

# AI ASSISTED CODING:

NAME:N.JYOSHNA SRI

HALL.NO:2403A52106

## LAB ASSIGNMENT – 7.3:

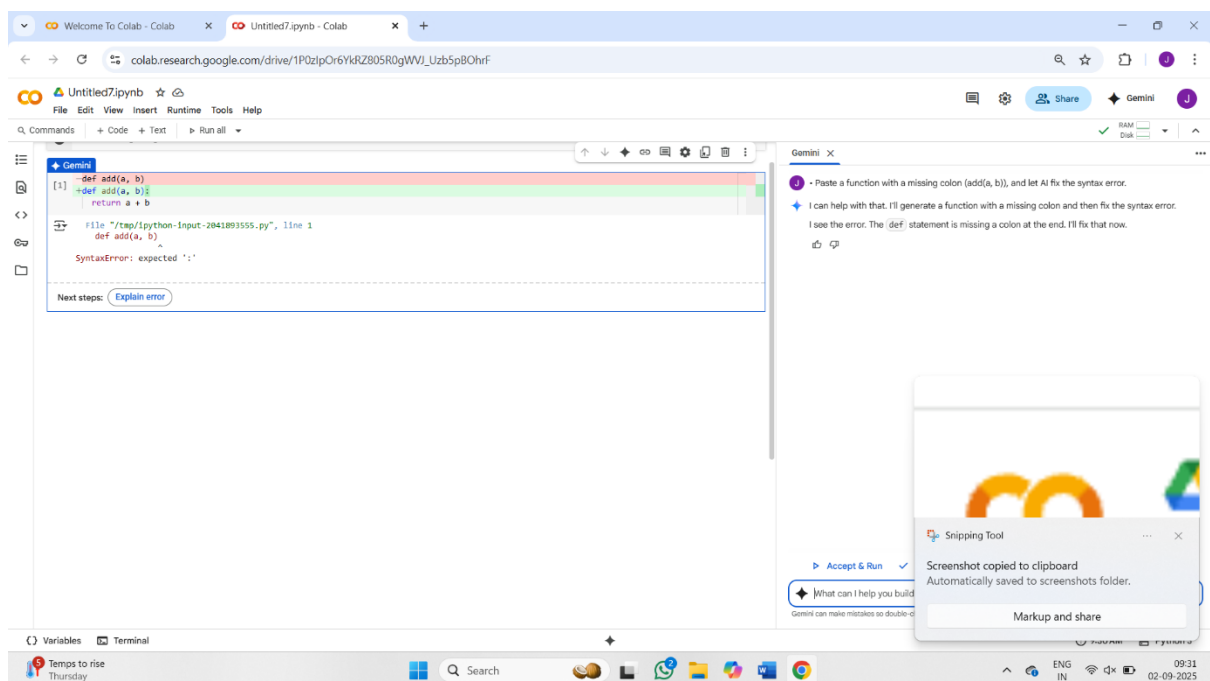
### TASK-1:

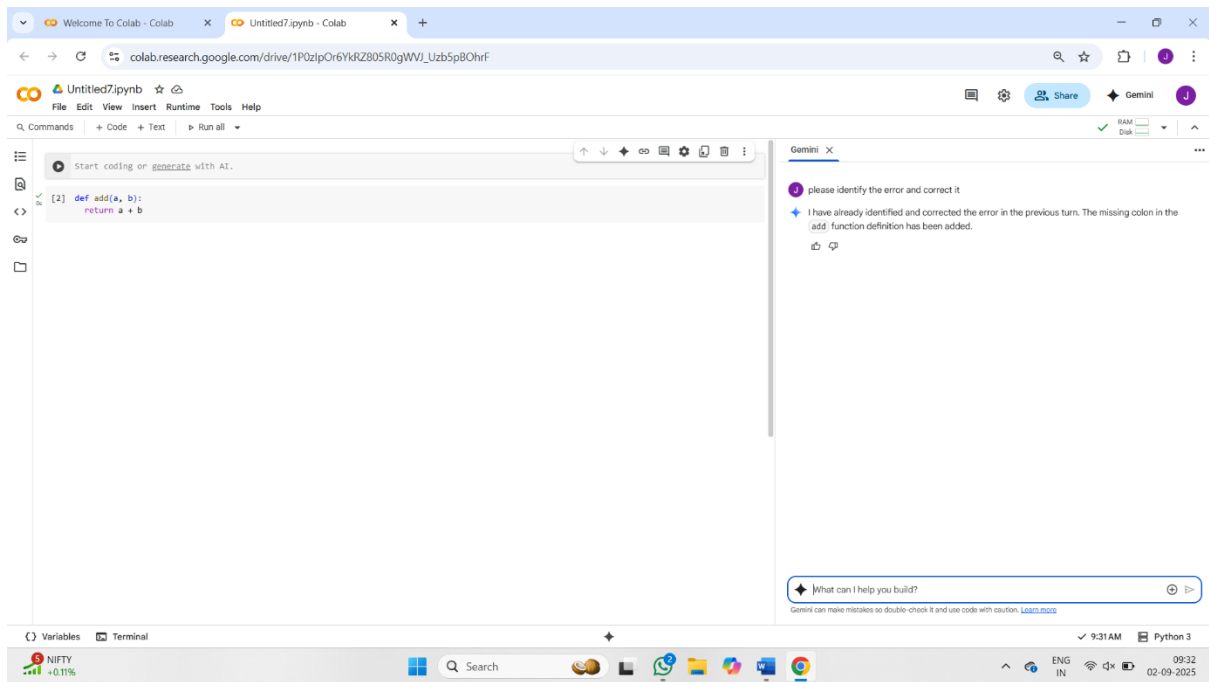
- Paste a function with a missing colon (add(a, b)), and let AI fix the syntax error.

```
python

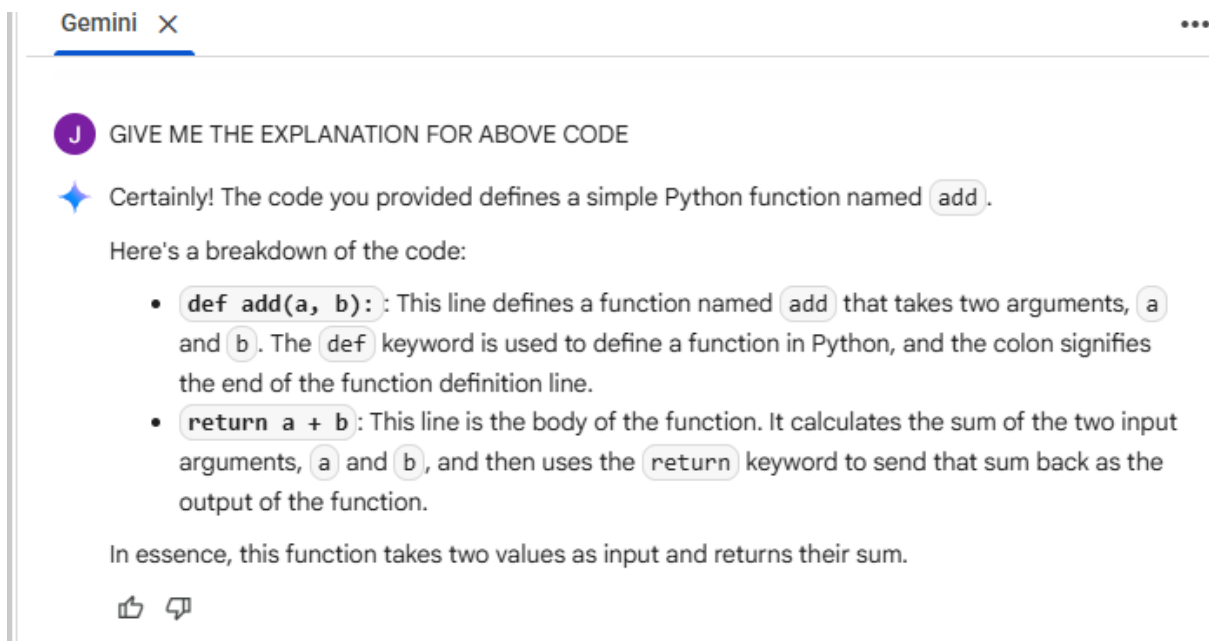
def add(a, b)
    return a + b
```

## CODE SCREENSHOTS BY USING GEMINI:

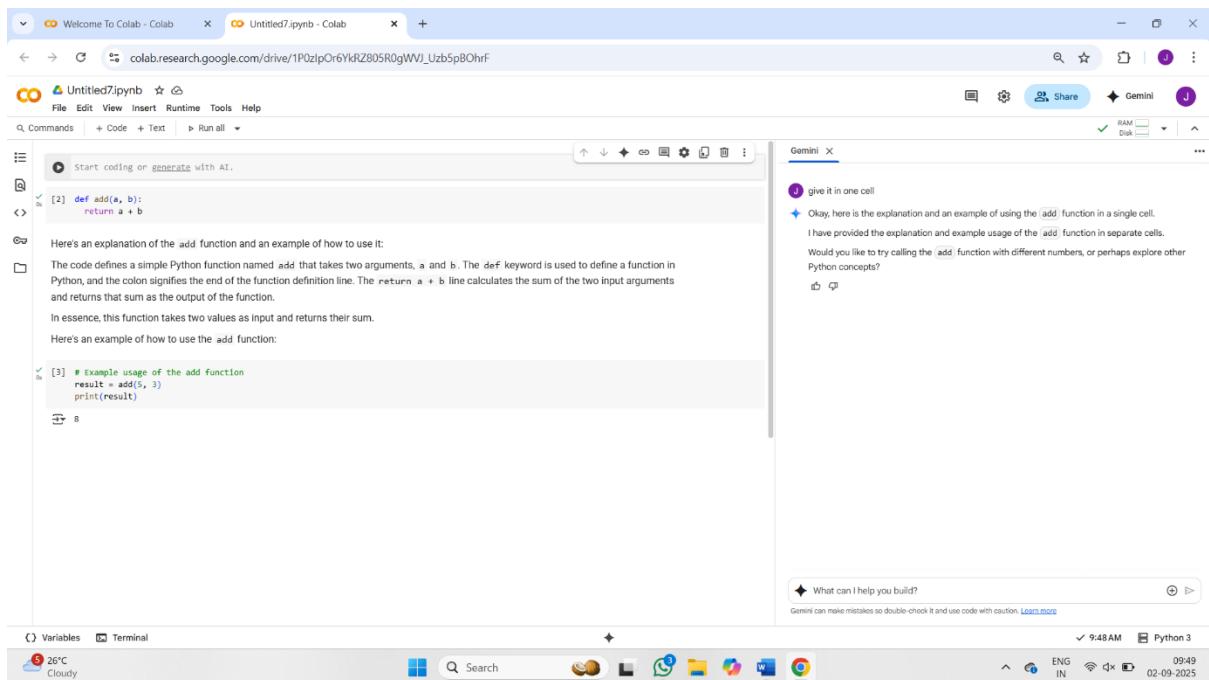




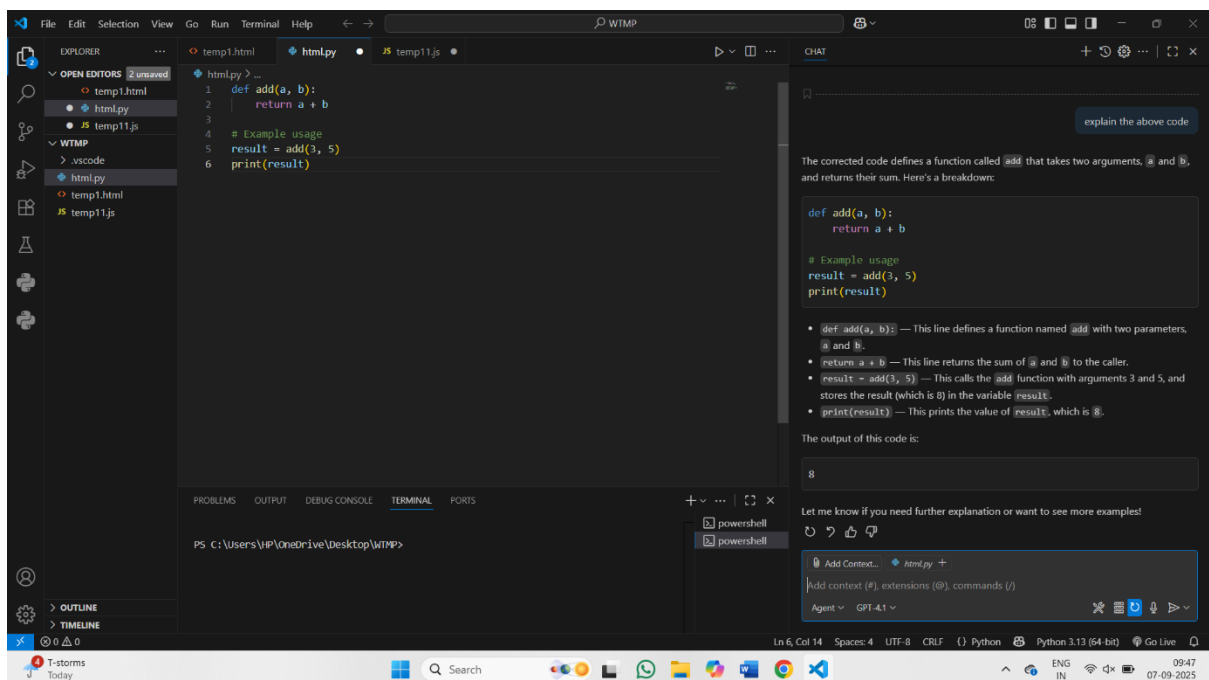
## CODE EXPLANATION:



## Code output with explanation:



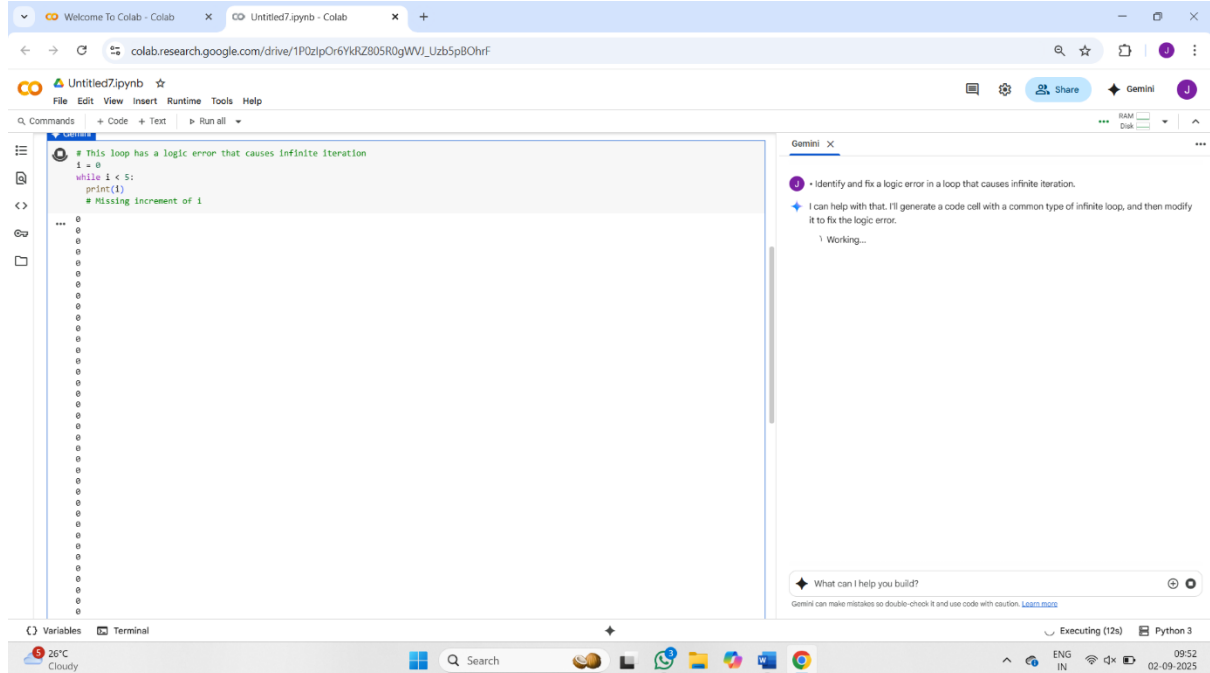
## Code output screenshots in copilot with code explanation:



## TASK-2:

- Identify and fix a logic error in a loop that causes infinite iteration.

```
def count_down(n):  
    while n >= 0:  
        print(n)  
        n += 1  # Should be n -= 1
```



J explain the above code

★ Sure! I can explain the code above.

The code defines a function called `count_down` that takes one argument, `n`.

- **Original (infinite loop) version:**

```
def count_down(n):  
    while n > 0:  
        print(n)
```

In this version, the `while n > 0:` loop continues as long as `n` is greater than 0. However, since `n` is never changed inside the loop, if you call `count_down` with a positive value for `n` (like `count_down(5)`), `n` will always be greater than 0, and the loop will run forever, printing the same value of `n` repeatedly. This is an infinite loop.

- **Corrected version:**

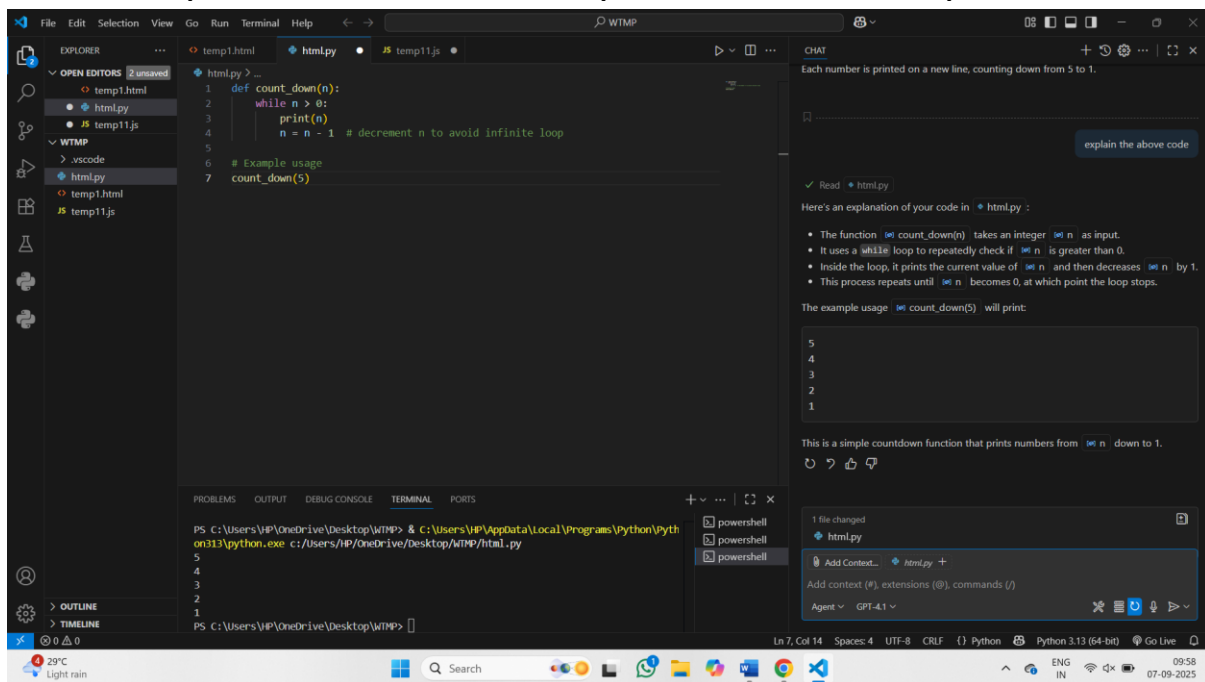
```
def count_down(n):  
    while n > 0:  
        print(n)  
        n -= 1
```

In this corrected version, the line `n -= 1` is added inside the `while` loop. This line subtracts 1 from the value of `n` in each iteration. Now, with each pass through the loop, `n` gets smaller. Eventually, `n` will become 0 or less, and the condition `n > 0` will become false, causing the loop to terminate. This allows the function to count down from the initial value of `n` to 1.

The example usage `count_down(5)` demonstrates how the corrected function works, printing the numbers 5, 4, 3, 2, and 1.



## Code output screenshots in copilot with code explanation:



## TASK-3:

- Debug a runtime error caused by division by zero. Let AI insert try-except.

```
# Debug the following code
def divide(a, b):
    return a / b

print(divide(10, 0))
```

## CODE SCREENSHOTS:

Welcome To Colab - Colab x Untitled9.ipynb - Colab x +

colab.research.google.com/drive/1kMyuXsoo9JqNbYdp2MEUeO2c98my1Df#scrollTo=qVnqK461tFjJ

Untitled9.ipynb

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text Run all

```
def divide(a, b):
    return a / b

# This will cause a ZeroDivisionError
result = divide(10, 0)
print(result)
```

The code above will cause a ZeroDivisionError: If you try to divide by zero. To prevent this runtime error, you can use a try-except block. The try block contains the code that might cause an error, and the except block contains the code to handle the error if it occurs.

```
[4] def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
        return None # Or you could return a specific value like float('inf') or a message

# Example usage:
result1 = divide(10, 2)
print(f"10 / 2 = {result1}")

result2 = divide(10, 0)
print(f"10 / 0 = {result2}")
```

```
10 / 2 = 5.0
Error: Cannot divide by zero
10 / 0 = None
```

Gemini

block in the divide function.

Do you have any other code you would like to debug or any other questions about error handling in Python?

explain the above code

Certainly! Let me explain the code I provided to handle the division by zero error.

The code defines a function called `divide` that takes two arguments, `a` and `b`. Its purpose is to divide `a` by `b`.

Here's a breakdown of the corrected code with the `try-except` block:

```
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
        return None # Or you could return a specific value like float('inf') or a message
```

- `def divide(a, b):`: This line defines the function named `divide` that accepts two parameters, `a` and `b`.
- `try:`: This keyword starts a `try` block. The code inside this block is what the program will attempt to execute. It's where you put the code that might cause an error. In this case, the division `a / b` is the operation that could potentially raise a `ZeroDivisionError`.
- `return a / b:` If the division `a / b` is successful (i.e., `b` is not zero), the result of the division is returned by the function.
- `except ZeroDivisionError:`: This keyword starts an `except` block. This block is executed only if a `ZeroDivisionError` occurs within the preceding `try` block. It

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal

25°C Cloudy

Search

10:13 AM Python 3

ENG IN 10:24 02-09-2025

Code explanation:

J explain the above code

✦ Certainly! Let me explain the code I provided to handle the division by zero error.

The code defines a function called `divide` that takes two arguments, `a` and `b`. Its purpose is to divide `a` by `b`.

Here's a breakdown of the corrected code with the `try-except` block:

```
def divide(a, b):  
    try:  
        return a / b  
    except ZeroDivisionError:  
        print("Error: Cannot divide by zero!")  
        return None # Or you could return a specific value like float('inf') or a me
```

- `def divide(a, b):`: This line defines the function named `divide` that accepts two parameters, `a` and `b`.
- `try:`: This keyword starts a `try` block. The code inside this block is what the program will attempt to execute. It's where you put the code that *might* cause an error. In this case, the division `a / b` is the operation that could potentially raise a `ZeroDivisionError`.
- `return a / b`: If the division `a / b` is successful (i.e., `b` is not zero), the result of the division is returned by the function.
- `except ZeroDivisionError:`: This keyword starts an `except` block. This block is executed *only* if a `ZeroDivisionError` occurs within the preceding `try` block. It specifically catches the `ZeroDivisionError`.
- `print("Error: Cannot divide by zero!")`: If a `ZeroDivisionError` is caught, this line prints an informative error message to the console.
- `return None`: After printing the error message, this line returns `None` from the function. This is one way to indicate that the division was not successful due to the error. You could

✦ What can I help you build?



Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)





RAM   
Disk 



Gemini X



executed *only* if a `ZeroDivisionError` occurs within the preceding `try` block. It specifically catches the `ZeroDivisionError`.

- `print("Error: Cannot divide by zero!")`: If a `ZeroDivisionError` is caught, this line prints an informative error message to the console.
- `return None`: After printing the error message, this line returns `None` from the function. This is one way to indicate that the division was not successful due to the error. You could choose to return a different value or handle the error in another way depending on your program's needs (e.g., return `float('inf')` for positive infinity or raise a different exception).

The example usage shows how the function behaves:



```
# Example usage:
result1 = divide(10, 2)
print(f"10 / 2 = {result1}")

result2 = divide(10, 0)
print(f"10 / 0 = {result2}")
```

When `divide(10, 2)` is called, the `try` block executes successfully, and the result `5.0` is returned and printed.

When `divide(10, 0)` is called, the division `10 / 0` inside the `try` block raises a `ZeroDivisionError`. The program then jumps to the `except ZeroDivisionError:` block, prints "Error: Cannot divide by zero!", and returns `None`, which is then printed.

This `try-except` block allows the program to handle the potential error gracefully instead of crashing.

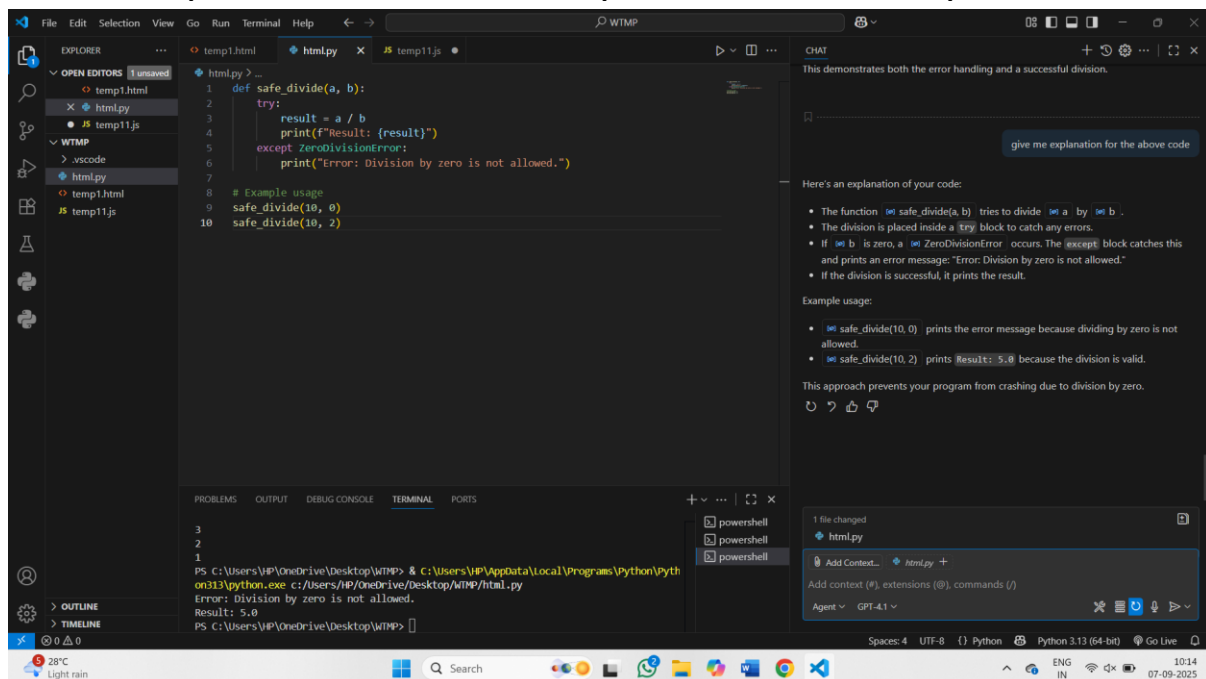
Let me know if you have any more questions!



What can I help you build?



## Code output screenshots in copilot with code explanation:



## TASK-4:

- Provide a faulty class definition (missing self in parameters). Let AI fix it

```
python

class Rectangle:
    def __init__(length, width):
        self.length = length
        self.width = width
```

## Code screenshots:

The screenshot displays a Google Colab notebook titled 'Untitled9.ipynb'. The notebook contains three code cells. The first cell defines a `rectangle` class with an `__init__` method that takes `length` and `width` as arguments and assigns them to `self.length` and `self.width`. The second cell creates an instance `my_rectangle` with `length=10` and `width=5`, and prints its attributes. The third cell repeats the same code. The output of the second and third cells shows the printed attributes: 'The length of the rectangle is: 10' and 'The width of the rectangle is: 5'. On the right side, the Gemini AI chat interface is visible, showing a user prompt 'give in one cell' and a response from Gemini providing the example usage of the `rectangle` class in a single cell. The bottom of the screen shows the Windows taskbar with the date and time as 10:43 AM on 02-09-2025.

```
class rectangle:
    def __init__(self, length, width): # Corrected __init__ and added self
        self.length = length # Corrected capitalization of self
        self.width = width

rect = rectangle(5, 3) # Corrected indentation and placed outside the class definition

# Create a rectangle object with length 10 and width 5
my_rectangle = rectangle(10, 5)

# Access and print the length and width
print(f"The length of the rectangle is: {my_rectangle.length}")
print(f"The width of the rectangle is: {my_rectangle.width}")

The length of the rectangle is: 10
The width of the rectangle is: 5

# Create a rectangle object with length 10 and width 5
my_rectangle = rectangle(10, 5)

# Access and print the length and width
print(f"The length of the rectangle is: {my_rectangle.length}")
print(f"The width of the rectangle is: {my_rectangle.width}")

The length of the rectangle is: 10
The width of the rectangle is: 5

class rectangle:
    def __init__(self, length, width): # Corrected __init__ and added self
        self.length = length # Corrected capitalization of self
        self.width = width

    my_rectangle = rectangle(10, 5)

    print(f"The length of the rectangle is: {my_rectangle.length}")
    print(f"The width of the rectangle is: {my_rectangle.width}")

The length of the rectangle is: 10
The width of the rectangle is: 5
```

give in one cell

Certainly! Here is the example usage of the `rectangle` class in a single cell, as you requested. I have provided the example usage of the `rectangle` class in a single cell, as you requested. Is there anything else you would like to do or learn about Python classes or objects?

What can I help you build?

Code explanation:

✓

RAM

Disk

▼

^

Gemini ✕

```
# Create a rectangle object with length 10 and width 5
my_rectangle = rectangle(10, 5)

# Access and print the length and width
print(f"The length of the rectangle is: {my_rectangle.length}")
print(f"The width of the rectangle is: {my_rectangle.width}")
```

Here's what each part does:

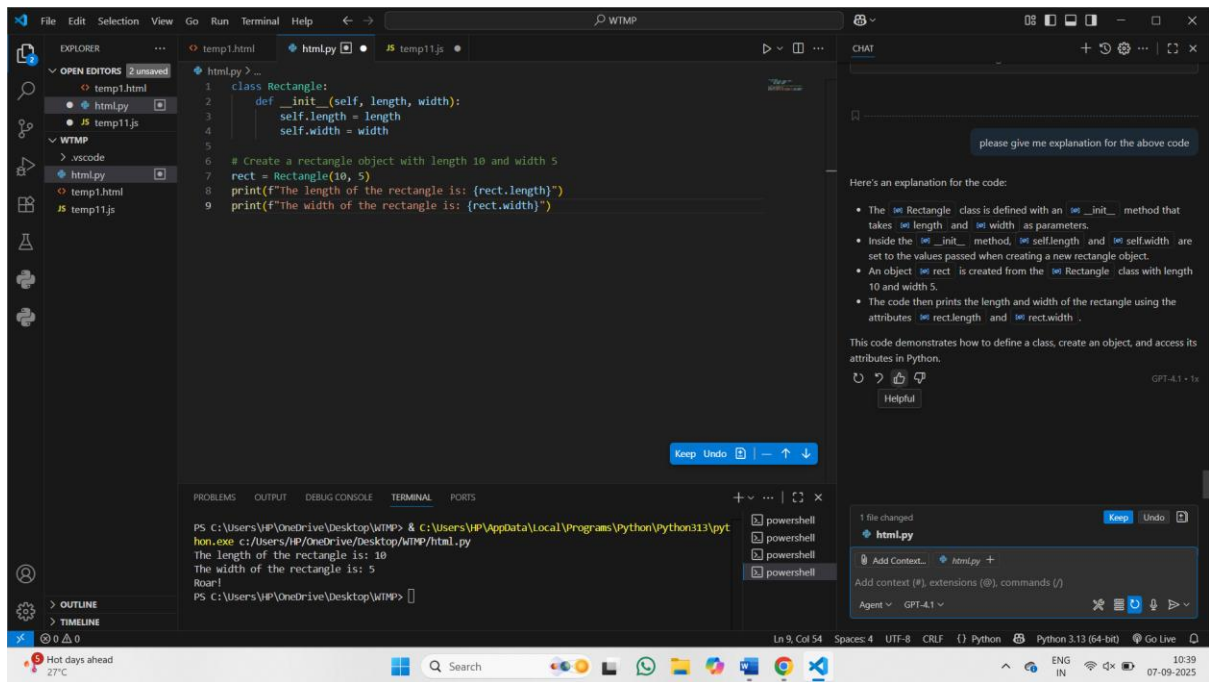
- `my_rectangle = rectangle(10, 5):`
  - This line creates a new instance (or object) of the `rectangle` class.
  - When you call the class name followed by parentheses (`rectangle(10, 5)`), you are actually calling the `__init__` method of the class.
  - The values `10` and `5` are passed as arguments to the `__init__` method. Inside `__init__`, these values are assigned to the `length` and `width` attributes of the newly created `my_rectangle` object.
  - The resulting `rectangle` object is then assigned to the variable `my_rectangle`.
- `print(f"The length of the rectangle is: {my_rectangle.length}"):` 
  - This line accesses the `length` attribute of the `my_rectangle` object using dot notation (`my_rectangle.length`).
  - Since we created the object with a length of 10, this will retrieve the value 10.
  - The `f""` syntax is a f-string, which allows you to embed the value of a variable directly within a string. The output will be "The length of the rectangle is: 10".
- `print(f"The width of the rectangle is: {my_rectangle.width}"):` 
  - Similar to the previous line, this accesses the `width` attribute of the `my_rectangle` object using `my_rectangle.width`.
  - Since we created the object with a width of 5, this will retrieve the value 5.
  - Using the f-string, the output will be "The width of the rectangle is: 5".

◆

What can I help you build?

⊕ ▶

Code output screenshots in copilot with code explanation:



## TASK-5:

- Access an invalid list index and use AI to resolve the Index Error.

```
python

numbers = [1, 2, 3]
print(numbers[5])
```

Code screenshots by using gemini:

The screenshot shows a Google Colab notebook titled 'Untitled13.ipynb'. The code in the cell is as follows:

```
[5]: # Create a list
my_list = [1, 2, 3]

# Access a valid index (e.g., index 0)
print(my_list[0])
```

The output of the cell is '1'. To the right of the code editor, the Gemini chat interface is open. It shows a message explaining an `IndexError` that would occur if the user tried to access `my_list[5]`. It provides the corrected code:

```
# Create a list
my_list = [1, 2, 3]

# Access a valid index (e.g., index 0)
print(my_list[0])
```

The chat also explains that the first element in the list is at index 0, so `my_list[0]` returns the value 1. At the bottom of the chat, there is a prompt: 'What can I help you build?'.

The bottom of the image shows the Windows taskbar with the date and time '10:44 07-09-2025'.

Code explanation by using gemini:

- ◆ The error in the code is an `IndexError` because you are trying to access an index (5) that is outside the valid range of the list `my_list` (which has indices 0, 1, and 2).

To fix this, you need to access an index within the valid range. Here is the corrected code accessing a valid index (0) and its output:

I have identified and corrected the `IndexError` for you.

Would you like to explore other types of Python errors or work with different data structures like dictionaries or tuples?



- J explain the above code

- ◆ Certainly! The code you provided is a simple Python example that demonstrates how to create a list and access its elements.

Here's a breakdown:

```
# Create a list
my_list = [1, 2, 3]
```



This line creates a list named `my_list` and initializes it with three integer values: 1, 2, and 3. Lists in Python are ordered collections of items, and each item has an index.

```
# Access a valid index (e.g., index 0)
print(my_list[0])
```



This line accesses the element at index 0 of the `my_list`. In Python, list indices start from 0. So, `my_list[0]` refers to the first element in the list, which is the number 1. The `print()` function then displays this value to the console.

Code output screenshots in copilot with code explanation:

