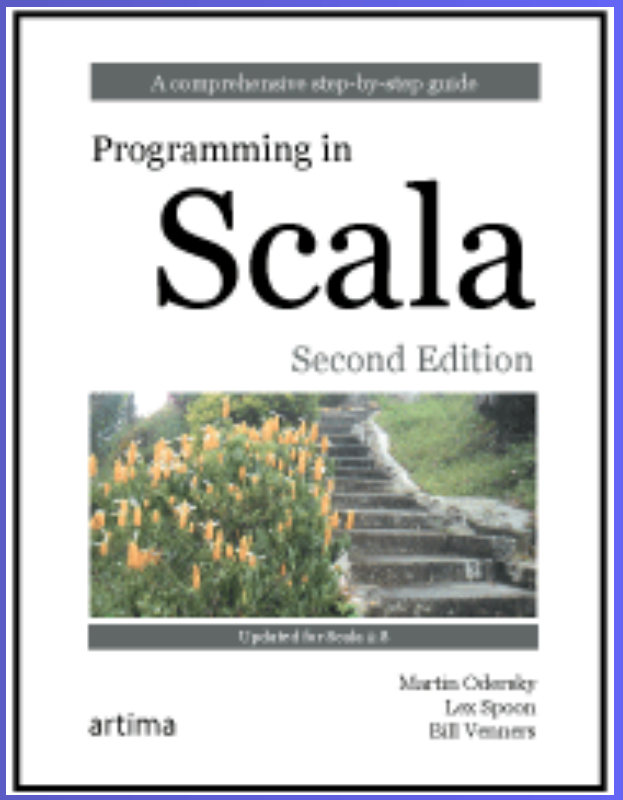
Stairway to Scala - Flight 6

Control abstraction

Bill Venners
Dick Wall



www.artima.com

Flight 6 goal

Learn to create your own control abstractions, and how to curry functions along the way.

Simplifying client code

- Problem: does a list of numbers contain any negatives?

```
def containsNeg(nums: List[Int]): Boolean = {
    var exists = false
    for (num <- nums)
        if (num < 0)
            exists = true
    exists
}

def containsNeg(nums: List[Int]) = nums.exists(_ < 0)
def containsLarge(nums : List[Int]) = nums.exists(_ > 100)
```

Printing current date to given file

```
def printCurrentDateToFile(file: File) {
    val writer = new PrintWriter(file)
    try {
        writer.println(new java.util.Date)
    } finally {
        writer.close()
    }
}
```

But what about if we want to do something else with the file?

Your own control abstraction

```
def withPrintWriter(file: File, op: PrintWriter => Unit) {
    val writer = new PrintWriter(file)
    try {
        op(writer)
    } finally {
        writer.close()
    }
}

withPrintWriter(new File("date.txt"),
    writer => writer.println(new java.util.Date))

withPrintWriter(new File("product.txt"),
    writer => { writer.print("2 times 3 is "); writer.print(2*3) })
```

Currying functions

```
scala> def plainOldSum(x: Int, y: Int) = x + y
plainOldSum: (Int,Int)Int
scala> plainOldSum(1, 2)
res4: Int = 3
```

Can also be done like this:

```
scala> def curriedSum(x: Int)(y: Int) = x + y
curriedSum: (Int)(Int)Int
scala> curriedSum(1)(2)
res5: Int = 3
```

What happens with currying?

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

scala> val second = first(1)
second: (Int) => Int = <function>

scala> second(2)
res6: Int = 3

scala> val onePlus = curriedSum(1)_
onePlus: (Int) => Int = <function>

scala> onePlus(2)
res7: Int = 3
```

Using currying in our control abstraction

```
def withPrintWriter(file: File)(op: PrintWriter => Unit) {
    val writer = new PrintWriter(file)
    try {
        op(writer)
    } finally {
        writer.close()
    }
}

val file = new File("date.txt")
withPrintWriter(file) {
    writer => writer.println(new java.util.Date)
}
```

By-name parameters

```
var assertionsEnabled = true
def myAssert(predicate: () => Boolean) =
    if (assertionsEnabled && !predicate())
        throw new AssertionError

myAssert(() => 5 > 3)
myAssert(5 > 3) // Won't work!

def byNameAssert(predicate: => Boolean) =
    if (assertionsEnabled && !predicate)
        throw new AssertionError

myAssert(5 > 3) // Works!
```