Click to verify

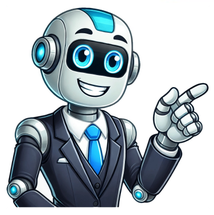

Illustration (and all in this article) by Adit BhargavaIn order to understand recursion, one must first understand recursion.Recursion can be tough to understand especially for new programmers. In its simplest form, a recursive function is one that calls itself. Let me try to explain with an example.Imagine you go to open your bedroom door and its locked. Your three-year-old son pops in from around the corner and lets you know he hid the only key in a box. (Just like him, you think.) You're late for work and you really need to get in the room to get your shirt.You open the box only to find more boxes. Boxes inside of boxes. And you dont know which one has the key! You need to get that shirt soon, so you have to think of a good algorithm to find that key.There are two main approaches to create an algorithm for this problem: iterative and recursive. Here are both approaches as flow charts:Which approach seems easier to you?The first approach uses a while loop. While the pile isnt empty, grab a box and look through it. Heres some JavaScript-inspired pseudocode that shows what is happening. (Pseudocode is written like code, but meant to be more like human speech.)function look_for_key(main_box) { let pile = main_box.make_a_pile_to_look_through(); while (pile is not empty) { box = pile.grab_a_box(); for (item in box) { if (item.is_a_box()) { pile.append(item) } else if (item.is_a_key()) { console.log("found the key!") } } }}The second way uses recursion. Remember, recursion is where a function calls itself. Heres the second way in pseudocode.function look_for_key(box) { for (item in box) { if (item.is_a_box()) { look_for_key(item); } else if (item.is_a_key()) { console.log("found the key!") } }}Both approaches accomplish the same thing. The main purpose for using the recursive approach is that once you understand it, it can be clearer to read. There is actually no performance benefit to using recursion. The iterative approach with loops can sometimes be faster. But mainly the simplicity of recursion is sometimes preferred.Also, since a lot of algorithms use recursion, its important to understand how it works. If recursion still doesnt seem simple to you, dont worry: Im going to go over a few more examples. Something you have to look out for when writing a recursive function is an infinite loop. This is when the function keeps calling itself and never stops calling itself!For instance, you may want to write a count down function. You could write it recursively in JavaScript like this:function countdown(i) { console.log(i) countdown(i - 1)} countdown(5); This function will keep counting down forever. If you do accidentally run code with an infinite loop you can press Ctrl-C to kill your script. (Or, if you sometimes use CodePen like me, you have to add ?turn_off_js=true to the end of the URL.)A recursive function always has to say when to stop repeating itself. There should always be two parts to a recursive function: the recursive case and the base case. The recursive case is when the function calls itself. The base case is when the function stops calling itself. This prevents infinite loops.Here is the countdown function again, with a base case:function countdown(i) { console.log(i) if (i = 0) is the product of all positive integers from 1 to n. To compute the factorial recursively, we calculate the factorial of n by using the factorial of (n-1). The base case for the recursive function is when n = 0, in which case we return 1. C++ #include using namespace std; int fact(int n){ // BASE CONDITION if (n == 0) return 1; return n * fact(n - 1);} int main(){ cout