Recursive Functions

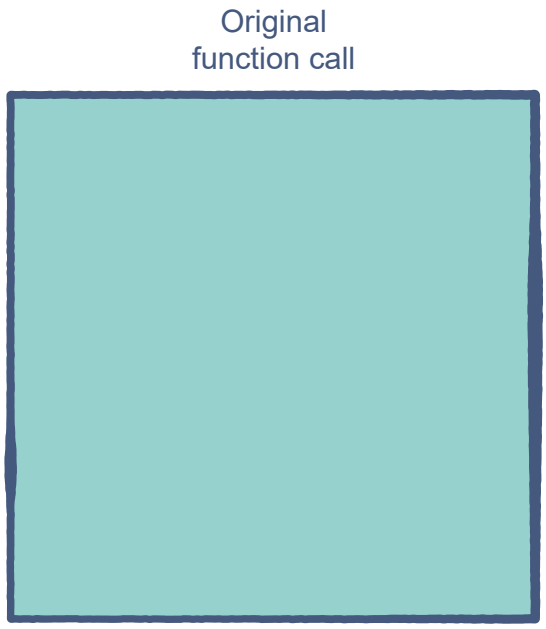
This lesson will teach us how to implement recursion in ReasonML.

WE'LL COVER THE FOLLOWING

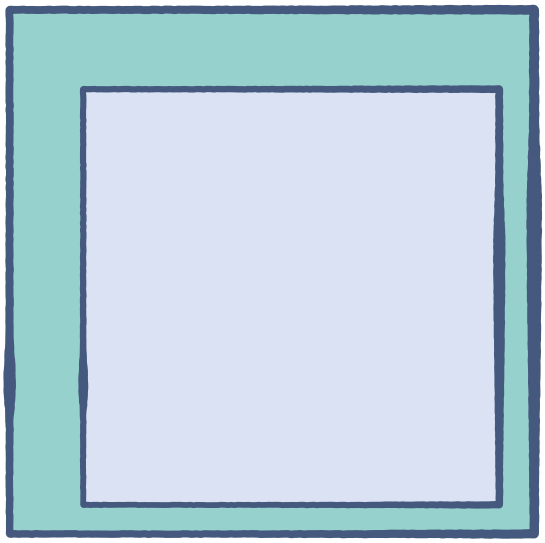
Definition

Recursion is the process in which a function calls itself during runtime. It stops at the **base case**. The base case is a check which specifies the final level of recursion.

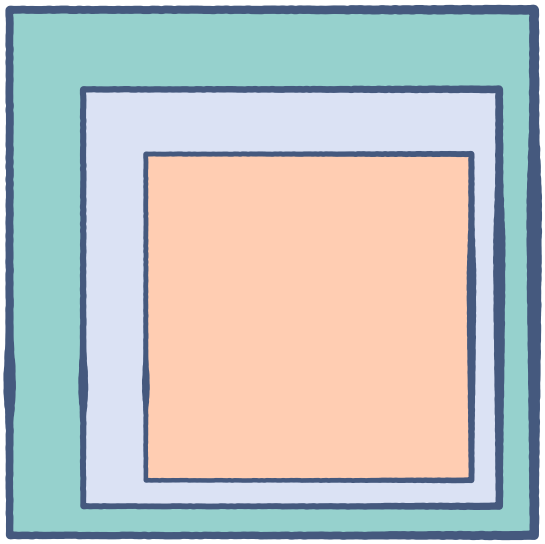
We can think of recursion as boxes within boxes, with each recursive call being a box. Recursion grows in a nested manner, and always reduces back to the original call.



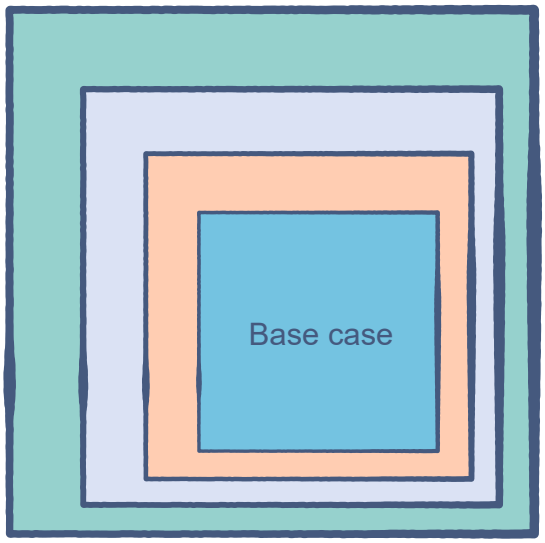
Recursive call within the function



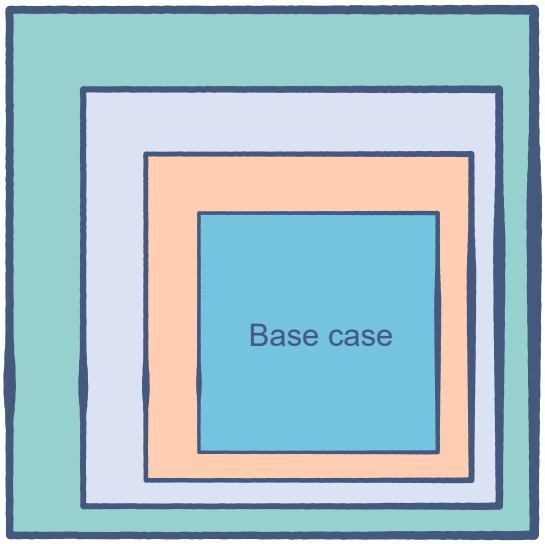
Each call creates a new box

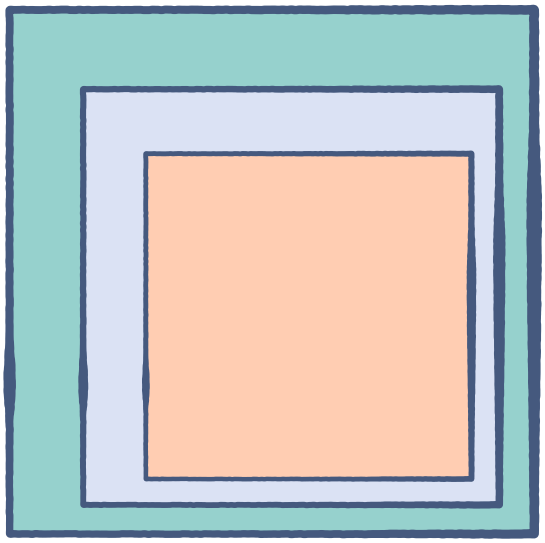


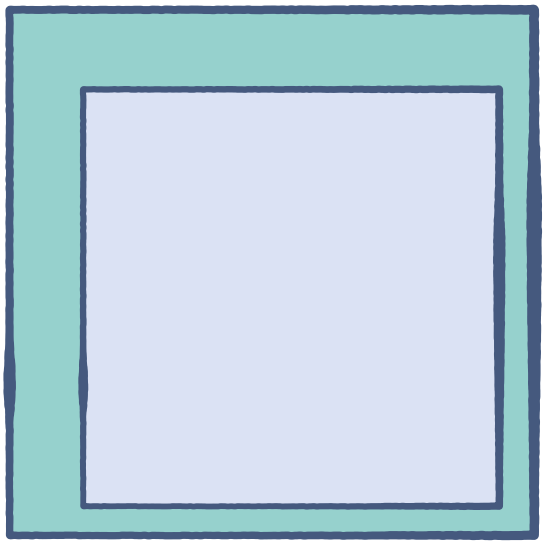
Base case has been reached

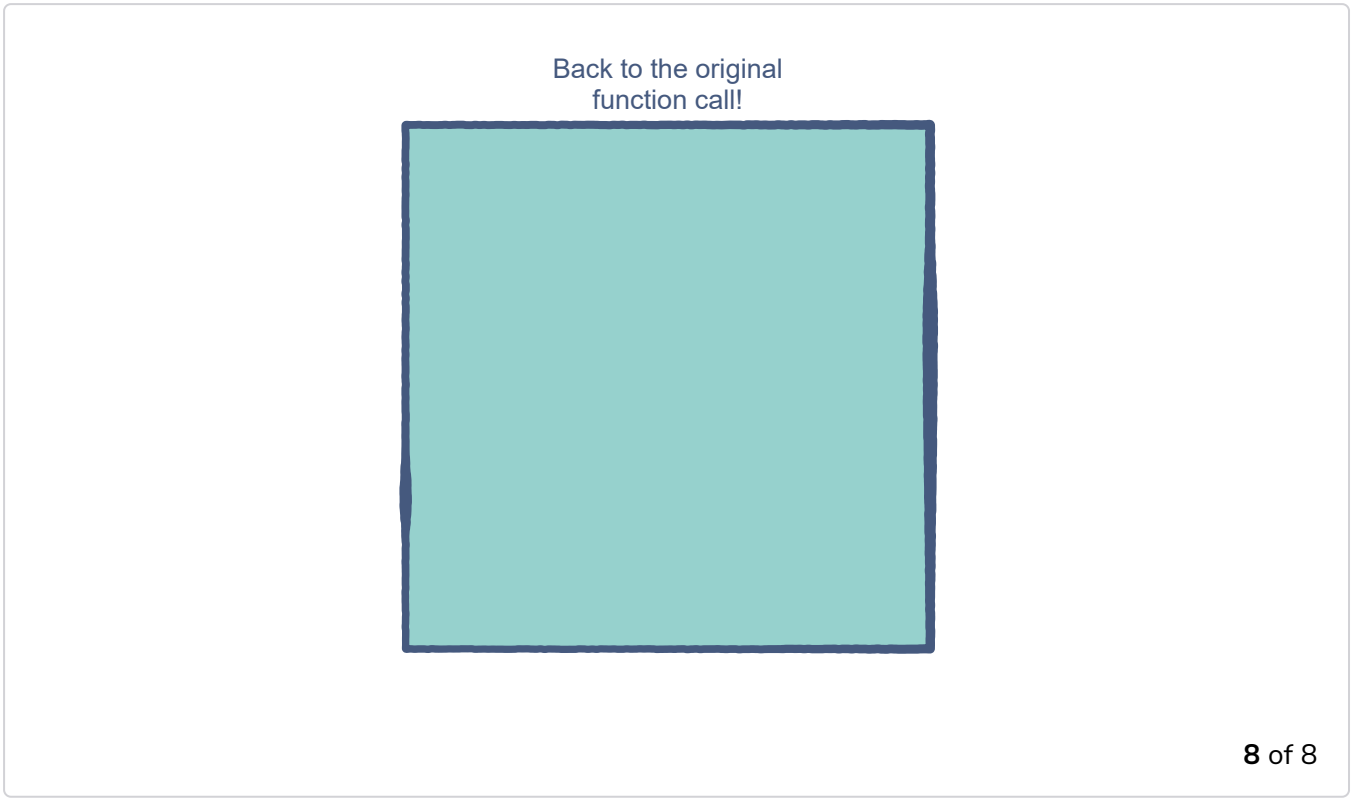


Each call ends one by one









8 of 8

For more on the theory of recursion, check out the Wikipedia page.

Recursion is an integral part of functional programming as it increases the computing power of functions, albeit, at the cost of memory.

Like JavaScript, Reason also supports recursion. However, the syntax is slightly different.

Let's take the example of an algorithm which computes the *n*-th Fibonacci number. In the Fibonacci sequence, each number is a sum of the previous two numbers in the sequence. The first and second values are `0` and `1`. Here are the first 7 numbers in the Fibonacci sequence:

```
0, 1, 2, 3, 5, 8, 13...
```

The pseudo-code would look something like this:

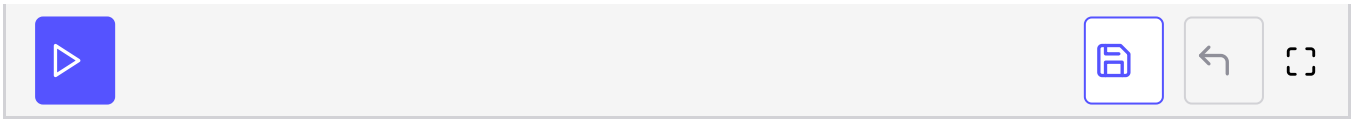
```
fib (n):
    // Base case
    if(n <= 1):
        return n

    // Recursive call
```

```
    fib(n-1) + fib(n-2)
```

In Reason syntax, we'd get the following function:

```
let fib = (n) => {
  if (n <= 1) {
    n;
  }
  else{
    fib(n-1) + fib(n-2);
  }
}
```



This code blows up!

Why? Well, we are telling the compiler to make calls to `fib(n-1)` and `fib(n-2)`, but it does not know that these are supposed to be recursive calls. It will look in the code for an implementation for these calls, and since they won't be found, the compiler will throw an error.

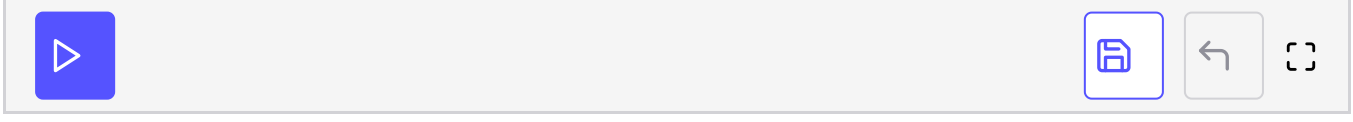
The `rec` Keyword

The `rec` keyword can be used to tell the compiler that the particular function will be used recursively. It is placed right after the `let` keyword.

Below, we can find the correct syntax for our `fib()` function:

```
let rec fib = (n) => {
  if (n <= 1) {
    n;
  }
  else{
    fib(n-1) + fib(n-2);
  }
}

Js.log(fib(7));
```



Now our recursive function works! So, always remember to use the `rec` keyword when dealing with recursive functions.

In the next lesson, we'll look at a special data type which is relevant to functions.