number to a string to easily access each digit
12    digits = str(number)
13    # Calculate the number of digits (used as the power for each digit)
14    num_digits = len(digits)
15    # Calculate the sum of each digit raised to the power of the number of digits
16    # This uses a generator expression inside the sum() function for efficiency
17    armstrong_sum = sum(int(digit) ** num_digits for digit in digits)
18    # Check if the calculated sum is equal to the original number
19    return armstrong_sum == number
20
21 # Test the function with dynamic input
22 if __name__ == "__main__":
23     num = int(input("Enter a number to check if it's an Armstrong number: "))
24     if is_armstrong(num):
25         print(f"{num} is an Armstrong number.")
26     else:
27         print(f"{num} is not an Armstrong number.")
```

OP:



## Observation:

`digits = str(number)`: Converts the number to a string to allow iteration over its digits. `num_digits = len(digits)`: Calculates the number of digits in the number, which determines the power to which each digit is raised. `sum(int(digit) ** num_digits for digit in digits)`: Uses a generator expression to calculate the sum of each digit raised to the power of `num_digits`. This avoids creating an intermediate list, making the code more memory-efficient. `return armstrong_sum == number`: Compares the calculated sum to the original number to determine if it's an Armstrong number

## 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: for the given code describes functionality and usage without rewriting the entire code

Code:

```

1 import numpy as np
2 def inverse_matrix(matrix):
3     """
4     Calculate the inverse of a given square matrix.
5     Args:
6         matrix (list of list of float): A square matrix to invert.
7     Returns:
8         numpy.ndarray: The inverse of the matrix if it exists.
9         str: An error message if the matrix is singular or not square.
10    """
11    try:
12        # Convert the input list to a numpy array
13        np_matrix = np.array(matrix)
14        # Calculate the inverse using numpy's linalg.inv function
15        inverse = np.linalg.inv(np_matrix)
16        return inverse
17    except np.linalg.LinAlgError:
18        return "Error: The matrix is singular and cannot be inverted."
19    except ValueError:
20        return "Error: The input must be a square matrix."
21    # Test the function with dynamic input
22    if __name__ == "__main__":
23        print("Enter the matrix row by row, with elements separated by spaces.")
24        print("Enter an empty line to finish input.")
25        matrix = []
26        while True:
27            row = input("Enter row (or press Enter to finish): ")
28            if not row:
29                break
30            matrix.append(list(map(float, row.split())))
31        # Check if the matrix is square
32        if all(len(row) == len(matrix) for row in matrix):
33            result = inverse_matrix(matrix)
34            print("\nInverse of the matrix:")
35            print(result)
36        else:
37            print("Error: The input must be a square matrix.")

```

4. Output

program returns an error message.

The program prints the inverse of the matrix if it exists.

Example Input/Output:

Input:

separated by spaces.

Output:

Inverse of the matrix:  
[[ 4. -1.]  
[-7. 2.]]

Input (Singular Matrix):

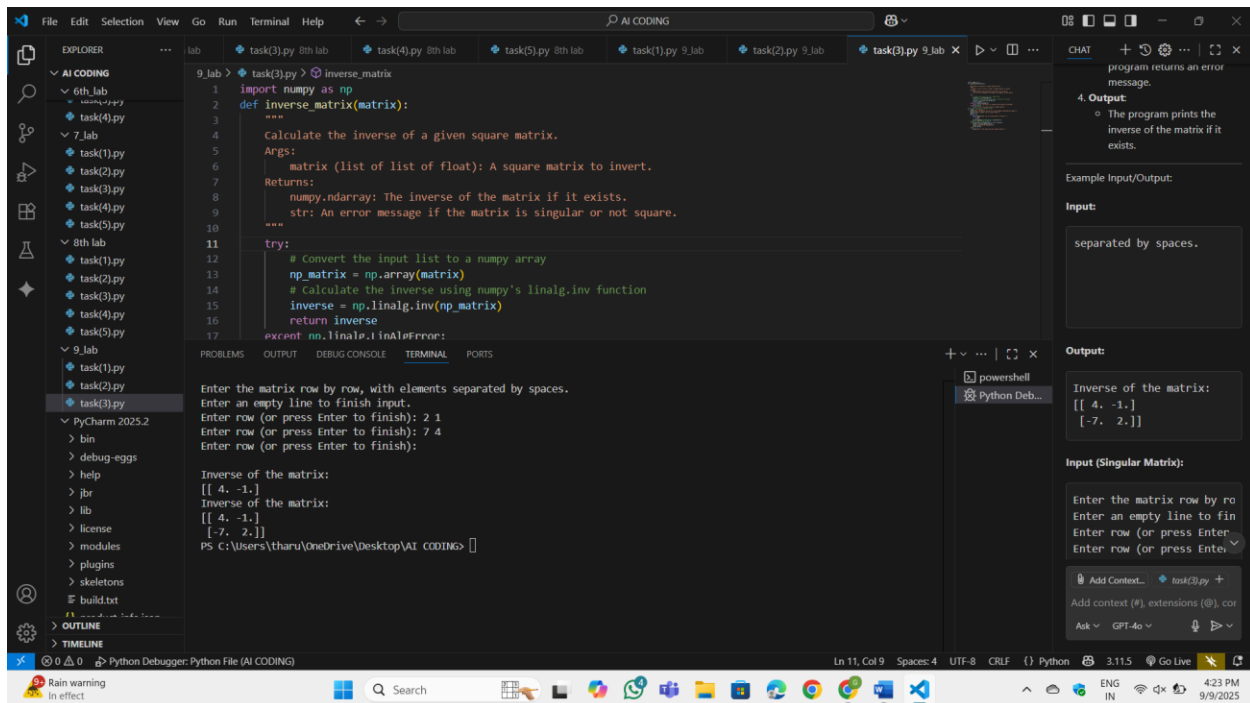
Enter the matrix row by row  
Enter an empty line to finish  
Enter row (or press Enter)  
Enter row (or press Enter)

Add Context... task(3).py +

Add context (#), extensions (@), cor

Ask GPT-4o

OP:



## Observation:

The numpy library is used for matrix operations.

The `np.linalg.inv()` function calculates the inverse of a matrix. The user inputs the matrix row by row. The input is converted into a list of lists, where each inner list represents a row. If the matrix is not square, the program returns an error message. If the matrix is singular (determinant = 0), the program returns an error message. The program prints the inverse of the matrix if it exists.

## Task-4: (Documentation – Convert Comments to Structured Docstrings)

- 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.

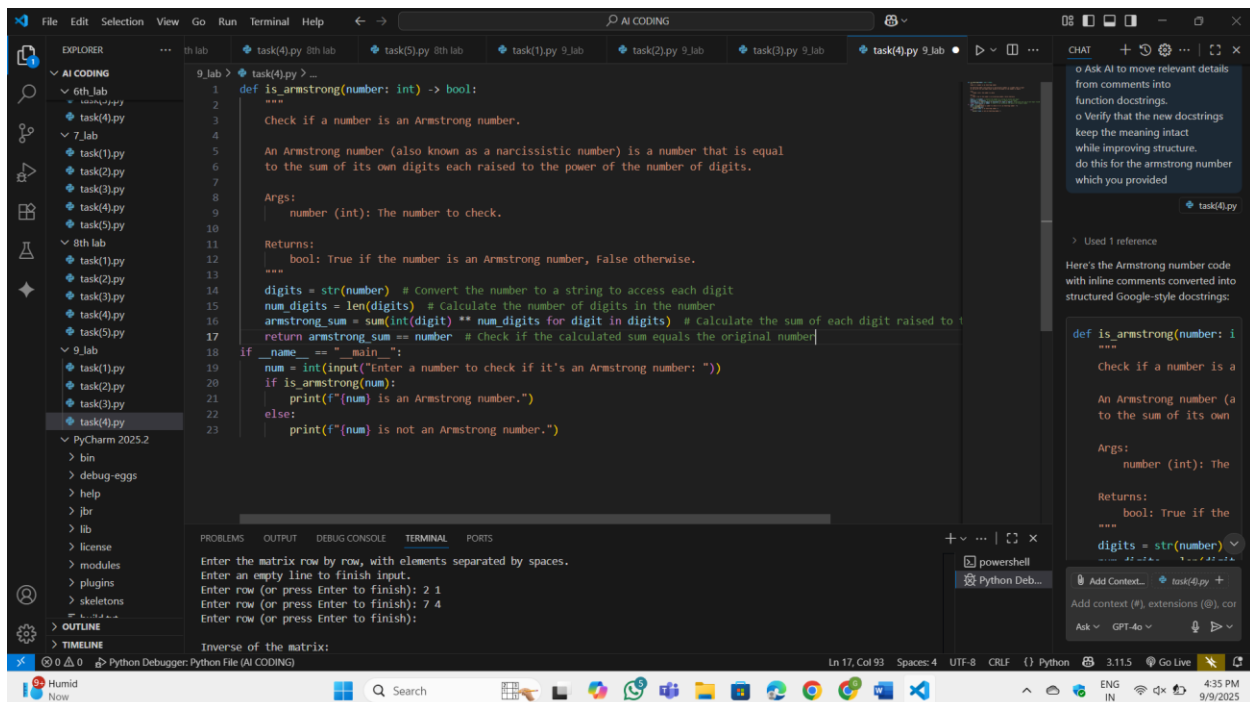
- o Verify that the new docstrings keep the meaning intact while improving structure.

- Expected Output #4:

- o Python code with comments replaced by clear, standardized docstrings

Prompt: convert Comments to Structured Docstrings for the code which I provided.

Code:



```
def is_armstrong(number: int) -> bool:
    """
    Check if a number is an Armstrong number.

    An Armstrong number (also known as a narcissistic number) is a number that is equal
    to the sum of its own digits each raised to the power of the number of digits.

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

    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    digits = str(number) # Convert the number to a string to access each digit
    num_digits = len(digits) # Calculate the number of digits in the number
    armstrong_sum = sum(int(digit) ** num_digits for digit in digits) # Calculate the sum of each digit raised to the power of the number of digits
    return armstrong_sum == number # Check if the calculated sum equals the original number

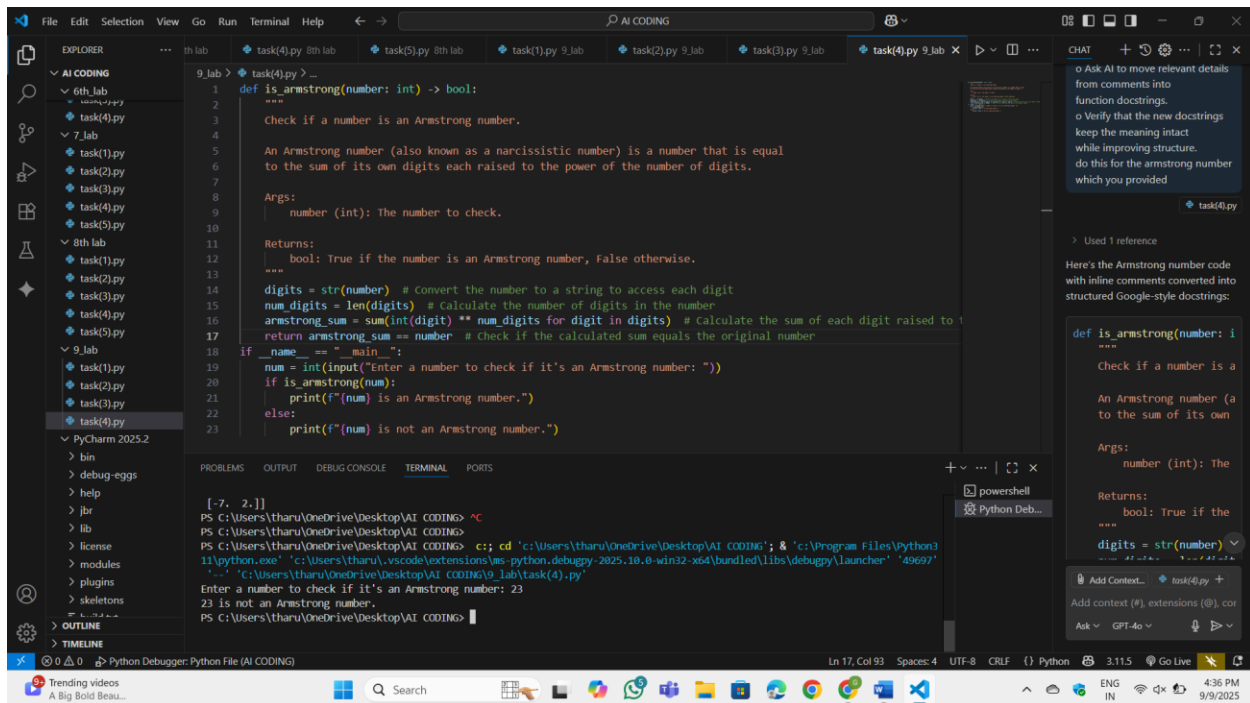
if __name__ == "__main__":
    num = int(input("Enter a number to check if it's an Armstrong number: "))
    if is_armstrong(num):
        print(f"{num} is an Armstrong number.")
    else:
        print(f"{num} is not an Armstrong number.")
```

Chat: 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. do this for the armstrong number which you provided

Terminal: Enter the matrix row by row, with elements separated by spaces. Enter an empty line to finish input. Enter row (or press Enter to finish): 2 1 Enter row (or press Enter to finish): 7 4 Enter row (or press Enter to finish):

OP:





## Observation:

Removed inline comments from the code. Added a detailed docstring to the `is_armstrong` function. The docstring explains: **Purpose:** What the function does. **Args:** The input parameter and its type. **Returns:** The return value and its type. **Logic:** A brief explanation of the Armstrong number concept. The docstring improves the structure and readability of the code while keeping the meaning intact

## Task-5: (Documentation – Review and Correct Docstrings)

- 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.

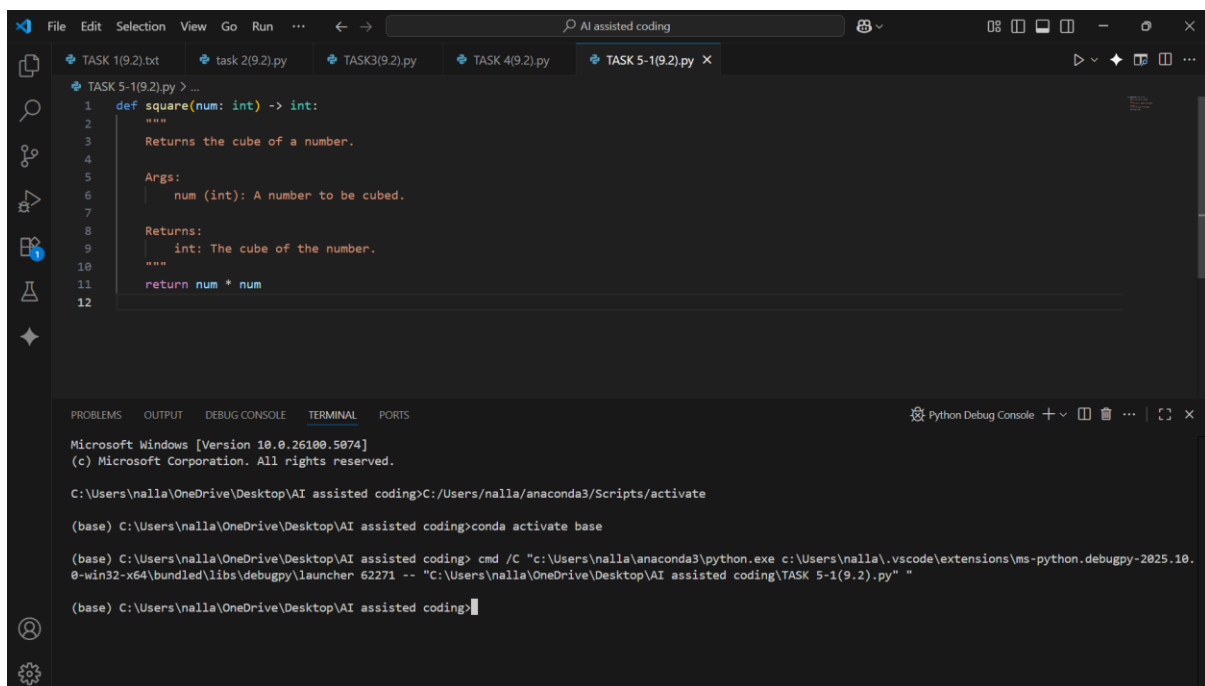
- o Ensure corrections follow Google-style formatting.

- Expected Output #5:

- o Python file with updated, accurate, and standardized docstrings

Prompt: Identify and correct inaccuracies in existing docstrings.

Code:



The screenshot shows a VS Code editor with a Python file named 'TASK 5-1(9.2).py'. The code defines a function 'square' that takes an integer 'num' and returns its square. The docstring is updated to accurately reflect the function's behavior. Below the editor, a terminal window shows the commands used to run the code.

```
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" "
```

```
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     """
14     return num * num
15
```

```
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 59788 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\task5-2(9.2).py" "
```

OP:

```
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 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:

(Documentation – Prompt Comparison Experiment)

- Task: 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.

- Expected Output #6:

- o A comparison table showing the results from both prompts with observations

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”).

Code:

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

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\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>

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, 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
```

```
1 def calculate_area(length, width):
2     """Calculates the area of a rectangle.
3
4     Args:
5         length (float or int): The length of the rectangle. Must be a positive number.
6         width (float or int): The width of the rectangle. Must be a positive number.
7
8     Returns:
9         float or int: The calculated area of the rectangle.
10
11     Raises:
12         ValueError: If length or width is not a positive number.
13
14     Example:
15         >>> area = calculate_area(10, 5)
16         >>> print(area)
17         50
18     """
19     if length <= 0 or width <= 0:
20         raise ValueError("Length and width must be positive.")
21     area = length * width
22     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\.vscode\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" "

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, 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
```

OP:

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.