AI ASSISTED CODING

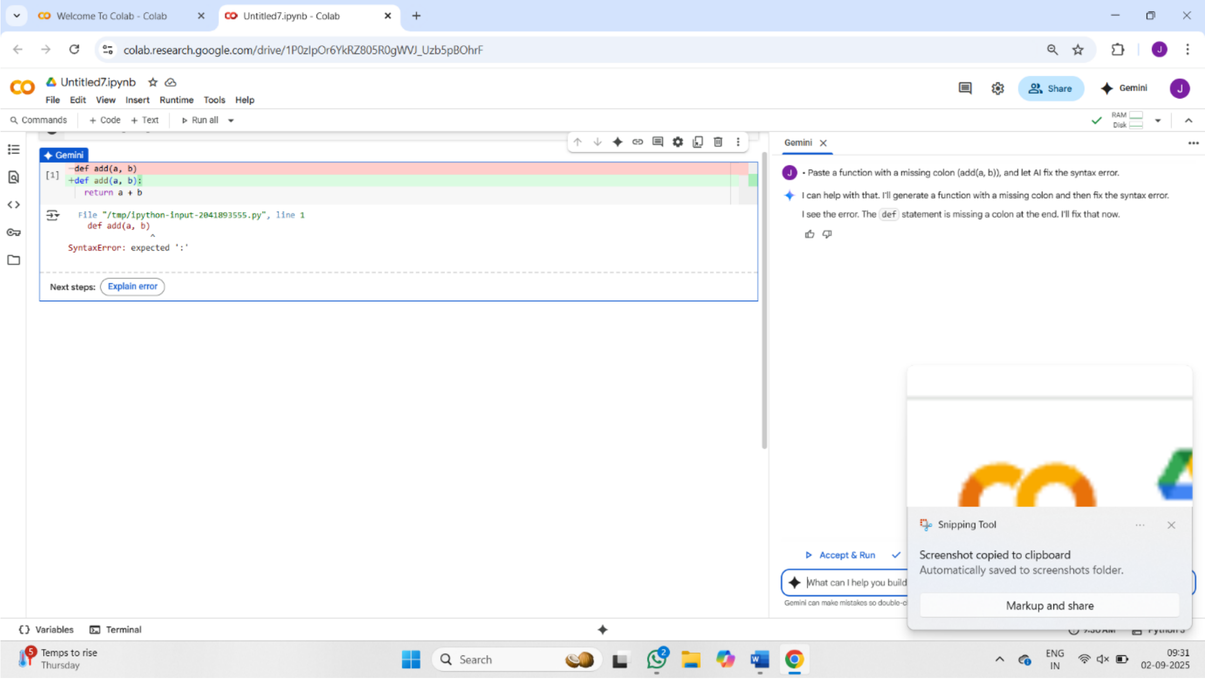
LAB ASSIGNMENT – 7.3:

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:


Welcome To Colab - Colab
Untitled7.ipynb - Colab
colab.research.google.com/drive/1P0zlpOr6YkRZ805R0gWVJ_Uzb5pBOhrF
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
RAM Disk
Gemini

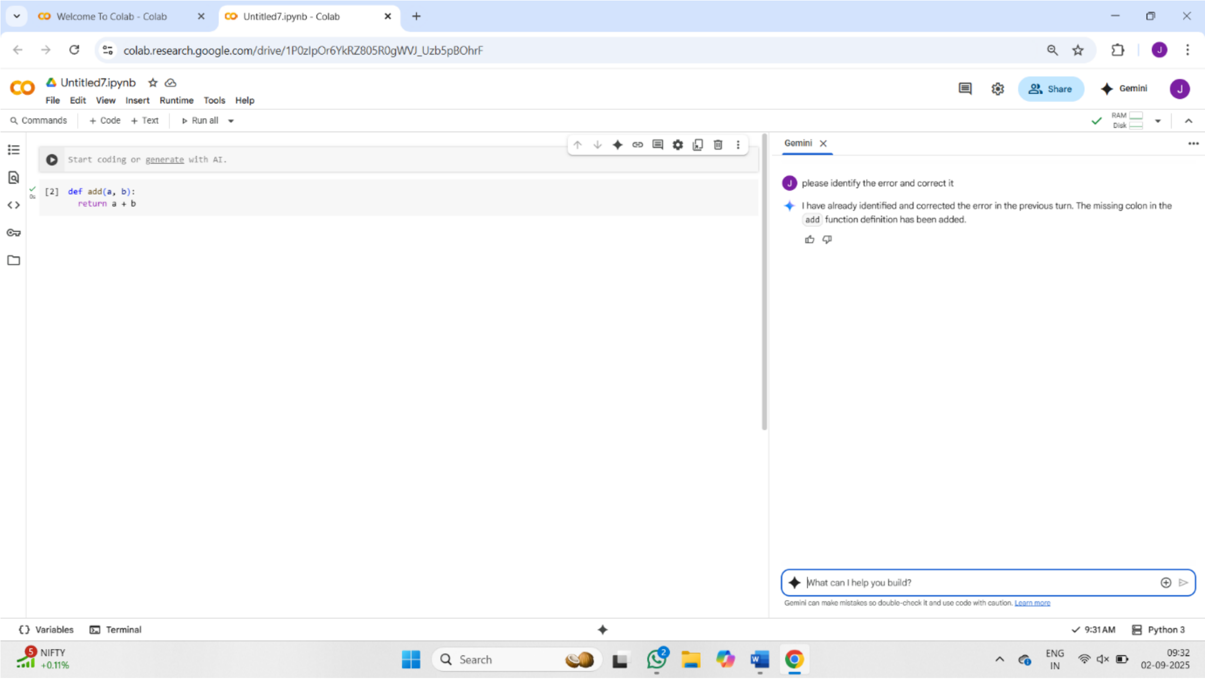
```
[1] -def add(a, b)
    +def add(a, b):
      return a + b

  File "/tmp/ipython-input-2041893555.py", line 1
    def add(a, b)
                 ^
SyntaxError: expected ':'
```

Next steps: Explain error

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

I can help with that. I'll generate a function with a missing colon and then fix the syntax error.

I see the error. The `def` statement is missing a colon at the end. I'll fix that now.

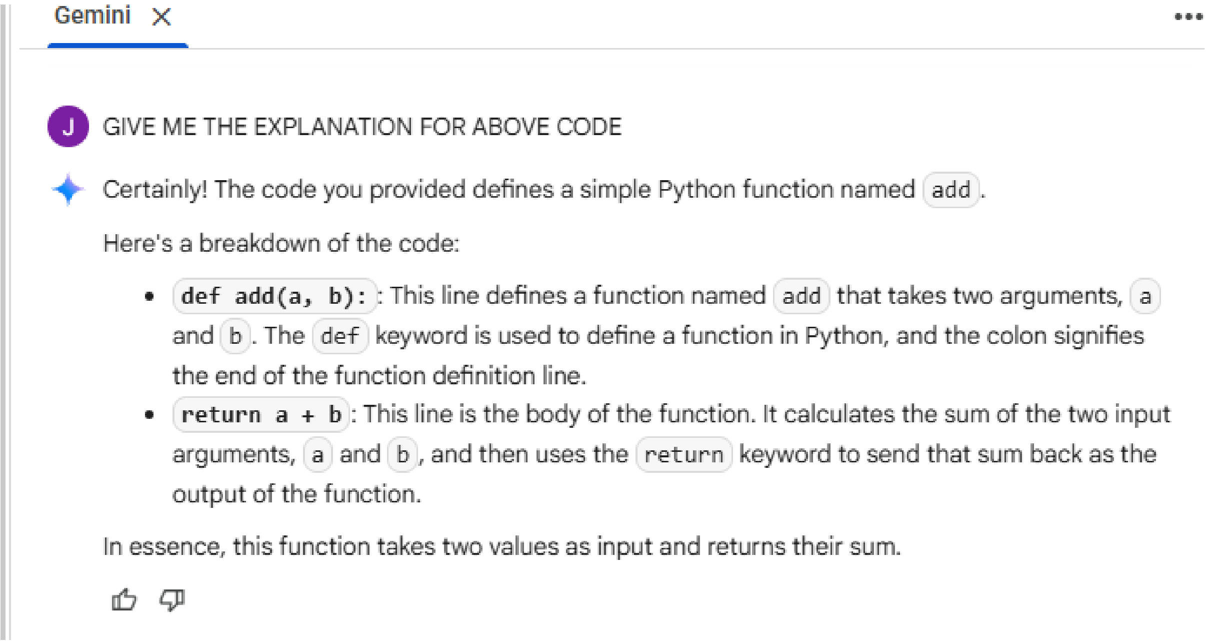

Welcome To Colab - Colab
Untitled7.ipynb - Colab
colab.research.google.com/drive/1P0zIpOr6YkRZ805R0gWVJ_Uzb5pBOhrF
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
Start coding or generate with AI.
Gemini
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more
Variables Terminal
9:31 AM Python 3
NIFTY +0.11%
Search
ENG IN
09:32 02-09-2025

```
[2] def add(a, b):
      return a + b
```

please identify the error and correct it

I have already identified and corrected the error in the previous turn. The missing colon in the `add` function definition has been added.

CODE EXPLANATION:



GIVE ME THE EXPLANATION FOR ABOVE CODE

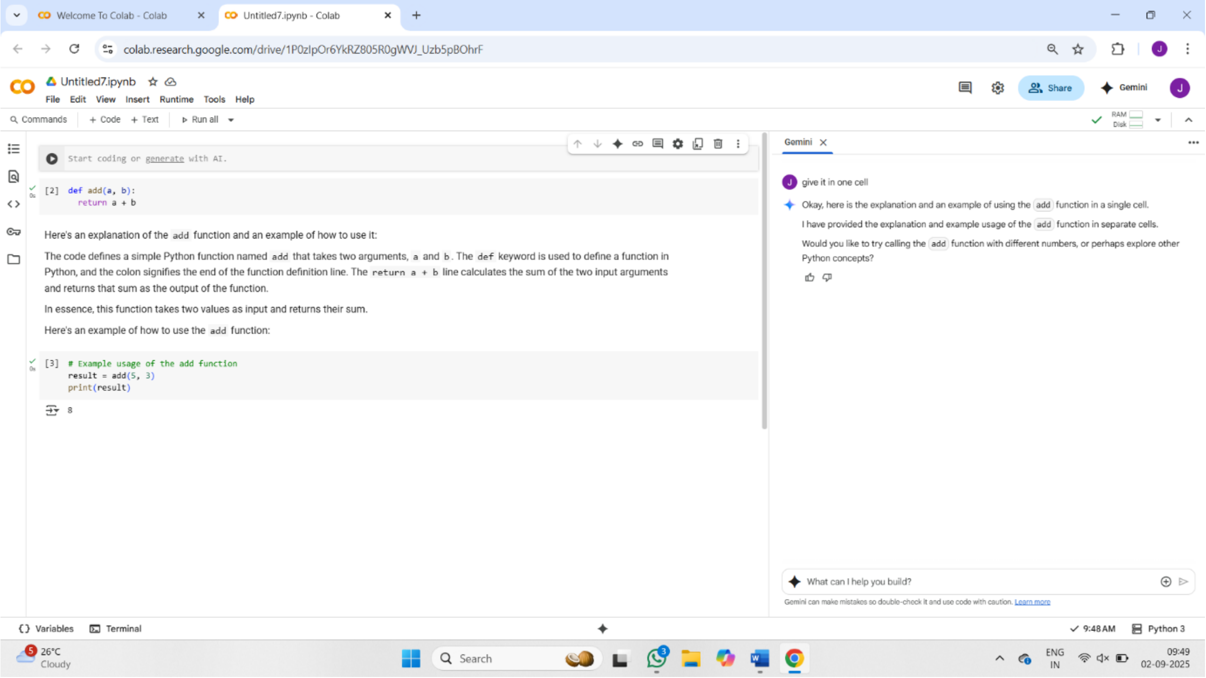
Certainly! The code you provided defines a simple Python function named `add`.

Here's a breakdown of the code:

- `def add(a, b):`: This line defines a function named `add` that takes two arguments, `a` and `b`. The `def` keyword is used to define a function in Python, and the colon signifies the end of the function definition line.
- `return a + b`: This line is the body of the function. It calculates the sum of the two input arguments, `a` and `b`, and then uses the `return` keyword to send that sum back as the output of the function.

In essence, this function takes two values as input and returns their sum.

Code output with explanation:



```
[2] def add(a, b):
      return a + b
```

Here's an explanation of the `add` function and an example of how to use it:

The code defines a simple Python function named `add` that takes two arguments, `a` and `b`. The `def` keyword is used to define a function in Python, and the colon signifies the end of the function definition line. The `return a + b` line calculates the sum of the two input arguments and returns that sum as the output of the function.

In essence, this function takes two values as input and returns their sum.

Here's an example of how to use the `add` function:

```
[3] # Example usage of the add function
    result = add(5, 3)
    print(result)
```

```
8
```

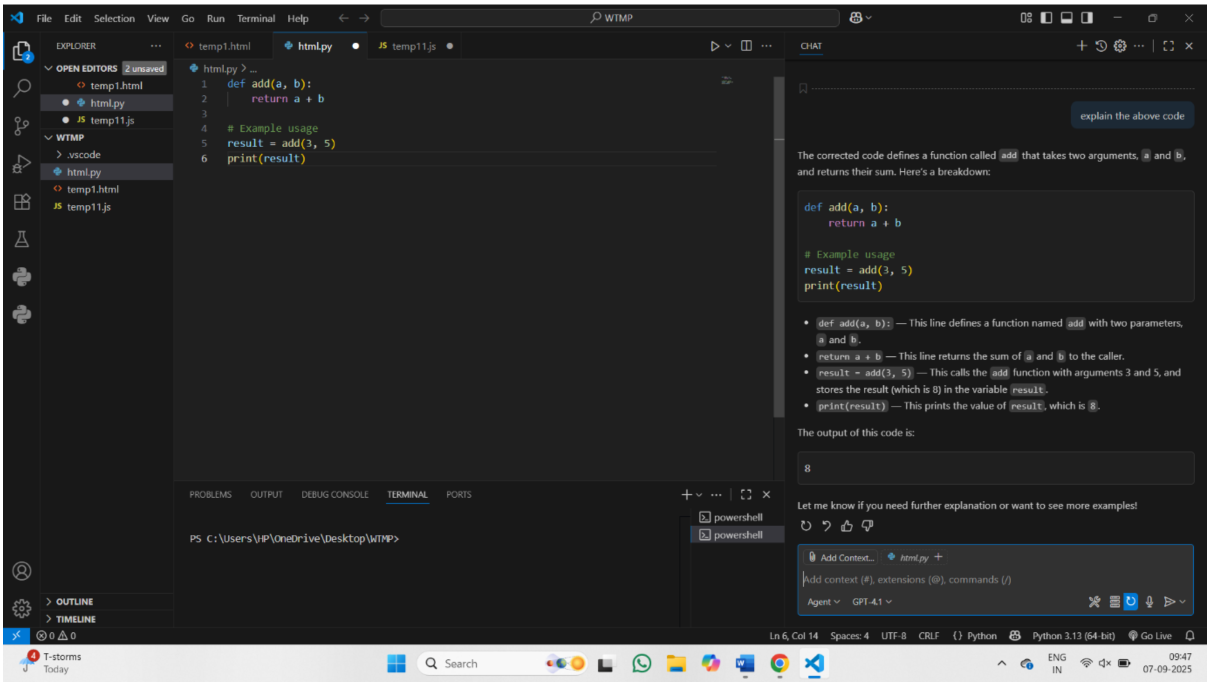
give it in one cell

Okay, here is the explanation and an example of using the `add` function in a single cell.

I have provided the explanation and example usage of the `add` function in separate cells.

Would you like to try calling the `add` function with different numbers, or perhaps explore other Python concepts?

Code output screenshots in copilot with code explanation:



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

# Example usage
result = add(3, 5)
print(result)
```

```
PS C:\Users\HP\OneDrive\Desktop\WTMP>
```

explain the above code

The corrected code defines a function called `add` that takes two arguments, `a` and `b`, and returns their sum. Here's a breakdown:

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

# Example usage
result = add(3, 5)
print(result)
```

- `def add(a, b):` — This line defines a function named `add` with two parameters, `a` and `b`.
- `return a + b` — This line returns the sum of `a` and `b` to the caller.
- `result = add(3, 5)` — This calls the `add` function with arguments 3 and 5, and stores the result (which is 8) in the variable `result`.
- `print(result)` — This prints the value of `result`, which is `8`.

The output of this code is:

```
8
```

Let me know if you need further explanation or want to see more examples!

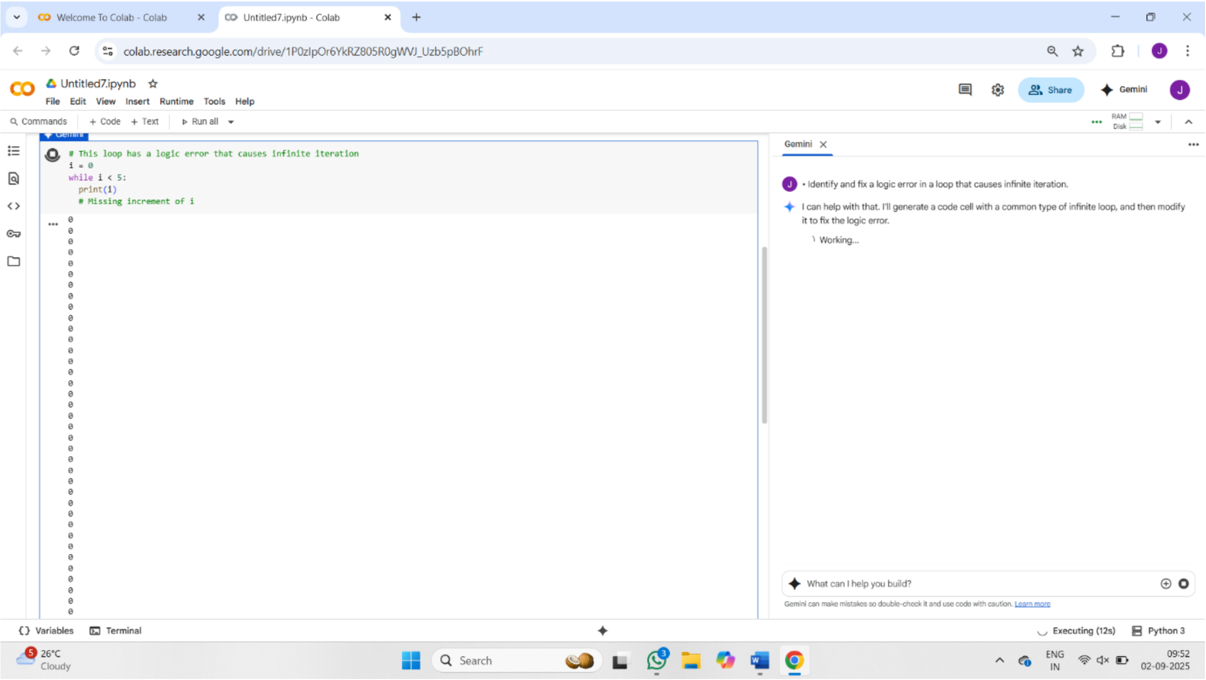
TASK-2:

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

python

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

Code Screenshots:



Welcome To Colab - Colab
Untitled7.ipynb - Colab
colab.research.google.com/drive/1P0zlpOr6YkRZ805R0gWVJ_Uzb5pBOhrF
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands
+ Code
+ Text
Run all
Share
Gemini
RAM
Disk
Gemini

```
# This loop has a logic error that causes infinite iteration
i = 0
while i < 5:
  print(i)
  # Missing increment of i
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

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

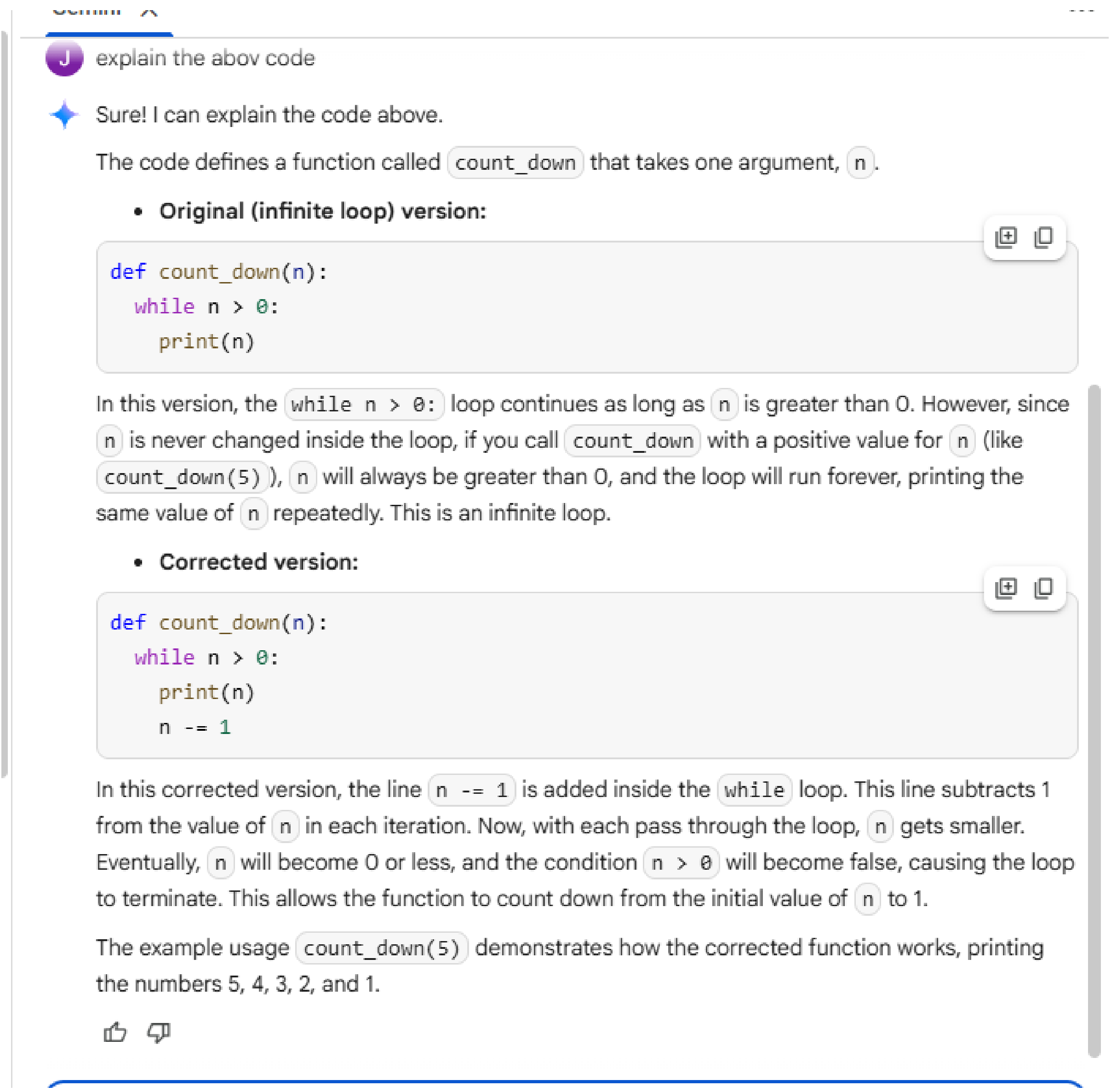
I can help with that. I'll generate a code cell with a common type of infinite loop, and then modify it to fix the logic error.

Working...

What can I help you build?

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

Code explanation by using gemini:



explain the abov 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.

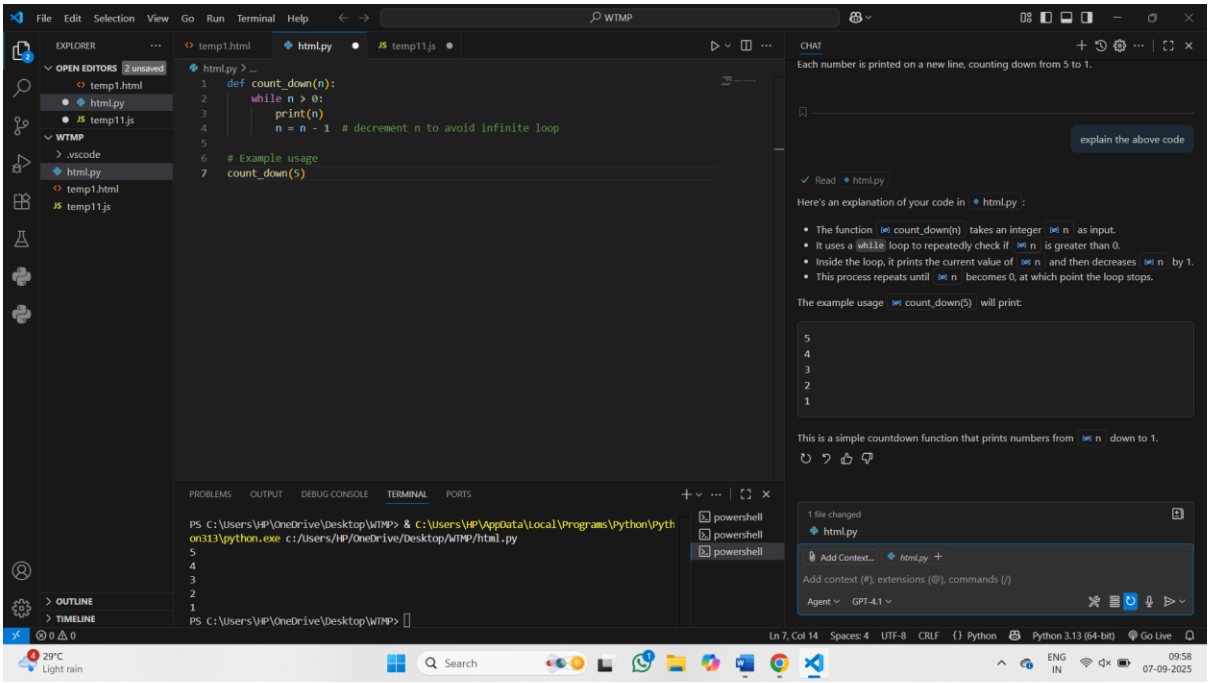
- **Corrected version:**

```
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:



```python
def count_down(n):
    while n > 0:
        print(n)
        n = n - 1  # decrement n to avoid infinite loop

# Example usage
count_down(5)
```

```
PS C:\Users\HP\OneDrive\Desktop\WTMP> & C:\Users\HP\AppData\Local\Programs\Python\Pyth
on313\python.exe c:/Users/HP/OneDrive/Desktop/WTMP/html.py
5
4
3
2
1
PS C:\Users\HP\OneDrive\Desktop\WTMP> 
```

Each number is printed on a new line, counting down from 5 to 1.

explain the above code

Read html.py

Here's an explanation of your code in html.py :

- The function count_down(n) takes an integer n as input.
- It uses a `while` loop to repeatedly check if n is greater than 0.
- Inside the loop, it prints the current value of n and then decreases n by 1.
- This process repeats until n becomes 0, at which point the loop stops.

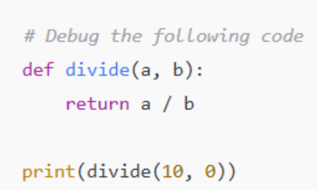
The example usage count_down(5) will print:

```
5
4
3
2
1
```

This is a simple countdown function that prints numbers from n down to 1.

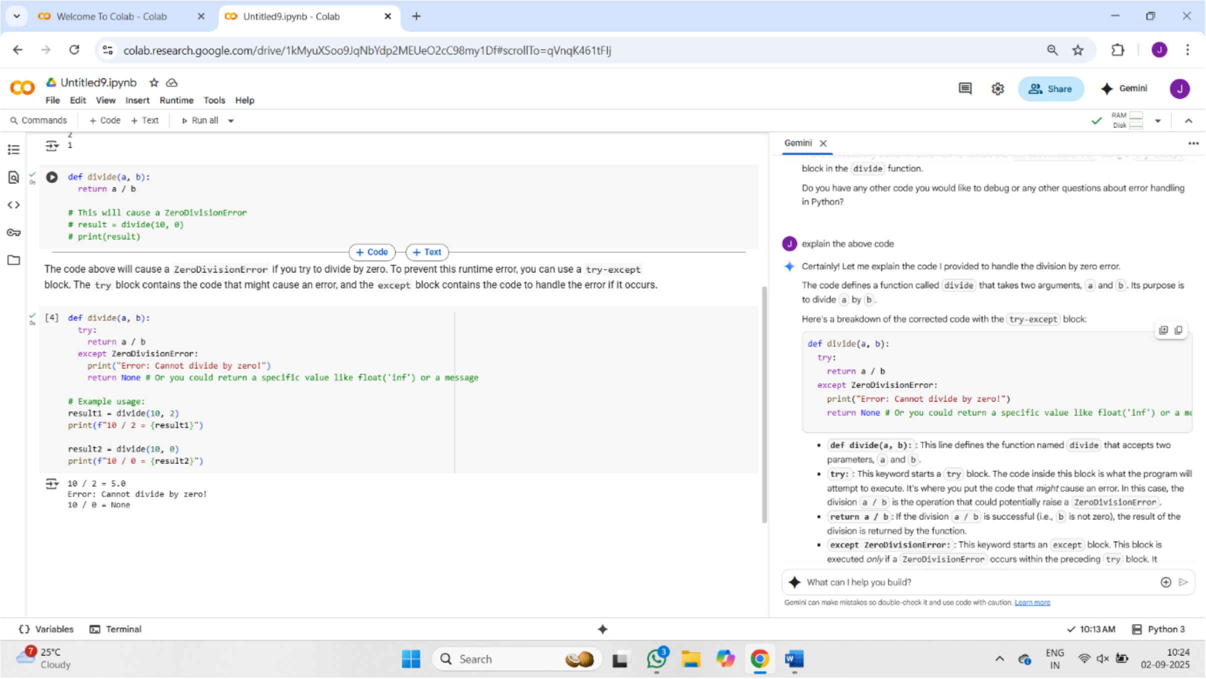
TASK-3:

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



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

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

CODE SCREENSHOTS:



```
2
1
```

```python
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.

```python
[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?

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:

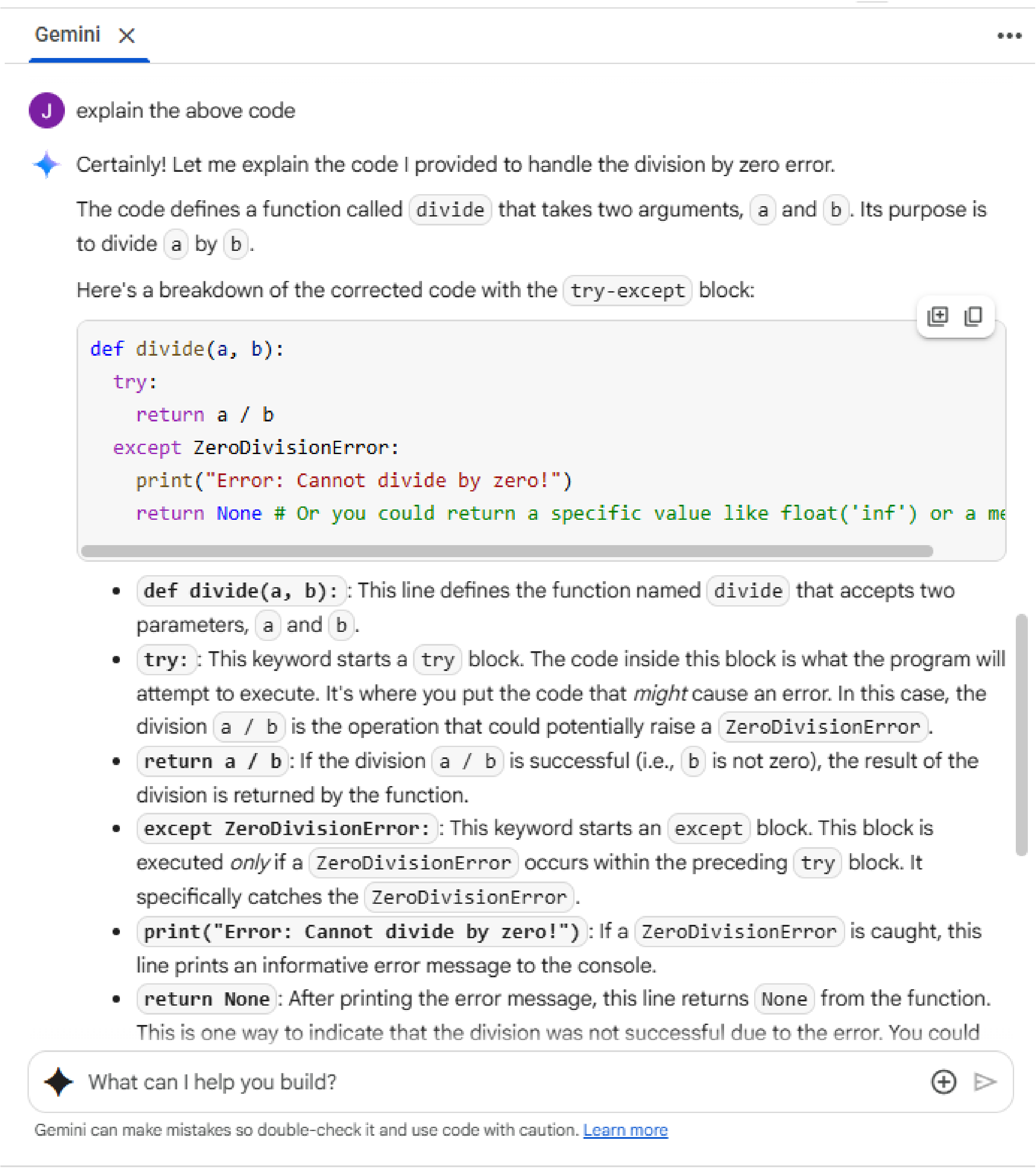
```python
def divide(a, b):
  try:
    return a / b
  except ZeroDivisionError:
    print("Error: Cannot divide by zero!")
    return None # Or you could return a specific value like float('inf') or a me
```

- **`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

What can I help you build?

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

Code explanation:



explain the above code

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

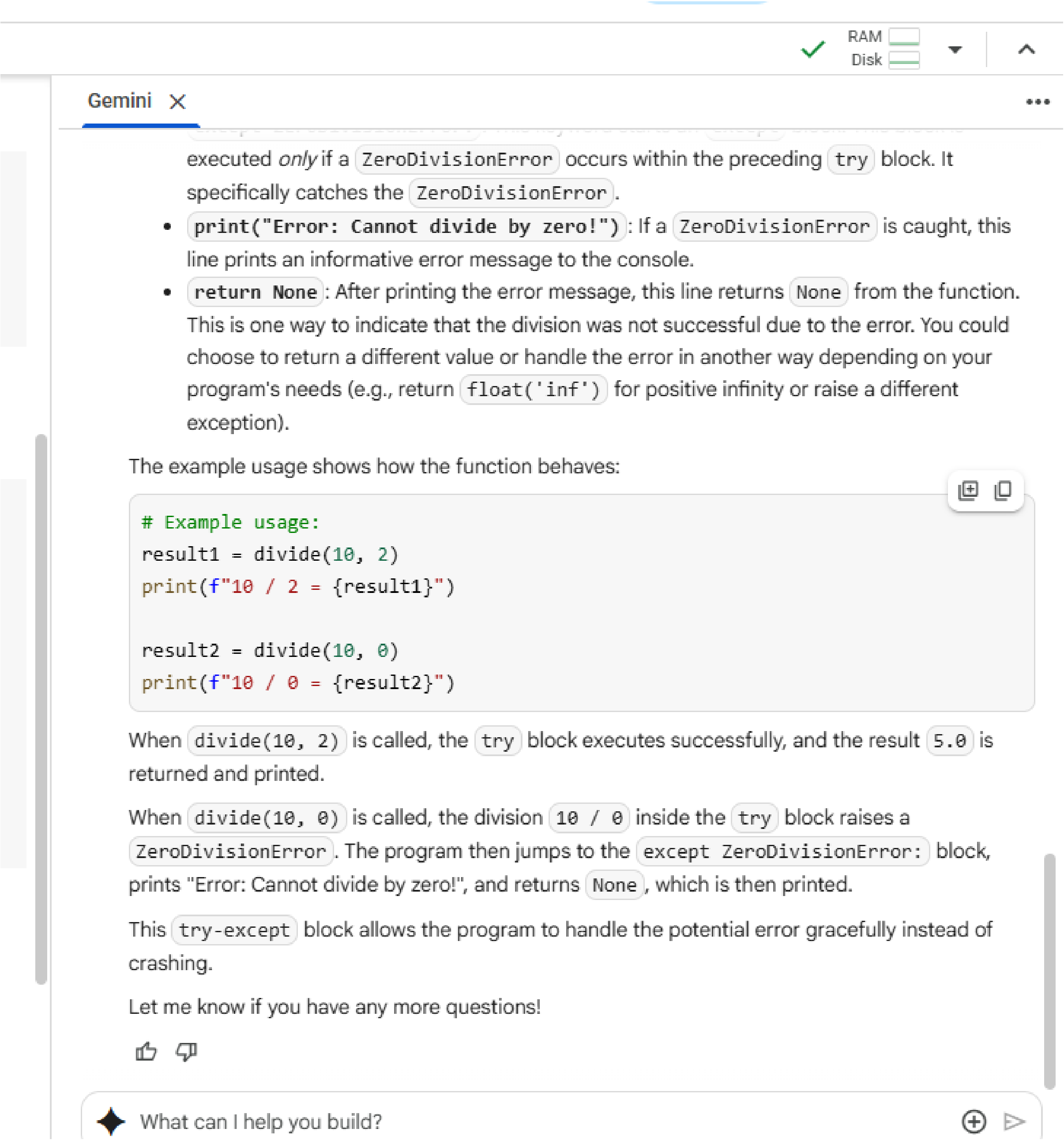
The code defines a function called `divide` that takes two arguments, `a` and `b`. Its purpose is to divide `a` by `b`.

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

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



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}")
```

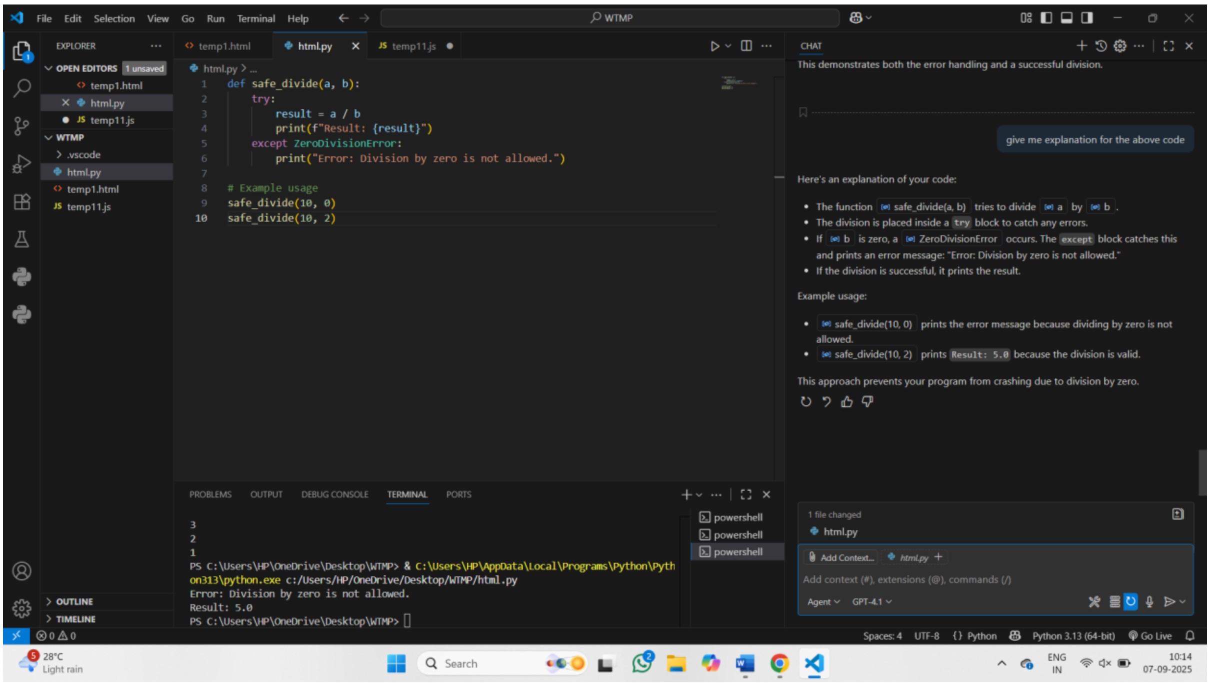
When `divide(10, 2)` is called, the `try` block executes successfully, and the result `5.0` is returned and printed.

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!

Code output screenshots in copilot with code explanation:



```python
def safe_divide(a, b):
    try:
        result = a / b
        print(f"Result: {result}")
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed.")

# Example usage
safe_divide(10, 0)
safe_divide(10, 2)
```

```
3
2
1
PS C:\Users\HP\OneDrive\Desktop\WTMP> & C:\Users\HP\AppData\Local\Programs\Python\Pyth
on313\python.exe c:/Users/HP/OneDrive/Desktop/WTMP/html.py
Error: Division by zero is not allowed.
Result: 5.0
PS C:\Users\HP\OneDrive\Desktop\WTMP> 
```

This demonstrates both the error handling and a successful division.

give me explanation for the above code

Here's an explanation of your code:

- The function safe_divide(a, b) tries to divide a by b .
- The division is placed inside a `try` block to catch any errors.
- If b is zero, a ZeroDivisionError occurs. The `except` block catches this and prints an error message: "Error: Division by zero is not allowed."
- If the division is successful, it prints the result.

Example usage:

- safe_divide(10, 0) prints the error message because dividing by zero is not allowed.
- safe_divide(10, 2) prints `Result: 5.0` because the division is valid.

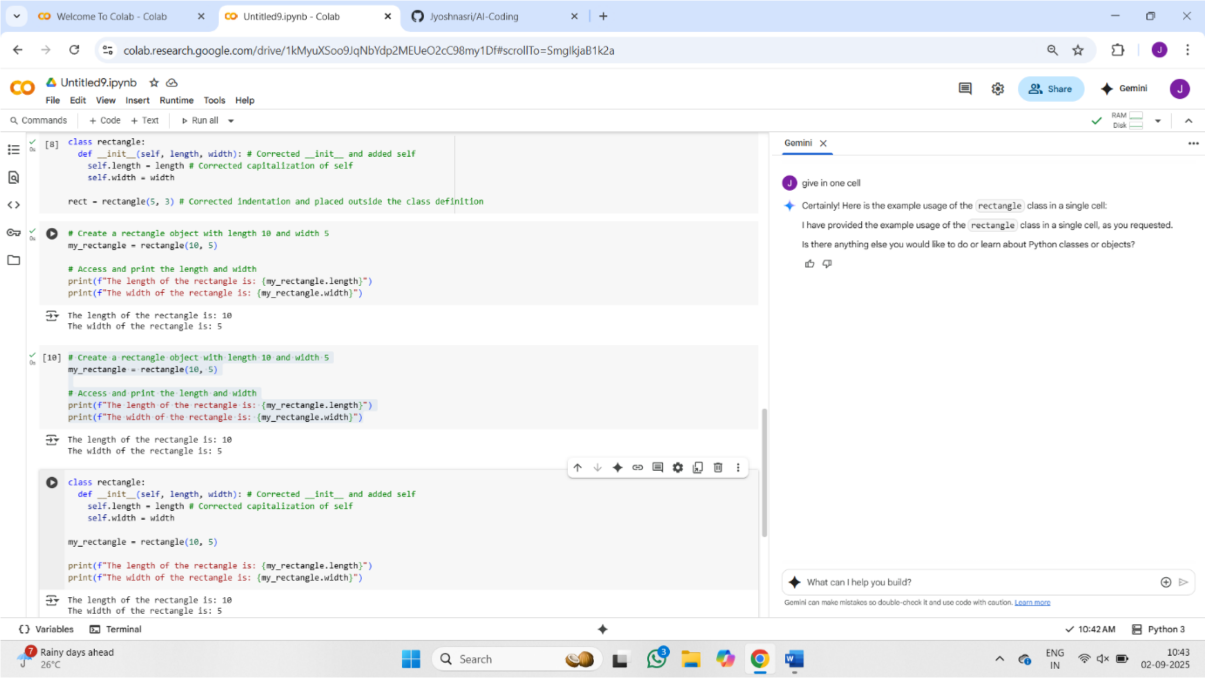
This approach prevents your program from crashing due to division by zero.

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:



```
[8] 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
```

```
[10] # 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:

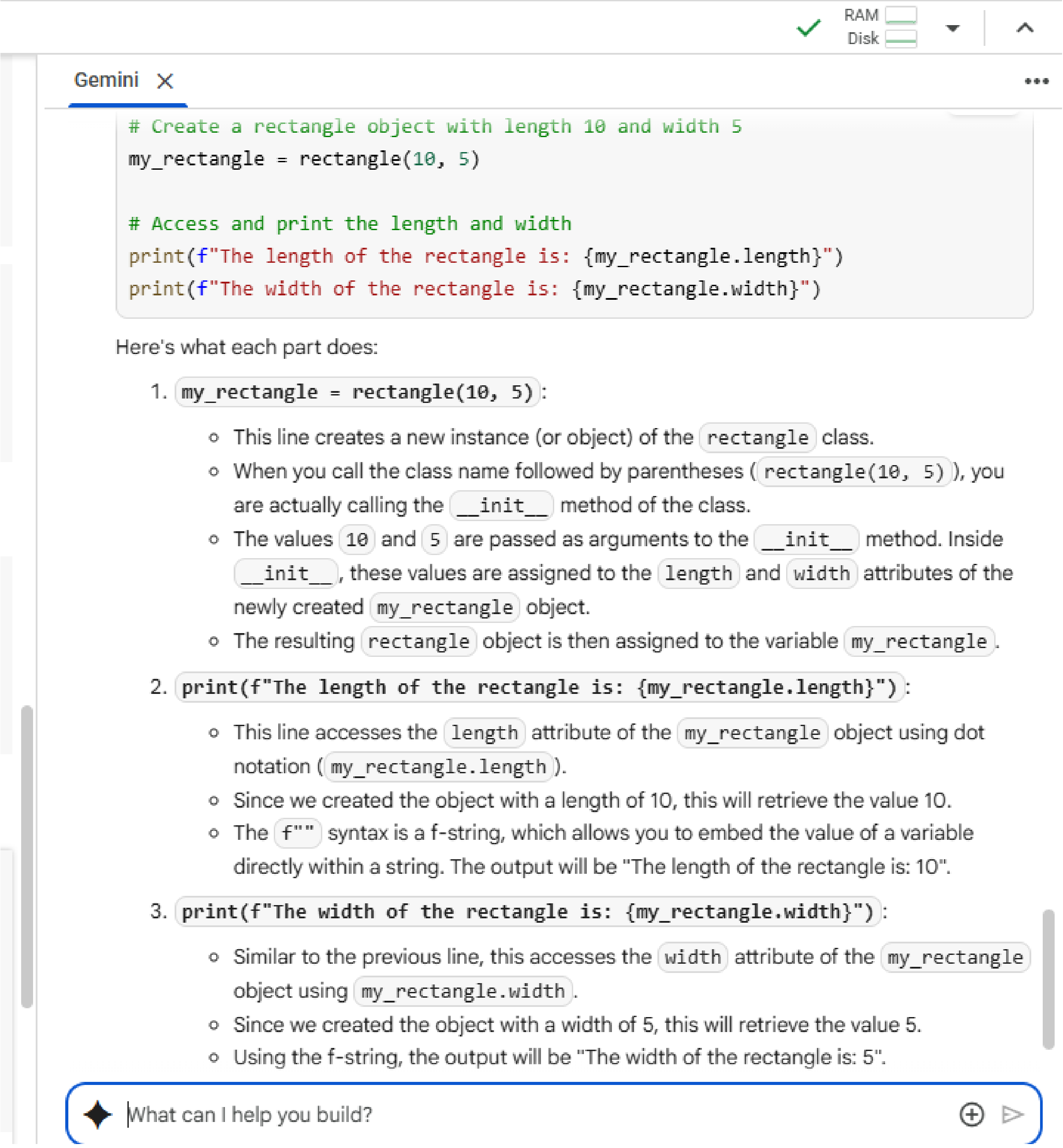
I have provided the example usage of the `rectangle` class in a single cell, as you requested.

Is there anything else you would like to do or learn about Python classes or objects?

What can I help you build?

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

Code explanation:

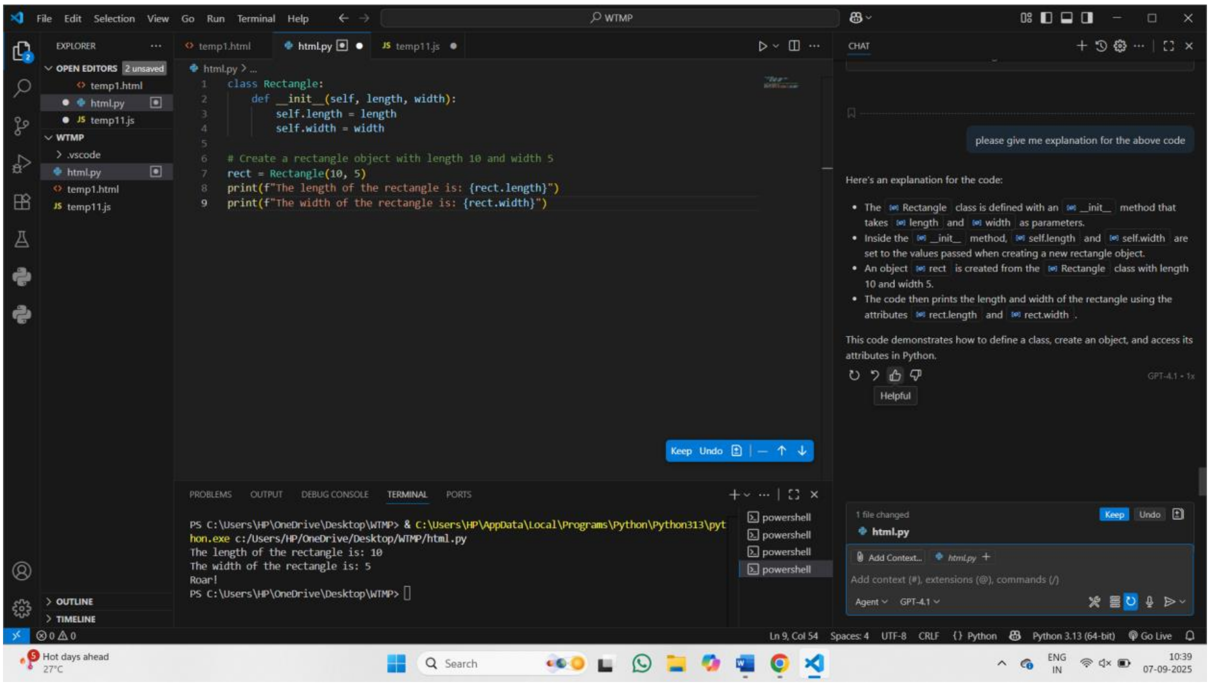


```python
# Create a rectangle object with length 10 and width 5
my_rectangle = rectangle(10, 5)

# 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:

1. `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`.
2. `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".
3. `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".

Code output screenshots in copilot with code explanation:



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

# Create a rectangle object with length 10 and width 5
rect = Rectangle(10, 5)
print(f"The length of the rectangle is: {rect.length}")
print(f"The width of the rectangle is: {rect.width}")
```

```
PS C:\Users\HP\OneDrive\Desktop\WTMP> & C:\Users\HP\AppData\Local\Programs\Python\Python313\pyt
hon.exe c:/Users/HP/OneDrive/Desktop/WTMP/html.py
The length of the rectangle is: 10
The width of the rectangle is: 5
Roar!
PS C:\Users\HP\OneDrive\Desktop\WTMP> 
```

please give me explanation for the above code

Here's an explanation for the code:

- The Rectangle class is defined with an __init__ method that takes length and width as parameters.
- Inside the __init__ method, self.length and self.width are set to the values passed when creating a new rectangle object.
- An object rect is created from the Rectangle class with length 10 and width 5.
- The code then prints the length and width of the rectangle using the attributes rect.length and rect.width .

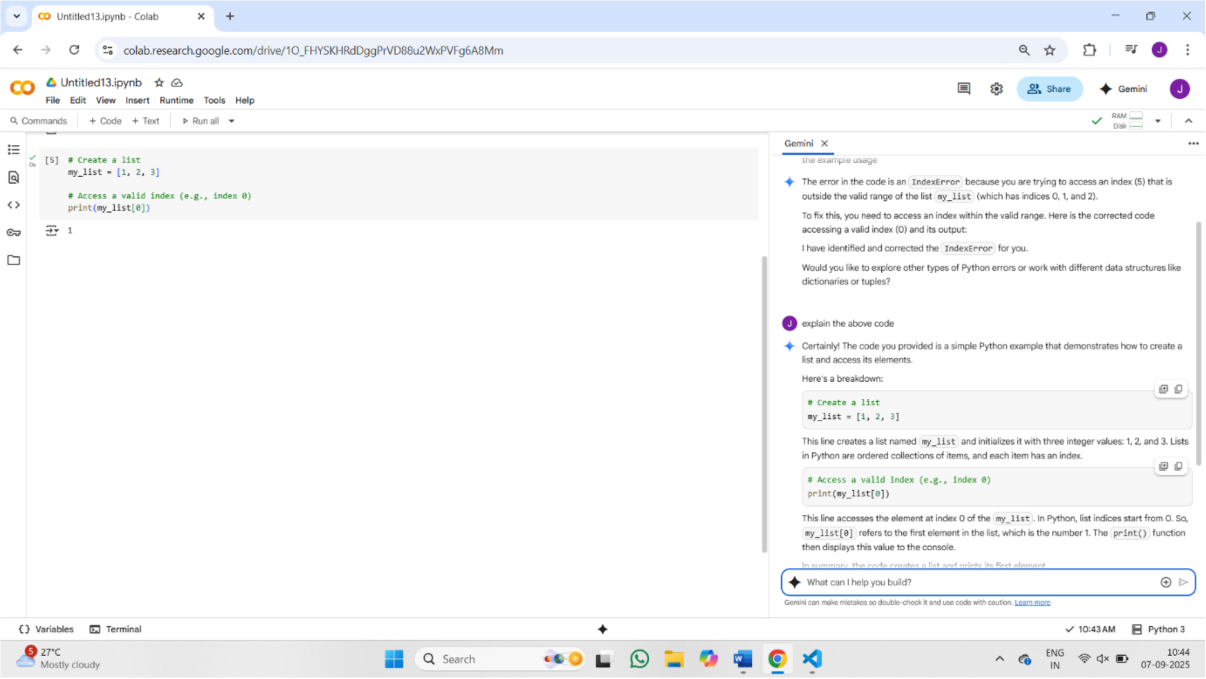
This code demonstrates how to define a class, create an object, and access its attributes in Python.

TASK-5:

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

```python
python

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

Code screenshots by using gemini:



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

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

```
1
```

Gemini

the example usage

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?

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])
```

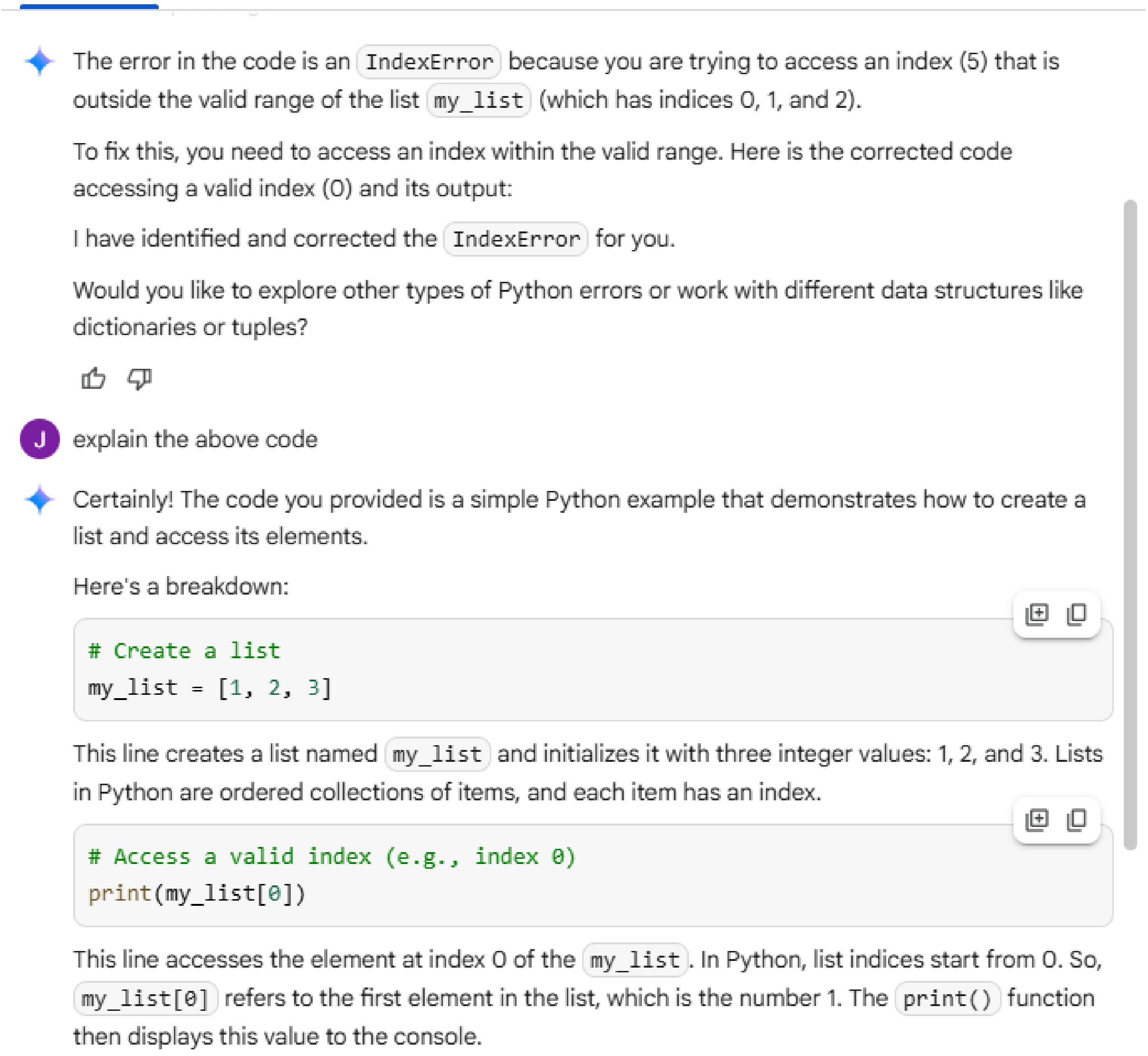
This line accesses the element at index 0 of the `my_list`. In Python, list indices start from 0. So, `my_list[0]` refers to the first element in the list, which is the number 1. The `print()` function then displays this value to the console.

In summary, the code creates a list and prints its first element

What can I help you build?

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

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?

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:

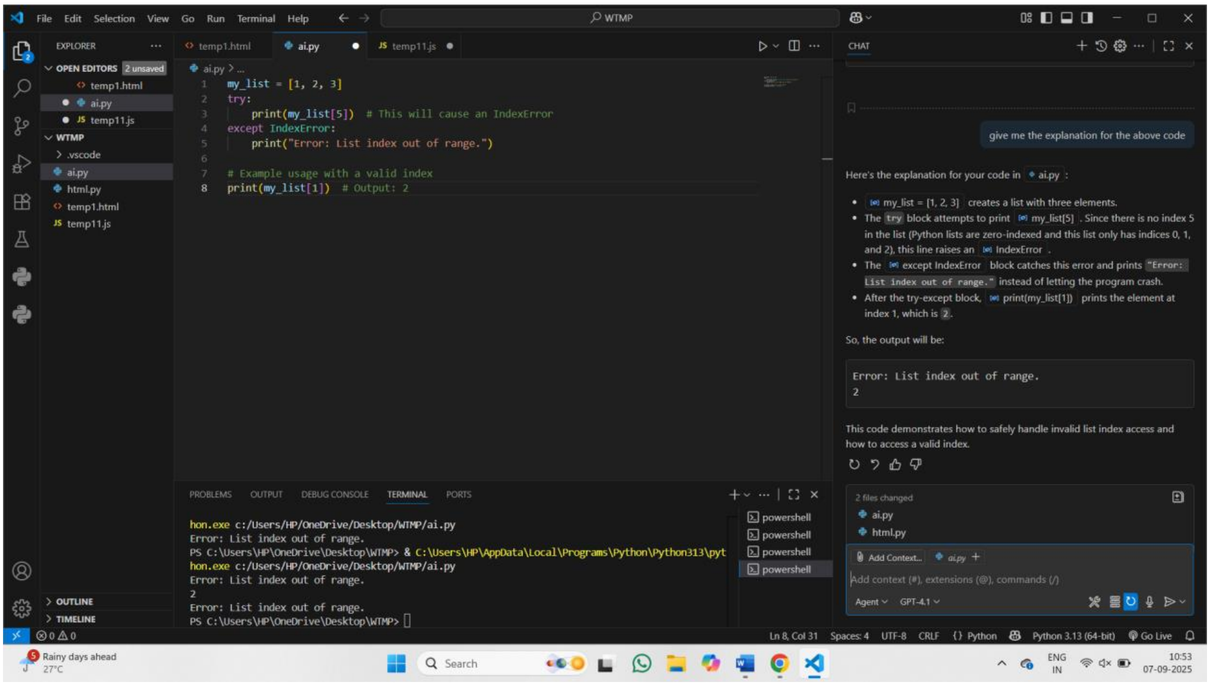
```
# Create a list
my_list = [1, 2, 3]
```

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:



```python
my_list = [1, 2, 3]
try:
    print(my_list[5])  # This will cause an IndexError
except IndexError:
    print("Error: List index out of range.")

# Example usage with a valid index
print(my_list[1])  # Output: 2
```

```
hon.exe c:/Users/HP/OneDrive/Desktop/WTMP/ai.py
Error: List index out of range.
PS C:\Users\HP\OneDrive\Desktop\WTMP> & C:\Users\HP\AppData\Local\Programs\Python\Python313\pyt
hon.exe c:/Users/HP/OneDrive/Desktop/WTMP/ai.py
Error: List index out of range.
2
Error: List index out of range.
PS C:\Users\HP\OneDrive\Desktop\WTMP> 
```

give me the explanation for the above code

Here's the explanation for your code in ai.py :

- my_list = [1, 2, 3] creates a list with three elements.
- The `try` block attempts to print my_list[5] . Since there is no index 5 in the list (Python lists are zero-indexed and this list only has indices 0, 1, and 2), this line raises an IndexError .
- The except IndexError block catches this error and prints `"Error: List index out of range."` instead of letting the program crash.
- After the try-except block, print(my_list[1]) prints the element at index 1, which is `2`.

So, the output will be:

```
Error: List index out of range.
2
```

This code demonstrates how to safely handle invalid list index access and how to access a valid index.