Currying

Functions Returning Functions

- Let's rewrite the sum function from last lecture in order to return a function that applies the given function parameter f and sums the results

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

Multiple Parameter Lists

- The definition of functions that return functions is so useful in functional programming that there is a special syntax for it in Scala
- For example, the following definition of sum is equivalent to the one with the nested sumF function:

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

Expansion of Multiple Parameter Lists

- In general, a definition of a function with multiple parameter lists: ***def f(ps$_1$) ... (ps$_n$) = E*** where n > 1, is equivalent to: ***def f(ps$_1$) ... (ps$_{n-1}$) = {def g(ps$_n$) = E; g}*** where g is a fresh identifier; or for short: ***def f(ps$_1$) ... (ps$_{n-1}$) = (ps$_n$ => E)***
- By repeating the process n times: ***def f(ps$_1$) ... (ps$_n$) = E*** is equivalent to:

  def f = (ps$_1$ => (ps$_2$ => ... (ps$_n$ => E) ...))
- This style of definition and function application is called currying
- The main idea of currying is that you can write any function as a sequence of anonymous functions that each takes one single parameter

Question: *Given* ***def sum (f: Int => Int) (a: Int, b: Int): Int = ...***, *what is the type of* ***sum***?

(Int => Int) => (Int, Int) => Int

Exercise:

1. *Write a product function that calculates the product of the values of a function for the points of a given interval.*

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

2. *Write factorial in terms of product.*

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

3. *Write a more general function, which generalizes both sum and product.*

```
def reduction(f: Int => Int, op: (Int, Int) => Int, zero: Int)(a: Int, b: Int): Int= {
  def loop(x: Int) :Int =
    if (x > b)
      zero
    else
      op(f(x), loop(x + 1))

  loop(a)
}

def sumReduction(f: Int => Int): (Int, Int) => Int = reduction(f, (x, y) => x + y, 0)

def productReduction(f: Int => Int): (Int, Int) => Int = reduction(f, (x, y) => x * y, 1)
```