TOUR OF SCALA

# MULTIPLE PARAMETER LISTS (CURRYING)

Methods may have multiple parameter lists.

## Example

Here is an example, as defined on the `Iterable` trait in Scala's collections API:

```
  trait Iterable[A] {
  ...
  def foldLeft[B](z: B)(op: (B, A) => B): B
  ...
}
```

`foldLeft` applies a two-parameter function `op` to an initial value `z` and all elements of this collection, going left to right. Shown below is an example of its usage.

Starting with an initial value of 0, `foldLeft` here applies the function `(m, n) => m + n` to each element in the List and the previous accumulated value.

```
  val numbers = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
val res = numbers.foldLeft(0)((m, n) => m + n)
println(res) // 55
```

## Use cases

Suggested use cases for multiple parameter lists include:

### Drive type inference

It so happens that in Scala, type inference proceeds one parameter list at a time. Say you have the following method:

```
  def foldLeft1[A, B](as: List[A], b0: B, op: (B, A) => B) = ???
```

Then you'd like to call it in the following way, but will find that it doesn't compile:

```
  def notPossible = foldLeft1(numbers, 0, _ + _)
```

you will have to call it like one of the below ways:

```
  def firstWay = foldLeft1[Int, Int](numbers, 0, _ + _)
def secondWay = foldLeft1(numbers, 0, (a: Int, b: Int) => a + b)
```

That's because Scala won't be able to infer the type of the function `_ + _`, as it's still inferring `A` and `B`. By moving the parameter `op` to its own parameter list, `A` and `B` are inferred in the first parameter list. These inferred types will then be available to the second parameter list and `_ + _` will match the inferred type `(Int, Int) => Int`

```
  def foldLeft2[A, B](as: List[A], b0: B)(op: (B, A) => B) = ???
def possible = foldLeft2(numbers, 0)(_ + _)
```

This definition doesn't need any type hints and can infer all of its type parameters.

### Implicit parameters

To specify only certain parameters as `implicit`, they must be placed in their own `implicit` parameter list.

An example of this is:

```
  def execute(arg: Int)(implicit ec: scala.concurrent.ExecutionContext) = ???
```

### Partial application

When a method is called with a fewer number of parameter lists, then this will yield a function taking the missing parameter lists as its arguments. This is formally known as partial application.

For example,

```
  val numbers = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
val numberFunc = numbers.foldLeft(List[Int]()) _

val squares = numberFunc((xs, x) => xs :+ x*x)
println(squares) // List(1, 4, 9, 16, 25, 36, 49, 64, 81, 100)

val cubes = numberFunc((xs, x) => xs :+ x*x*x)
println(cubes)  // List(1, 8, 27, 64, 125, 216, 343, 512, 729, 1000)
```

← previous next →

Contributors to this page: