Reading Recursive Code

coursera.org/learn/pointers-arrays-recursion/supplement/ITtDs/reading-recursive-code

As we typically do, we will start by learning to read before we learn to write. However, unlike most things we will encounter, there are not actually any new semantics to learn—we just follow the same rules that we have already learned. However, we will still remind you of the rules for executing function calls, and underscore how they are *exactly* the same for a recursive function.

We will start by working with the function as an example. Recall that in math, the factorial of a number (written *n!* in math notation) is:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

One thing you may notice here is that the mathematical definition of the function is recursive—the general case of *n>0* is expressed in terms of the factorial of a smaller number ((*n-1*)*!*). Since the problem itself is recursive, a recursive algorithm is a natural choice. We will see how we go about devising and implementing an algorithm later, but for now, we will just work from the following code which computes factorial:

```
3
4
5
6
1
2
    return 1;
  }
  return n * factorial(n-1);
}
int factorial(int n) {
  if (n <= 0) {
```

If you take a second to examine this code, you will see that everything in it is something we have seen. Line 1 tells us that this function is called factorial and takes an **int** (*n*), and returns an **int**. Line 2 has an **if** statement with a conditional expression (*n <= 0*) which should be imminently familiar by now. Line 3 contains the "then" clause of the **if** statement, which just returns 1. Line 5 returns an expression which is the result of multiplying n by the result of calling the factorial function with argument *n-1*.

Whenever our execution arrow reaches Line 5, we do everything exactly as we have previously learned. We multiply *n* (which we would read out of its box) by the value returned by *factorial*(*n-1*). To compute *factorial(n-1)*—which is itself a function call—we would need to draw a stack frame for factorial (with a box for the parameter *n*, compute *n-1* and copy it into the box for the parameter, note down where to return to, and move our execution arrow into the start of the factorial function.

The next video will step through execution of factorial to show you how execution of recursive code follows the now-familiar rules of calling and returning from functions.