Functions Returning Functions

We know that in Scala, functions can return other function. In this lesson we will explore how that is.

We'll cover the following

- A Recap
- Modifying the sum Function
- Understanding the sum Function
- Modifying the Summation Functions

A Recap

Let's look at the summation functions which we created using anonymous functions.

```
def sumOfInts(a: Int, b: Int) = sum(x => x, a, b )

def sumOfCubes(a: Int, b: Int) = sum(x => x*x*x, a, b)

def sumOfFactorials(a: Int, b: Int) = sum(factorial, a, b)
```

Notice how `a` and `b` are being passed from the summation functions to the `sum` function without any modifications, completely unchanged. Because Scala is all about concise code, we will rewrite our summation functions to remove any unnecessary parameters.

First, we will have to rewrite the `sum` function. Just to jog up the memory, here is the current `sum` function.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
def sum(f: Int => Int, a: Int, b: Int): Int ={
  if(a>b) 0
  else f(a) + sum(f, a+1, b)
}
```

```
// Driver Code
def sumOfInts(a: Int, b: Int) = sum(x => x, a, b )



println(sumOfInts(1,5))
```

Modifying the `sum` Function

The modified `sum` function will be written in such a way that it only takes a single parameter, a function `f`. As before, the function parameter `f` takes a parameter of type `Int` and returns a value of type `Int`. However, this time around, `sum` isn't returning an `Int` type value, rather it is *returning a complete function*. Let's look at the code below.

```
def sum(f: Int => Int): (Int,Int) => Int ={
  def sumHelper(a: Int, b: Int): Int =
    if(a>b) 0
    else f(a) + sumHelper(a+1, b)
  sumHelper
}
```

Understanding the `sum` Function

If the above code appears a bit intimidating, don't worry. Let's go over it step by step.

Line 1 is defining the function `sum`.

```
def sum(f: Int => Int):(Int,Int) => Int
```

The dark green section of the illustration above represents the parameters of the function `sum` which is a single parameter that takes a function which further takes a single parameter of type `Int` and returns a value of type `Int`.

The red section of the illustration represents the return value of `sum` which is another function. `sum` returns a function which takes two parameters, both of type `Int`, and returns a value of type `Int`.

From **line 2** onwards, we have the body of the function `sum`. This starts with a definition of a nested function `sumHelper`. `sumHelper` takes 2 parameters, `a` and `b` (the upper and lower bounds) and returns a value of type `Int`. The function body of `sumHelper` is identical to the function body of the previous `sum` function.

Line 5 is simply returning the `sumHelper` function.

In conclusion, `sum` is now a function that returns another function; particularly a locally defined or nested function.

Modifying the Summation Functions

Below shows how the summation functions would now be redefined. Notice how the parameters of the functions aren't being specified anymore. This being said, when we call the functions, we still pass the upper and lower bounds as parameters.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
def sum(f: Int => Int): (Int,Int) => Int ={
  def sumHelper(a: Int, b: Int): Int =
    if(a>b) 0
    else f(a) + sumHelper(a+1, b)
  sumHelper
}

def factorial(x: Int): Int ={
  if(x==0) 1
  else x * factorial(x-1)
}

def sumOfInts = sum(x => x)
println(sumOfInts(1,5))

def sumOfCubes = sum(x => x*x*x)
println(sumOfCubes(1,5))

def sumOfFactorials = sum(factorial)
println(sumOfFactorials(1,5))
```

Let's further explore these concepts in the next lesson.