Recursive Functions

In this lesson, you will be given a brief introduction to recursion and go over how recursion is implemented in Dart.

We'll cover the following

Recursive functions are functions that call themselves in their own function body. This may seem a bit strange right now, but let's see how this works.

Recursion

Recursion is the process of breaking down an expression into smaller and smaller expressions until you're able to use the same algorithm to solve each expression. The smaller expressions are similar versions of the original expression.

One way to implement recursive functions is by using an `if-else` statement. The `if` represents the *base case* which is the smallest possible expression on which an algorithm will run and the `else` represents the *recursive call*; when a function calls itself, it is known as a **recursive call**. The recursive function will keep calling itself in a **nested** manner without terminating the call until it is equivalent to the base case in which case the algorithm will be applied. The function calls will move in an outward manner, terminating before moving on to the next one, reducing themselves until they reach the original function call.

Let's look at recursive calls as boxes within boxes.

7 of 8

8 of 8

Factorials

The implementation of factorials is a very famous recursive problem. A factorial of

The implementation of factorials is a very famous recursive problem. A factorial of a number is achieved by multiplying all consecutive numbers starting with **1** until

you reach the number in question. So, the factorial of **4** (represented as **4!**) is equivalent to **1** x **2** x **3** x **4** = **24**. Recursively, it would look like **4!** = **4 x (3!)**.

Before we start coding, let's look at the problem from a different perspective.

One way to think of recursion is the smart-alec way of answering something, answering without really answering. So, you ask someone what the factorial of **5** is, they say, it is **5** times the factorial of **4**. Well, hello! If I knew what the factorial of **4** is, I would probably also know how to find the factorial of **5**. So, the question isn't really answered. You follow up with, well, what is the factorial of **4**, then? You get the expected reply until the conversation comes to what is the factorial of **1**. At that point you hear, yes, that's easy; I know it is **1**. From that point, you start putting all the pieces together and get your answer.

Dart's Implementation

Let's look at how we would implement a factorial function in Dart.

```
int factorial(int x) {
  if (x == 1) { // Base Case
    return 1;
  } else {
    return x*factorial(x-1); // Recursive Call
  }
}

main() {
  // Driver Code
  var result = factorial(5);
  print(result);
}
```

`factorial` is a recursive function that takes one parameter; the number whose factorial is to be calculated. The base case is if the number is **1**, in which case all we have to do is return **1** as that is its own factorial (**lines 2-3**). We cannot break down the expression further than this, where the factorial of a number is the number itself.

However, if the number is not equal to **1**, the `else` expression will be executed in which the return value is the number being multiplied with the factorial of the number before it (**lines 4-5**). This will continue until we pass **1** in our recursive call. Let's look at a graphical implementation of how this works.

1 of 10

2 of 10

5 of 10

6 of 10

7 of 10

8 of 10

9 of 10

10 of 10

Do you see how the result of an inner function call is used to terminate its outer function call?

Recursion is a difficult concept to wrap your head around, but once you do, it is an extremely powerful tool you can use in a functional programming language.

In the next lesson, you will be challenged to write your own recursive function.