# Expanding a Multiple Parameter List

In the following lesson, we will expand a multiple parameter list to see how a function with a multiple parameter list is executed.

**We'll cover the following**



## General Form #

The general form of a function with multiple parameter lists is as follows:

```
def f(args_1)...(args_n) = Exp
```

In the illustration above, `n` >**1**.

The above function is equivalent to the following:

```
def f(args_1)...(args_n-1) = {def h(args_n) = Exp; h}
```

The function `f` is taking the first `n-1` lists of parameters and creating a new function `h` which takes the `n`th list of parameters. `h` then maps the `n`th list of parameters to the function body `Exp` with `h` being the function that gets returned.

The above, can also be written using anonymous functions as follows.

```
def f(args_1)...(args_n-1) = (args_n => Exp)
```

The above function can be further expanded:

```
def f(args_1)...(args_n-2) = (args_n-1 => (args_n => Exp))
```

If we do this **n** times, we would get the following:

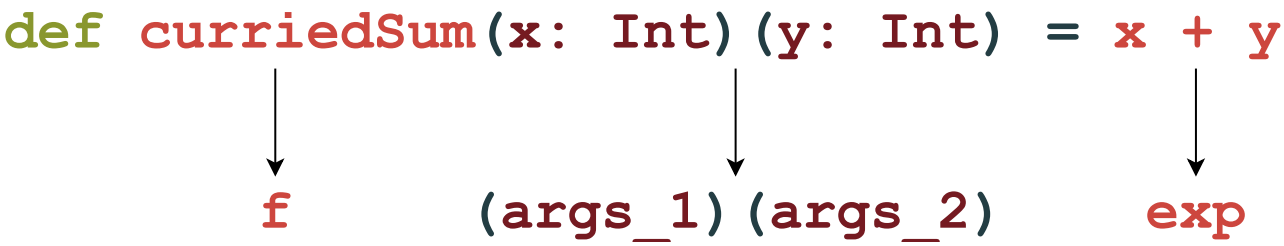
```
def f = (args_1 => (args_2 => ... (args_n => Exp)...))
```

## An Example #

We will expand the `curriedSum` function created in a previous lesson to better understand the general sequence of expansion explained above.

Just as a reminder. here is the `curriedSum` function:

```
def curriedSum(x: Int)(y: Int) = x + y
```




Let's start expanding!

```
def curriedSum(x: Int) = (y: Int) => x + y
```

In the first expansion `CurriedSum` is now a function which takes a single parameter of type `Int` and returns a function.

```
def curriedSum = (x: Int => ((y: Int) => x + y)
```

In conclusion, `curriedSum` is a combination of two nested function calls. As mentioned in a previous lesson, the first function call takes a single parameter of type `Int` and returns a function value which will be used by the second function (the function returned by the first function is the second function). The second function, in turn, takes a parameter `y` of type `Int` and returns the sum of `x` and `y`.

Let's try to implement both functions. The first function will be known as `first` and the second function will be known as `second`.

For our example, `x` = **3** and `y` = **2**.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
def first(x: Int) = (y: Int) => x+y

def second = first(3)

val result = second(2)

print(result)
```

`second` is defined by calling `first` and passing the value of `x` to the first function. We can then call `second` by passing the value of `y` to the second function which will in turn return in the final sum, i.e. **5**.

---

In the next lesson, you will be asked to make your own function using the currying syntax.