Kadir Malak

Follow 72 Followers About

Currying in Scala: A Useful Example

 Kadir Malak Dec 28, 2019 · 2 min read

Photo by Tom Hermans on Unsplash

Currying is the process of taking a function that accepts *N* arguments and turning it into a function that accepts *N-K (N minus K)* arguments, missing *K* arguments are fixed in the resulting function. It's named after Haskell Curry.

```scala
def someFunc(a: Int, b: String, c: Double) = {
  println("a: " + a + ", b: " + b + ", c: " + c)
}

someFunc(1, "str", 3.14) // we need all parameters to call it

// we're turning 3-arg function into 1-arg function
def someFuncCurried(a: Int, b: String) = {
  def temp(c: Double) = {
    someFunc(a, b, c) // a and b are remembered, only c is needed
  }
  temp _
}

val f = someFuncCurried(1, "str") // first 2 params saved
f(3.14) // now we can call using the third
```

currying-1.scala hosted with ♥ by GitHub view raw

Scala has a separate syntax to ease currying (notice that we first take 2 arguments and then a single argument)

```scala
// notice that we first take 2 arguments and then a single argument
def betterWayToCurry(a: Int, b: String)(c: Double) = {
  println("a: " + a + ", b: " + b + ", c: " + c)
}

val f2 = betterWayToCurry(1, "str") _ // first 2 params saved
f2(3.14) // now we can call using the third
```

currying-2.scala hosted with ♥ by GitHub view raw

We can even further increase the nesting:

```scala
def nestedCurrying(a: Int)(b: String)(c: Double) = {
  println("a: " + a + ", b: " + b + ", c: " + c)
}

def g1 = nestedCurrying(1) _ // we pass first param
def g2 = g1("str") // and then second param
g2(3.14) // and use it with third param

```

So, you may wonder if this type of functionality has any use in a programming language. Or you may say “All it’s doing is remembering the parameters, I can do this another way”. I’ll show the benefits of currying in a concrete example.

Suppose that we’re running a test code and we want to run the test with a combination of some parameters.

Here is the code without currying:

- the visual separation between the group of arguments
- no lambdas used
- you can clearly see that something is done in a *scope*

In Scala, if you have a function that takes a single parameter, you may call it like *f{argument} (using a {} block)* instead of *f(argument)*, if you combine this feature with proper currying, the full magic happens:

Look at it! It almost looks like a macro :) But it's not. Additional parameters are passed silently.

Look at it! It almost looks like a *macro* :) But it's not. Additional parameters are passed silently.

So you may use currying to…

- separate the parameters passed into logical groups and reuse the middle functions
- call the final function *parameter by parameter* as you obtain some values on the way (I'm assuming that you're passing the resulting functions around)
- combine it with Scala's single-parameter block usage - demonstrated above - and obtain macro-like results
- reduce the syntax when you find yourself returning functions

Functional Programming Currying Scala