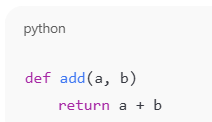
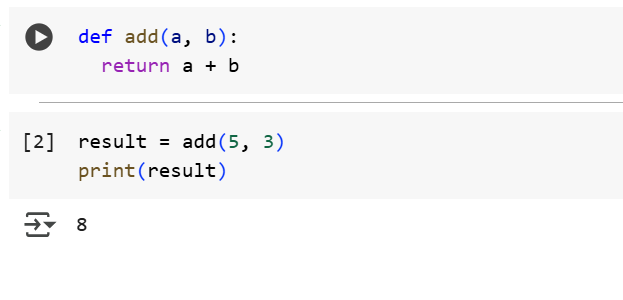
Assignment-7

Task-1

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



CODE:





Explanation:

**def**

* This keyword **defines a function**.
* It tells Python: “Hey, I’m about to declare a reusable block of code.”

🔹 add

* The **name** of the function.
* You can call this function later using add(2, 3).

🔹 **(a, b)**

* These are **parameters**—placeholders for the values you’ll pass in.
* When you call add(5, 7), a = 5 and b = 7.

**:**

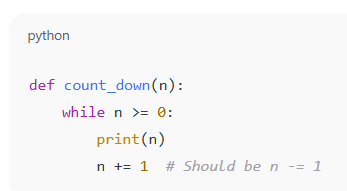
* The colon signals the start of the **function body**.
* Everything indented below it belongs to the function.

🔹 **return a + b**

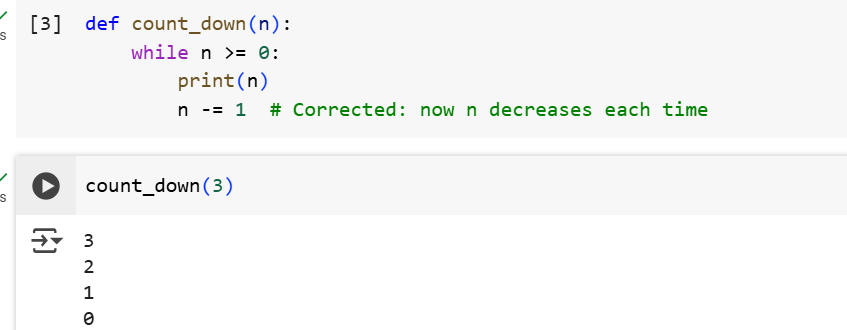
* This line **executes** when the function is called.
* It calculates the sum of a and b, then **returns** the result to the caller.

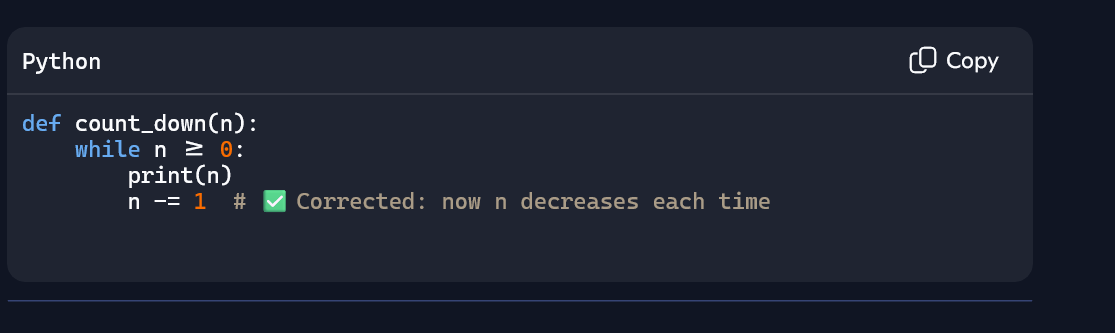
Task-2

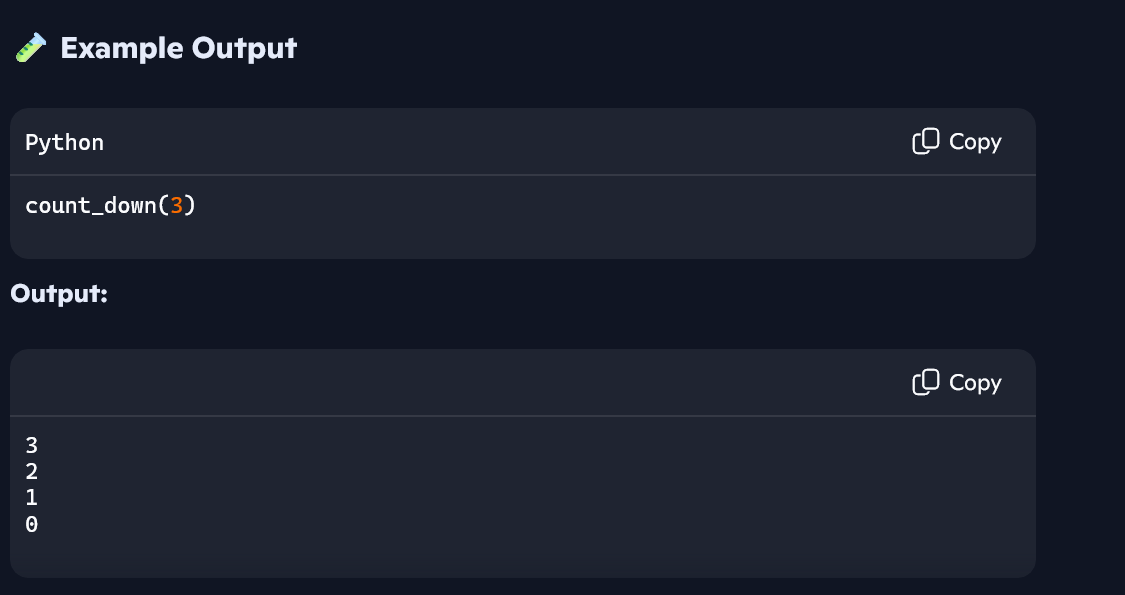
Identify and fix a logic error in a loop that causes infinite iteration.



CODE:







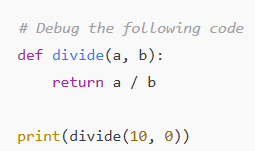
Explanation:

* **def add(a, b):**: This defines a function named add that takes two arguments, a and b. The def keyword is used to define a function, add is the function name, (a, b) are the parameters, and the colon : signifies the start of the function's body.
* **return a + b**: This is the body of the add function. It calculates the sum of a and b and uses the return keyword to send that value back when the function is called.
* **result = add(5, 3)**: This line calls the add function with the values 5 and 3 for a and b respectively. The value returned by the function (which is 8) is then stored in the variable result.
* **print(result)**: This line prints the value stored in the result variable to the console.
* **def count\_down(n):**: This defines a function named count\_down that takes one argument, n.
* **while n >= 0:**: This is a while loop. The code inside the loop will continue to execute as long as the condition n >= 0 is true.
* **print(n)**: This line prints the current value of n inside the loop.
* **n -= 1**: This line decrements the value of n by 1 in each iteration of the loop. This is the crucial part that ensures the loop eventually terminates when n becomes less than 0.
* **count\_down(3)**: This line calls the count\_down function with the value 3 for n. This starts the countdown from 3.

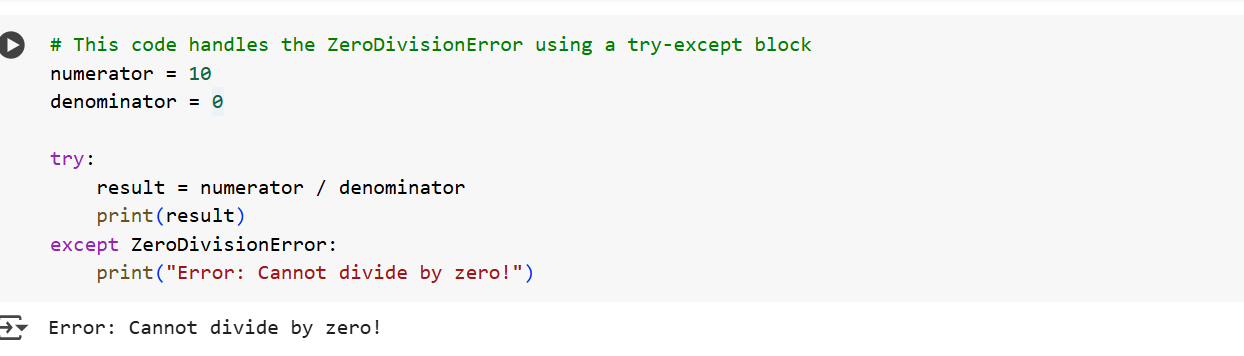
In summary, the first set of cells defines and uses a simple function to add two numbers, while the second set defines and uses a function to count down from a given number to zero using a while loop.

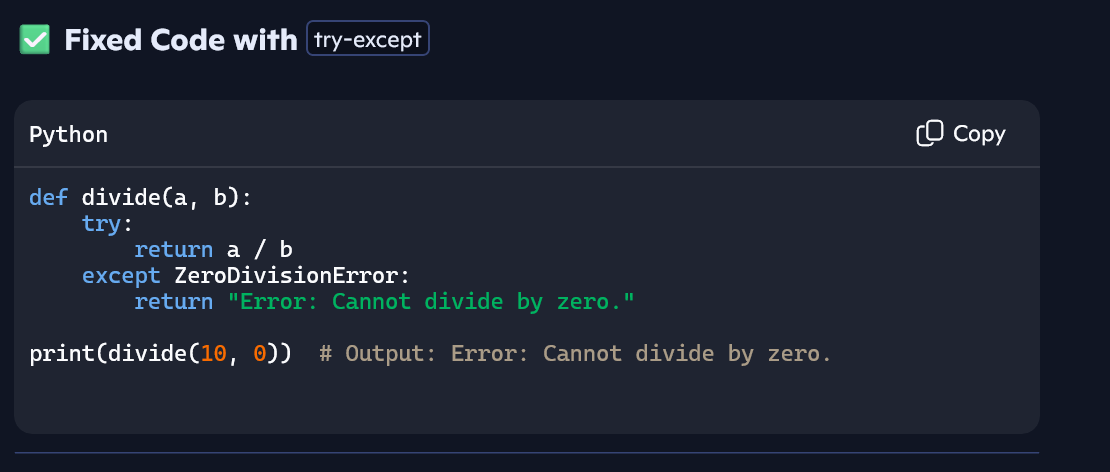
Task-3

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



CODE:

****

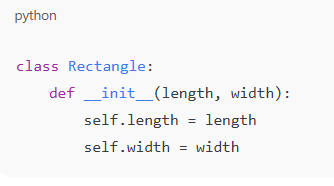
****

Explanation:

1. **try:**: This block contains the code that might potentially raise an error. In this case, the division numerator / denominator is placed inside the try block because it can cause a ZeroDivisionError.
2. **except ZeroDivisionError:**: This block is executed only if a ZeroDivisionError occurs within the try block.
3. **print("Error: Cannot divide by zero!")**: If a ZeroDivisionError is caught, this line is executed, printing an informative message instead of crashing the program.

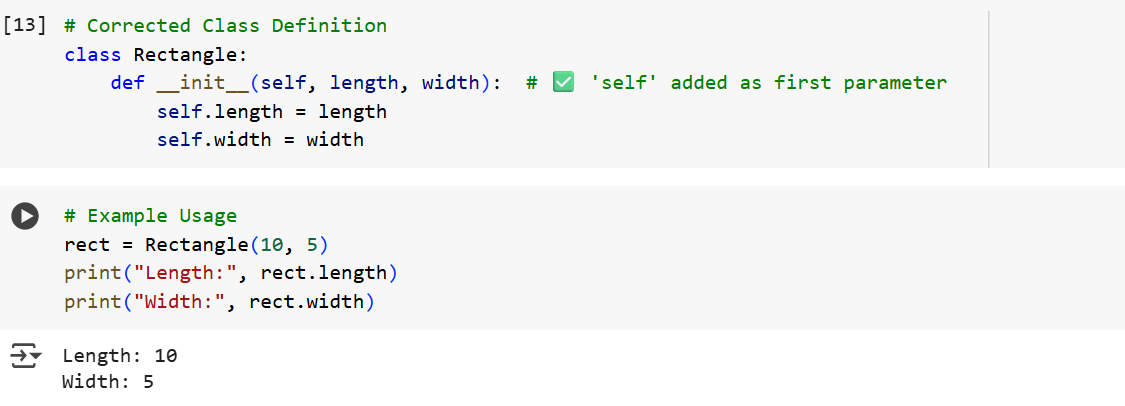
Task-4

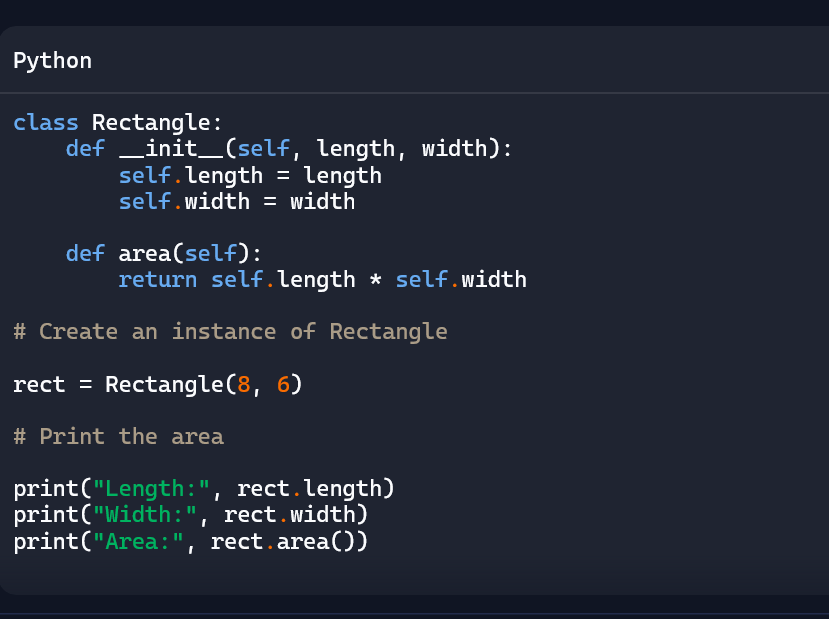
Provide a faulty class definition (missing self in parameters). Let AI fix it

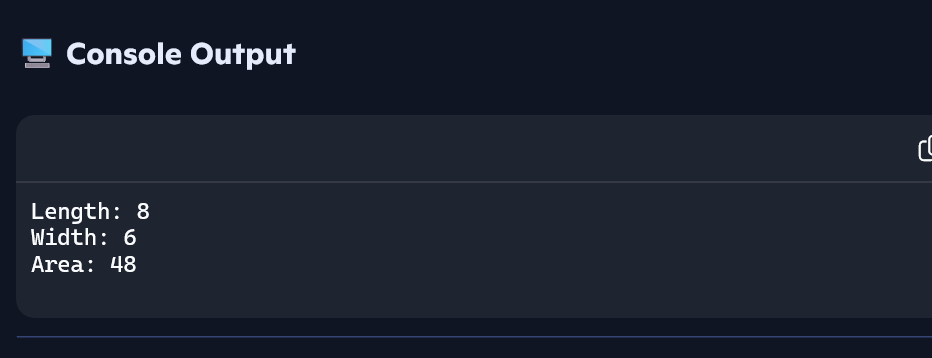


Correct \_\_init\_\_() method and explanation.

CODE:

****

****

****

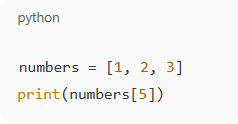
Explanation:

* **\_\_init\_\_(self, ...)**: This is a special method in Python classes called the constructor. It's automatically called when you create a new object (an instance) of the class. The purpose of \_\_init\_\_ is to initialize the attributes (like length and width in this case) of the object. The double underscores at the beginning and end indicate that it's a special method.
* **self**: This is a conventional name (though you could technically use another name, it's strongly recommended to use self) that refers to the instance of the class itself. When you call a method on an object (like my\_rectangle.area()), the object (my\_rectangle) is automatically passed as the first argument to the method, and this argument is typically named self in the method definition. Inside the \_\_init\_\_ method, self refers to the newly created object that is being initialized.
* **self.length = length**: This line takes the value passed for the length parameter when creating the object and assigns it to an attribute of the object itself, named length. The self. is necessary to distinguish the instance's attribute (self.length) from the local parameter (length).
* **self.width = width**: Similarly, this line assigns the value passed for the width parameter to the width attribute of the object.

In essence, \_\_init\_\_ sets up the initial state of an object when it's created, and self is the way methods within the class refer to the specific object they are operating on.

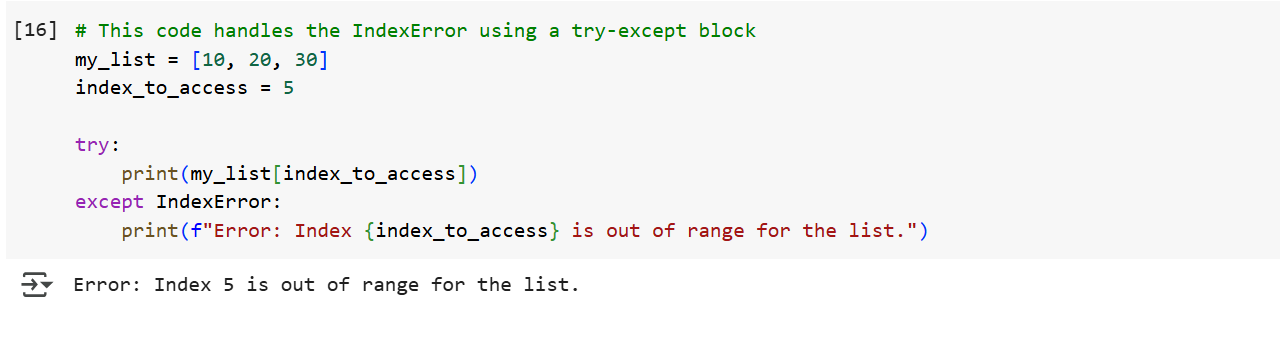
Task-5

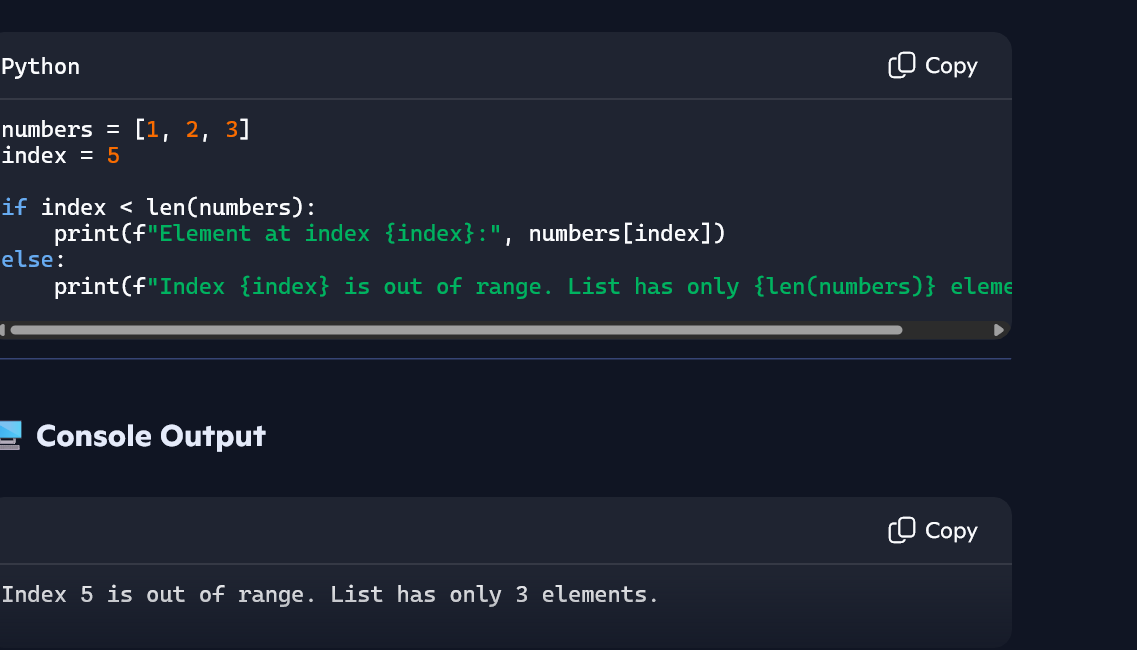
Access an invalid list index and use AI to resolve the Index Error.



AI suggests checking length or using safe access logic

CODE:





Explanation:

1. **my\_list = [10, 20, 30]**: This creates a list with three elements at indices 0, 1, and 2.
2. **index\_to\_access = 5**: This variable stores the index we want to access, which is intentionally out of the list's range.
3. **try:**: This block contains the code that might raise an IndexError.
4. **print(my\_list[index\_to\_access])**: This line attempts to access the element at index\_to\_access. Since the index is 5 and the list only has indices 0, 1, and 2, this will raise an IndexError.
5. **except IndexError:**: This block is executed if an IndexError occurs within the try block.
6. **print(f"Error: Index {index\_to\_access} is out of range for the list.")**: If an IndexError is caught, this informative message is printed, indicating that the requested index is outside the bounds of the list.