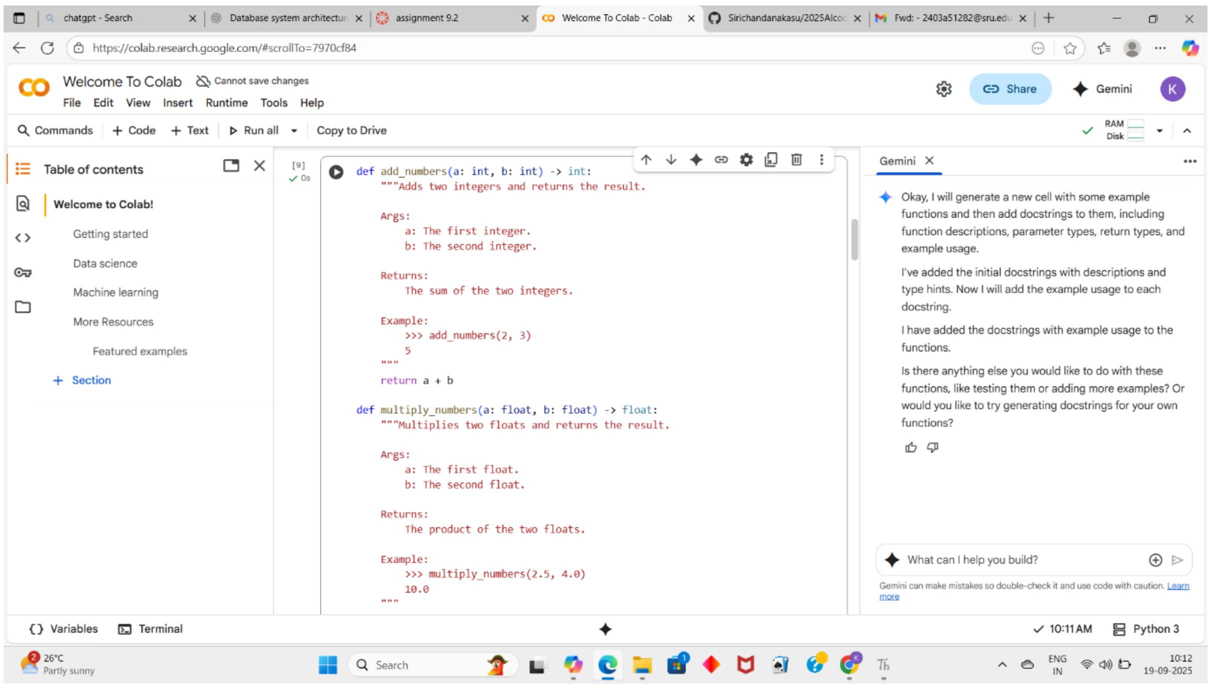
ASSIGNMNET

HTNO: 2403A51282

TASK1:



```python
def add_numbers(a: int, b: int) -> int:
    """Adds two integers and returns the result.

    Args:
        a: The first integer.
        b: The second integer.

    Returns:
        The sum of the two integers.

    Example:
        >>> add_numbers(2, 3)
        5
    """
    return a + b

def multiply_numbers(a: float, b: float) -> float:
    """Multiplies two floats and returns the result.

    Args:
        a: The first float.
        b: The second float.

    Returns:
        The product of the two floats.

    Example:
        >>> multiply_numbers(2.5, 4.0)
        10.0
    """
```

Okay, I will generate a new cell with some example functions and then add docstrings to them, including function descriptions, parameter types, return types, and example usage.

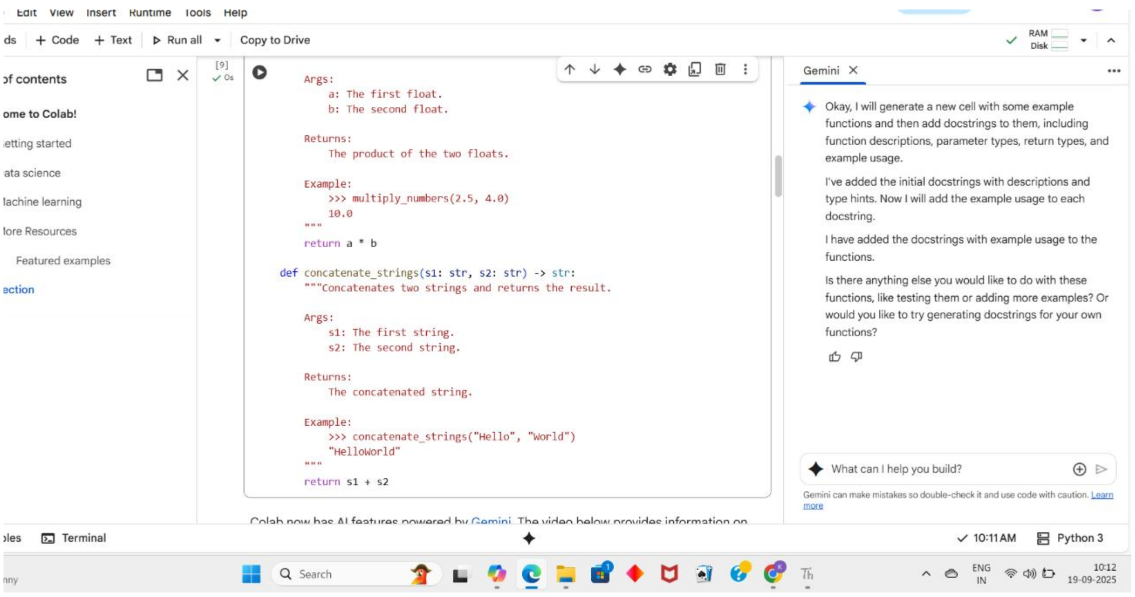
I've added the initial docstrings with descriptions and type hints. Now I will add the example usage to each docstring.

I have added the docstrings with example usage to the functions.

Is there anything else you would like to do with these functions, like testing them or adding more examples? Or would you like to try generating docstrings for your own functions?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more


Edit View Insert Runtime Tools Help
Code Text Run all Copy to Drive
Terminal
10:11 AM Python 3

```python
    Args:
        a: The first float.
        b: The second float.

    Returns:
        The product of the two floats.

    Example:
        >>> multiply_numbers(2.5, 4.0)
        10.0
    """
    return a * b

def concatenate_strings(s1: str, s2: str) -> str:
    """Concatenates two strings and returns the result.

    Args:
        s1: The first string.
        s2: The second string.

    Returns:
        The concatenated string.

    Example:
        >>> concatenate_strings("Hello", "World")
        "HelloWorld"
    """
    return s1 + s2
```

Colab now has AI features powered by Gemini. The video below provides information on

Okay, I will generate a new cell with some example functions and then add docstrings to them, including function descriptions, parameter types, return types, and example usage.

I've added the initial docstrings with descriptions and type hints. Now I will add the example usage to each docstring.

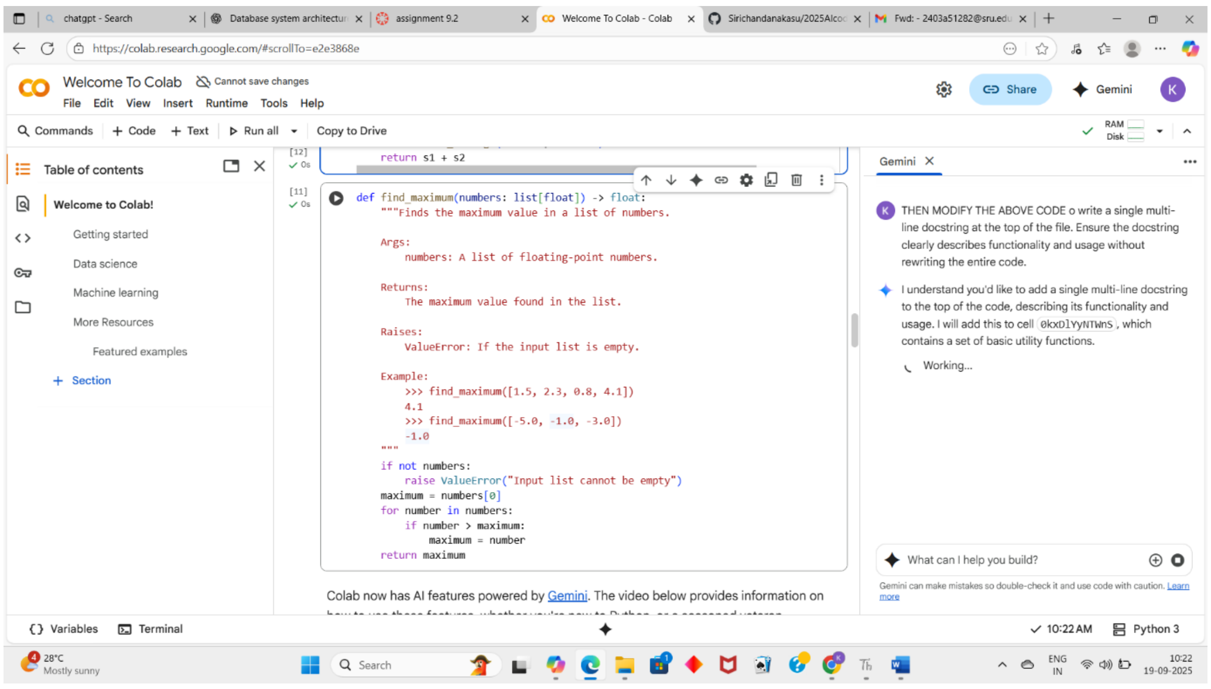
I have added the docstrings with example usage to the functions.

Is there anything else you would like to do with these functions, like testing them or adding more examples? Or would you like to try generating docstrings for your own functions?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

TASK3:



```python
    return s1 + s2
```

```python
def find_maximum(numbers: list[float]) -> float:
    """Finds the maximum value in a list of numbers.

    Args:
        numbers: A list of floating-point numbers.

    Returns:
        The maximum value found in the list.

    Raises:
        ValueError: If the input list is empty.

    Example:
        >>> find_maximum([1.5, 2.3, 0.8, 4.1])
        4.1
        >>> find_maximum([-5.0, -1.0, -3.0])
        -1.0
    """
    if not numbers:
        raise ValueError("Input list cannot be empty")
    maximum = numbers[0]
    for number in numbers:
        if number > maximum:
            maximum = number
    return maximum
```

Colab now has AI features powered by Gemini. The video below provides information on

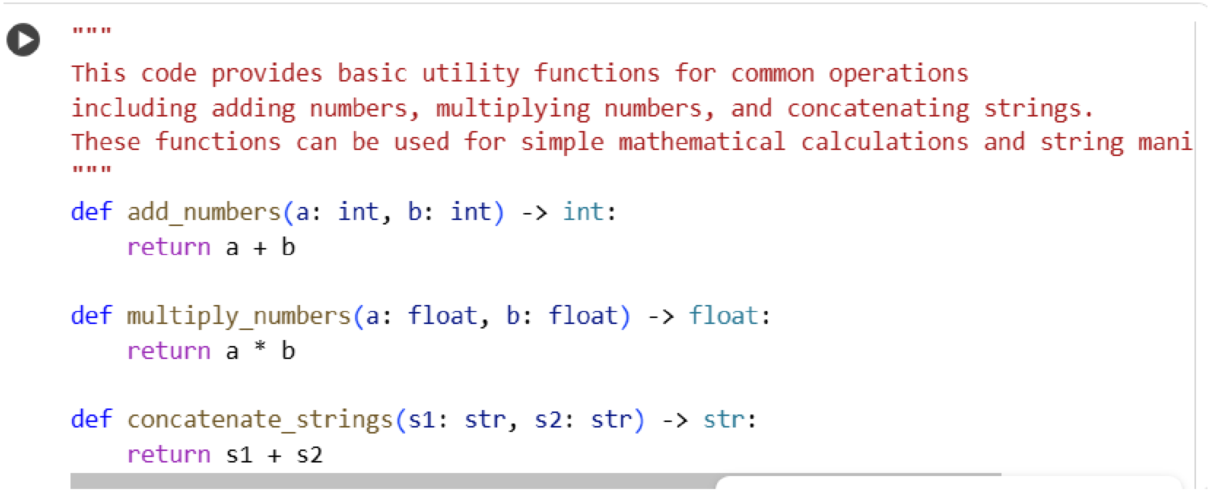
THEN MODIFY THE ABOVE CODE o write a single multi-line docstring at the top of the file. Ensure the docstring clearly describes functionality and usage without rewriting the entire code.

I understand you'd like to add a single multi-line docstring to the top of the code, describing its functionality and usage. I will add this to cell `0kxDlYyNTWnS`, which contains a set of basic utility functions.

Working...

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more



```python
"""
This code provides basic utility functions for common operations
including adding numbers, multiplying numbers, and concatenating strings.
These functions can be used for simple mathematical calculations and string mani
"""
def add_numbers(a: int, b: int) -> int:
    return a + b

def multiply_numbers(a: float, b: float) -> float:
    return a * b

def concatenate_strings(s1: str, s2: str) -> str:
    return s1 + s2
```

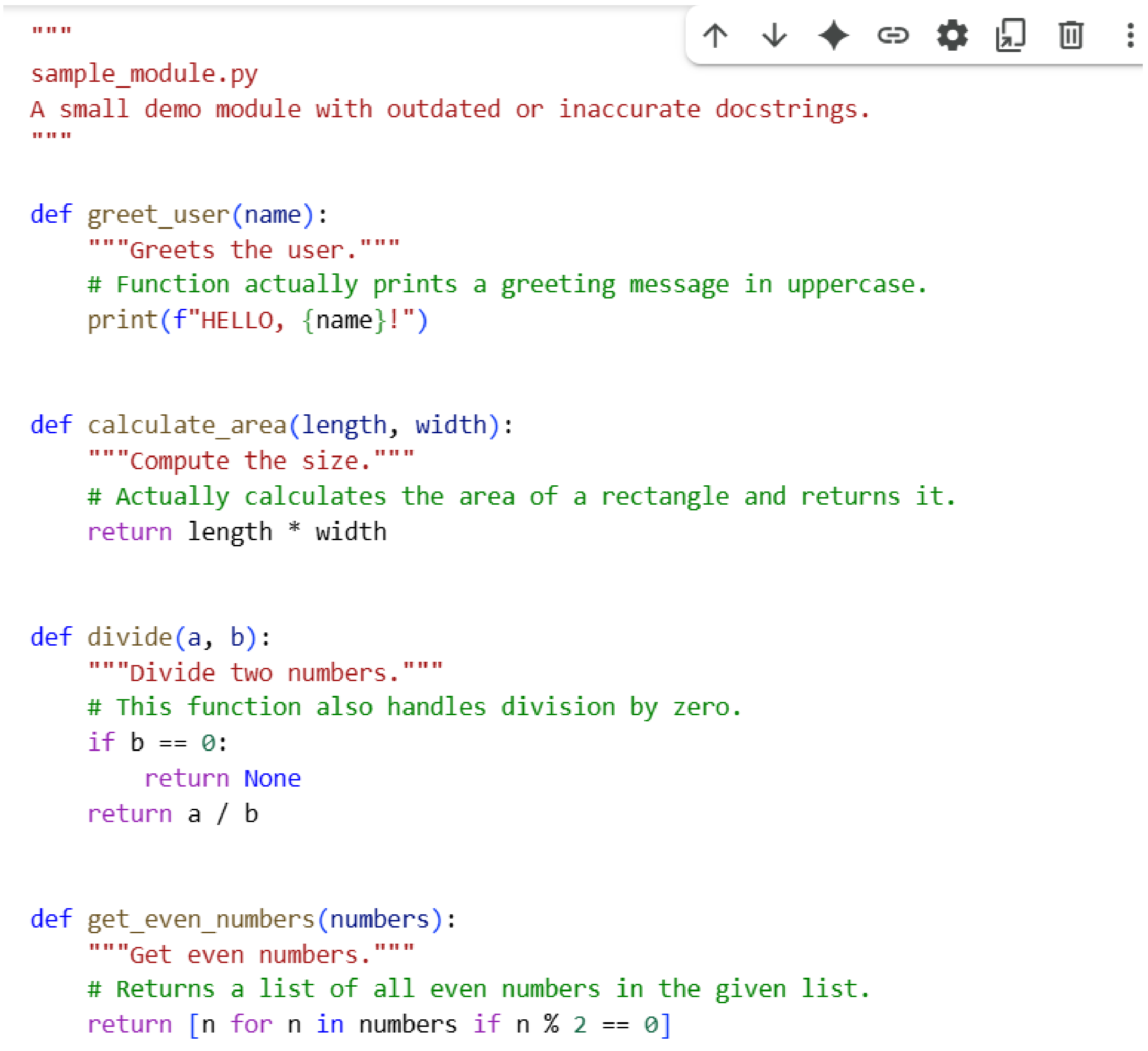
TASK4:

```python
def greet(name):
    """Prints a greeting message."""
    print("Hello,", name)
```

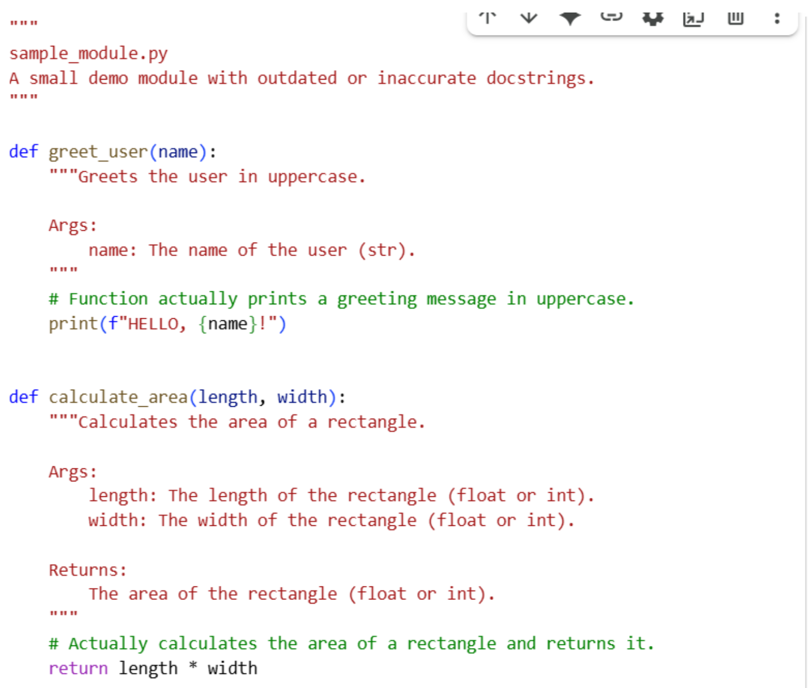
```python
def greet(name):
    """Greets a person by name.

    Args:
        name: The name of the person to greet (a string).
    """
    print("Hello,", name)

# You can call the function like this:
# greet("Colab User")
```

TASK5:

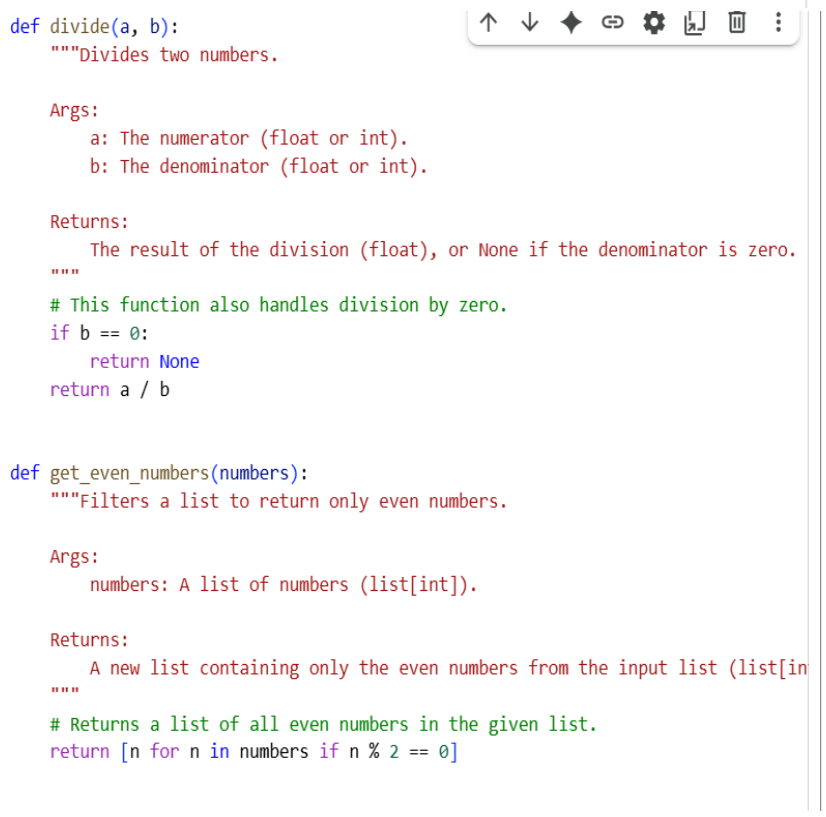
<u>DOCSTRING FILE</u>



```python
"""
sample_module.py
A small demo module with outdated or inaccurate docstrings.
"""

def greet_user(name):
    """Greets the user."""
    # Function actually prints a greeting message in uppercase.
    print(f"HELLO, {name}!")


def calculate_area(length, width):
    """Compute the size."""
    # Actually calculates the area of a rectangle and returns it.
    return length * width


def divide(a, b):
    """Divide two numbers."""
    # This function also handles division by zero.
    if b == 0:
        return None
    return a / b


def get_even_numbers(numbers):
    """Get even numbers."""
    # Returns a list of all even numbers in the given list.
    return [n for n in numbers if n % 2 == 0]
```



```python
"""
sample_module.py
A small demo module with outdated or inaccurate docstrings.
"""

def greet_user(name):
    """Greets the user in uppercase.

    Args:
        name: The name of the user (str).
    """
    # Function actually prints a greeting message in uppercase.
    print(f"HELLO, {name}!")


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

    Args:
        length: The length of the rectangle (float or int).
        width: The width of the rectangle (float or int).

    Returns:
        The area of the rectangle (float or int).
    """
    # Actually calculates the area of a rectangle and returns it.
    return length * width
```



```python
def divide(a, b):
    """Divides two numbers.

    Args:
        a: The numerator (float or int).
        b: The denominator (float or int).

    Returns:
        The result of the division (float), or None if the denominator is zero.
    """
    # This function also handles division by zero.
    if b == 0:
        return None
    return a / b


def get_even_numbers(numbers):
    """Filters a list to return only even numbers.

    Args:
        numbers: A list of numbers (list[int]).

    Returns:
        A new list containing only the even numbers from the input list (list[in
    """
    # Returns a list of all even numbers in the given list.
    return [n for n in numbers if n % 2 == 0]
```

```python
def power(base, exponent=2):
    """Raises a base number to a given exponent.

    Args:
        base: The base number (float or int).
        exponent: The exponent (float or int), defaults to 2.

    Returns:
        The result of the exponentiation (float or int).
    """
    # Default exponent is 2, so it also works as a square() helper.
    return base ** exponent
```