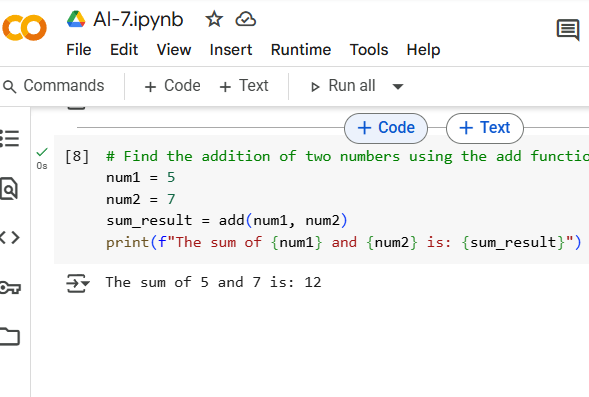
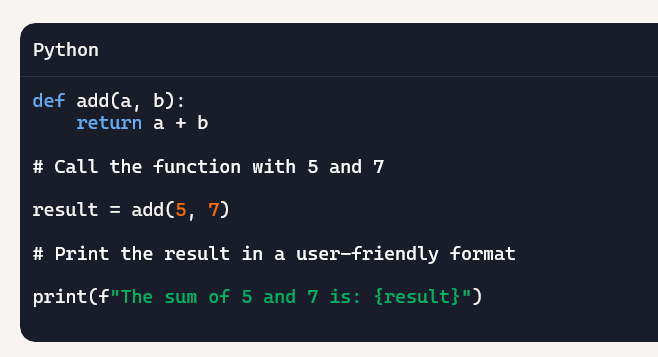
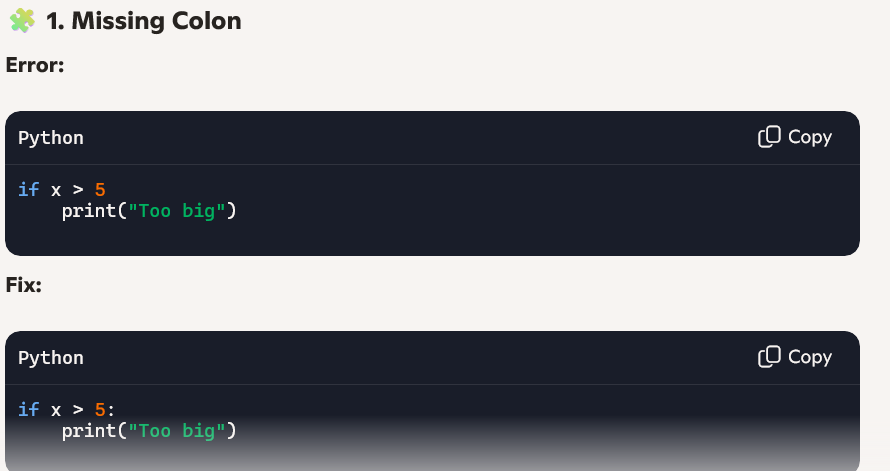
Assingme-7

Task-1: Using the add function, find the sum of the numbers 5 and 7 and print the result in a user-friendly format.







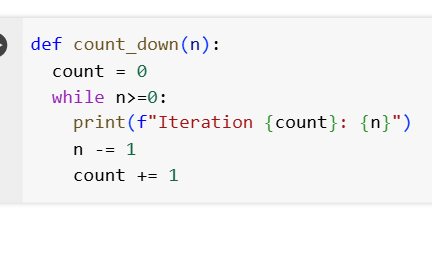
Explaination:

1. It assigns the value 5 to the variable num1.
2. It assigns the value 7 to the variable num2.
3. It calls the add function (which we defined earlier) with num1 and num2 as arguments. The result of this addition (which is 12) is stored in the variable sum\_result.
4. Finally, it prints a formatted string that includes the values of num1, num2, and sum\_result to show the sum of the two numbers.

Task-2:

Prints numbers from n down to 0 and tracks how many times the loop runs





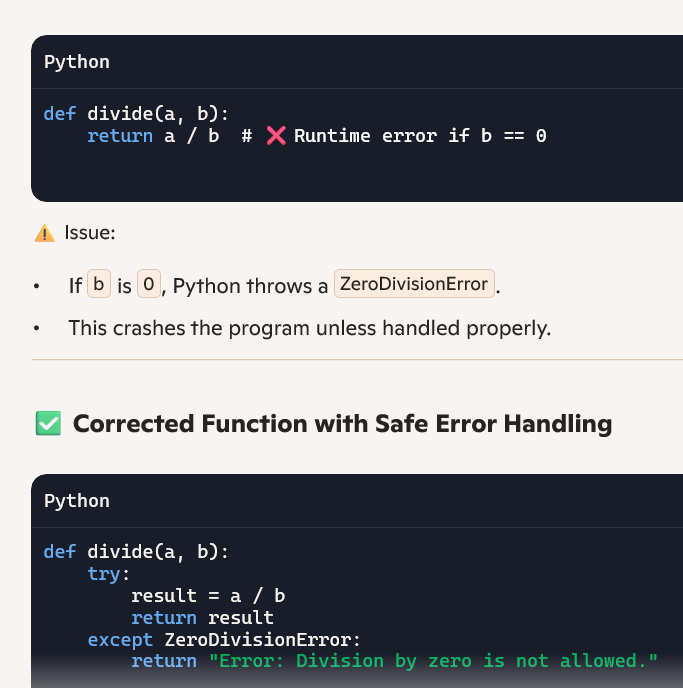
**Explaination:**

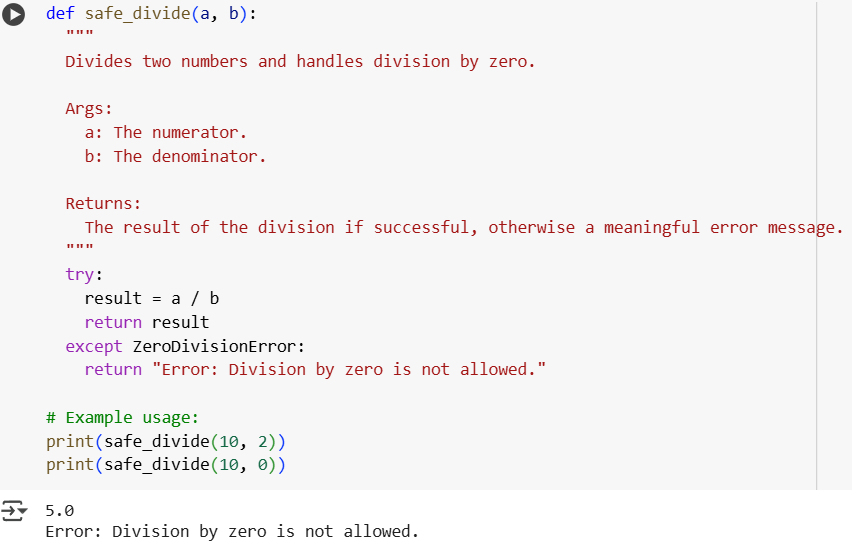
1. **def count\_down(n):**: This line defines a function named count\_down that accepts one input parameter, n.
2. **count = 0**: Inside the function, a variable count is initialized to 0. This variable will be used to keep track of the number of iterations the loop performs.
3. **while n>=0:**: This is a while loop that continues to execute as long as the value of n is greater than or equal to 0.
4. **print(f"Iteration {count}: {n}")**: Inside the loop, this line prints the current iteration number (count) and the current value of n. The f"" is a formatted string literal that allows you to embed the values of variables directly within the string.
5. **n -= 1**: This line decrements the value of n by 1 in each iteration. This is crucial for the loop to eventually terminate when n becomes less than 0.
6. **count += 1**: This line increments the count variable by 1 in each iteration, keeping track of how many times the loop has run.

In summary, the count\_down function will print a countdown from the initial value of n down to 0, showing the iteration number for each step.

Task-3:

Write a Python function that divides two numbers. Then, debug a runtime error caused by division by zero by inserting appropriate try-except handling. Return a meaningful message if an error occurs."





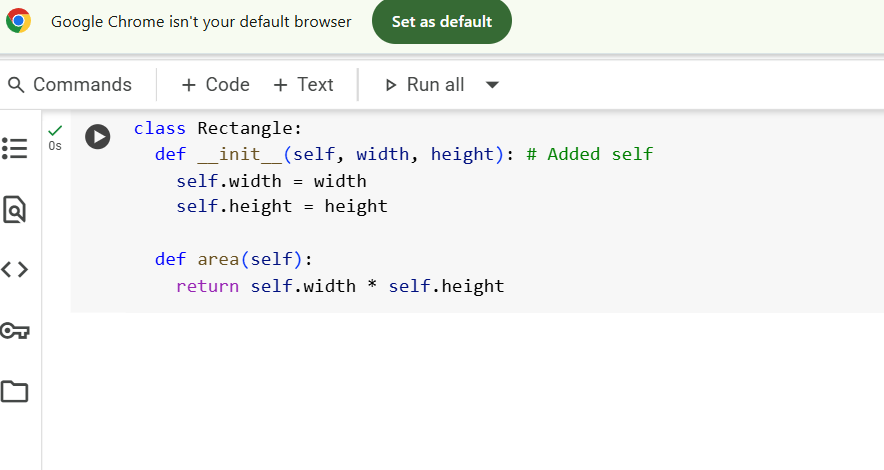
Explanation:

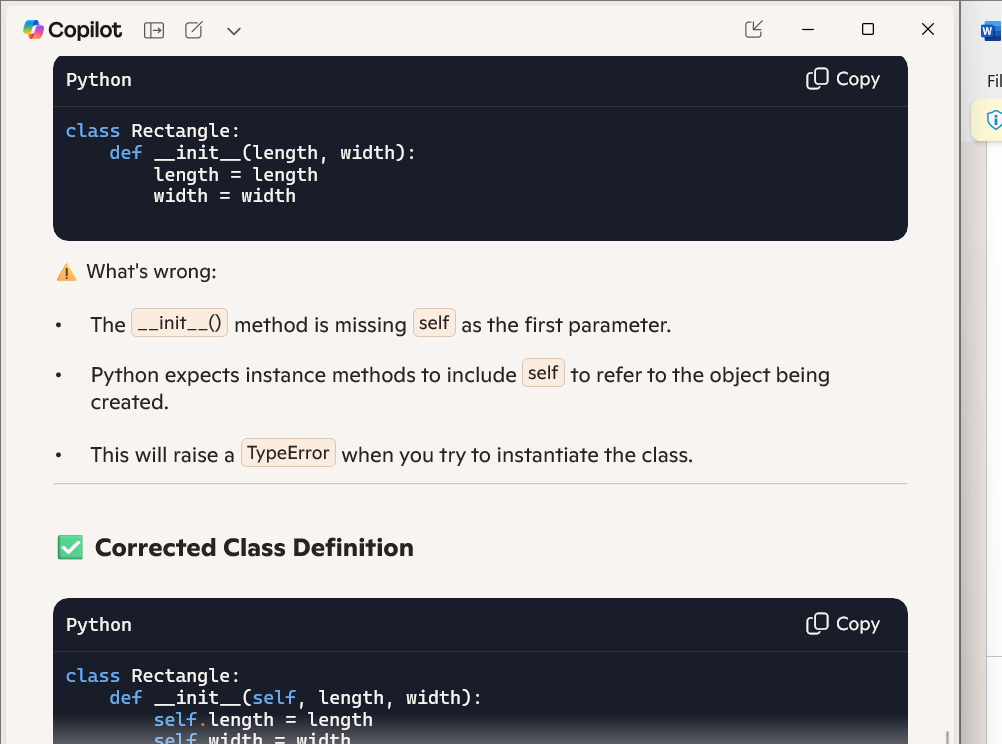
* **def safe\_divide(a, b):**: This line defines the function named safe\_divide that takes two arguments, a (the numerator) and b (the denominator).
* **try:**: This block of code attempts to execute the code within it. If an error occurs during the execution of the code in the try block, the program will jump to the corresponding except block.
* **result = a / b**: This line performs the division of a by b and stores the result in the result variable.
* **return result**: If the division is successful (i.e., no error occurs), the function returns the calculated result.
* **except ZeroDivisionError:**: This block of code is executed only if a ZeroDivisionError occurs in the try block. This specific error happens when you try to divide a number by zero.
* **return "Error: Division by zero is not allowed."**: If a ZeroDivisionError is caught, the function returns a meaningful error message instead of crashing the program.

The example usage at the end demonstrates how to call the function with valid inputs (10, 2) and an input that would cause a division by zero (10, 0), showing how the function handles both cases.

Task-4:

create a Python class called Rectangle with an \_\_init\_\_() method that takes length and width as parameters. Introduce a bug by omitting self from the method definition. Then, correct the method and explain why self is required in instance methods**."**

****

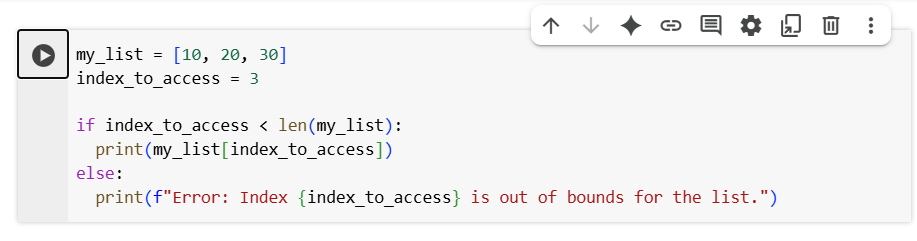
.**Explaination:**

What This Code Does

* class Rectangle:  
  This defines a new class named Rectangle. A class is a blueprint for creating objects that share common properties and behaviors.
* def \_\_init\_\_(self, length, width):  
  This is the constructor method. It runs automatically when a new Rectangle object is created.
  + self refers to the instance of the class being created.
  + length and width are parameters passed during object creation.
* self.length = length and self.width = width  
  These lines assign the values of length and width to the instance variables self.length and self.width.  
  This means each Rectangle object will store its own length and width.

Task-5:

Prompt: Access and print the last element of a list safely, regardless of its length.





**Explaination:**

* if index\_to\_access < len(my\_list):: This is the crucial part for error prevention.
  + len(my\_list) returns the number of elements in the list, which is 3 in this case.
  + The condition checks if the index\_to\_access (which is 3) is less than the length of the list (which is 3).
* print(my\_list[index\_to\_access]): This line is inside the if block and will only be executed if the condition index\_to\_access < len(my\_list) is True. If the index is valid, it prints the element at that index.
* else:: This block is executed if the condition in the if statement is False, meaning the index\_to\_access is not less than the length of the list (i.e., it's an invalid index).
* print(f"Error: Index {index\_to\_access} is out of bounds for the list."): This line is inside the else block and prints an informative error message indicating that the requested index is outside the valid range for the list.