Summation with Higher-Order Functions

In this lesson, we will go over a practical example of higher-order functions and see a higher-order function in action.

We'll cover the following

Solution

To solve the problem discussed in the previous lesson, let's first define a higher-order recursive function `sum` which takes three parameters; a function `f` and two integers `a` and `b` (representing the lower and upper bounds respectively). The basic functionality of `sum` is to take the sum of all the integers between `a` and `b`. The function to be passed as an argument will perform some operation on the integers between `a` and `b`, ex. returning their cube. The result of the parameter function will then be summed to get the result of the higher-order function.

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

Before we can actually execute the `sum` function, we need to create small helper functions which we will pass to `sum`. You can write these helper functions based on your requirements. For our problem, we need three helper functions:

1. Which simply returns the specified integer
2. Which returns the cube of an integer
3. Which returns the factorial of an integer

```
def id(x: Int): Int = x
```

```
def cube(x: Int): Int = x*x*x

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

Now using our higher-order function and helper functions, let's redefine our summation functions `sumOfInts()`, `sumOfCubes()`, and `sumOfFactorials()` which we initially defined in the previous lesson.

`sumofInts()`

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)
}

def id(x: Int): Int = x
def cube(x: Int): Int = x*x*x
def factorial(x: Int): Int = if (x==0) 1 else x * factorial(x-1)

def sumOfInts(a: Int, b: Int) = sum(id, a, b)

// Driver Code
println(sumOfInts(1, 5))
```

`sumOfCubes()`

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)
}

def id(x: Int): Int = x
def cube(x: Int): Int = x*x*x
def factorial(x: Int): Int = if (x==0) 1 else x * factorial(x-1)

def sumofCubes(a: Int, b: Int) = sum(cube,a,b)

// Driver Code
println(sumofCubes(1,5))
```

`sumOfFactorials()`

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)
}

def id(x: Int): Int = x
def cube(x: Int): Int = x*x*x
def factorial(x: Int): Int = if (x==0) 1 else x * factorial(x-1)

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

// Driver Code
println(sumOfFactorials(1,5))
```

All three functions return the correct output. However, you might be thinking...this is a lot of work. While we did remove the redundant code, we still had to write many small functions, which is the result of passing functions as parameters. It becomes tedious to have to define, name, and keep track of these functions. Scala solves this problem by introducing anonymous functions.

Before we move on to anonymous functions, try writing your own higher-order function in the next lesson.