Solution Review: Using a Curried Function

In the following lesson, we will go over the solution of the challenge: Using a Curried Function.

We'll cover the following

- Task
- Solution

Task

In this challenge, you had to create a non-recursive factorial function `fact` in terms of a curried function `product` which calculates the product of the values of a function for the points on a given interval.

Solution

A skeleton of the function was already provided for you. Let's look it over.

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

def fact(n: Int) = {

}
```

`product` is a recursive curried function and has 2 parameter lists.

1. The first parameter list contains a single parameter; a function which has a single parameter of type `Int` and returns an integer.
2. The second parameter list contains two parameters `a` and `b`, both of type `Int`. `a` represented the minimum bound of the interval and `b` represented the maximum bound.

`fact` has a single parameter `n` of type `Int`. `n` is the integer in question whose factorial is to be computed factorial is to be computed.

To write the function body of `fact`, you needed to call `product` and pass it an anonymous function that acts as an identifier and returns the integer as is.

```
x => x
```

The second argument to be passed to `product` was the interval `a-b`. Since a factorial of a number `n` is simply the product of **1** and all consecutive numbers until `n`, our range was simply:

```
(1, n)
```

You can find the complete solution below:

> You were required to write the code on **line** 7.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```scala
def product(f: Int => Int)(a: Int, b: Int): Int ={
  if(a > b) 1
  else f(a) * product(f)(a+1,b)
}

def fact(n: Int) =
  product(x => x)(1, n)


// Driver Code
print(fact(5))
```

Let’s wrap up this chapter in the next lesson with a quiz to test what you have learned so far.