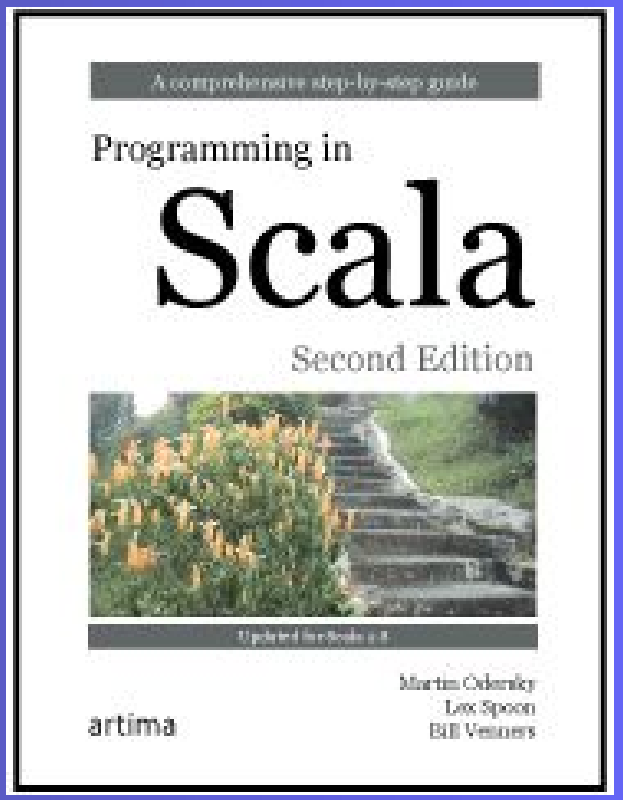
Stairway to Scala - Flight 1

# First steps in Scala

Bill Venners
Dick Wall




escalatesoft.com

Copyright (c) 2010-2014 Escalate Software Inc.
All Rights Reserved.

# Flight 1 goal

Introduce Scala, get you used to the interactive environment and how it can be used for experimentation, and write some Scala scripts.

# Scala interpreter:

```
$ scala
Welcome to Scala version 2.11.0
Type in expressions to have them evaluated.
Type :help for more information.
scala> 1 + 2
res0: Int = 3
scala> res0 * 3
res1: Int = 9
```

# The classic:

```
scala> println("Hello, world!")
Hello, world!
```

# Defining variables:

```
scala> val msg = "Hello, world!"
msg: String = Hello, world!

scala> val msg: java.lang.String = "Hello, world!"
msg: java.lang.String = Hello, world!

scala> val msg: String = "Hello, world!"
msg: String = Hello, world!
```

# Vals and vars:

```
scala> println(msg)
Hello, world!

scala> msg = "Goodbye, cruel world!"
<console>:5: error: reassignment to val
msg = "Goodbye cruel world!"

scala> var greeting = "Hello, world!"
greeting: String = Hello, world!

scala> greeting = "Leave me alone, world!"
greeting: String = Leave me alone, world!
```

# Odds and ends:

```
scala> val multiLine =
| "This is the next line."
multiLine: String = This is the next line.

scala> val oops
|
|
You typed two blank lines. Starting a new command.
```

# Code completion

```
scala> val s = "Hello, World"
s: String = Hello, World
```

s.<Hit Tab!>

What happens?

# Shell :commands

The Scala shell has several useful commands each begins with a :

**:help** prints up a list of the : commands

**:load** load up a scala file and evaluate it

**:history** print up a history of statements

**:replay** reset the shell and replay all commands
(:replay will reload files as you change them!)

**:sh** execute a shell command (and bring any results back in a useful form)

## Java's if statement:

```
if (a > b)
    System.out.println(a);
else
    System.out.println(b);
```

## Java's ternary operator:

```
int m = (a > b) ? a : b;
System.out.println(m);
```

## Scala's if expression:

```
val m = if (a > b) a else b
println(m)
```

# Defining a function:

```
scala> def max(x: Int, y: Int): Int = {
            if (x > y) x
            else y
        }
max: (Int,Int)Int

scala> max(3, 5)
res6: Int = 5
```

# Streamlining a function:

```
scala> def max2(x: Int, y: Int) = if (x > y) x else y
max2: (Int,Int)Int

scala> def greet() = println("Hello, world!")
greet: ()Unit

scala> greet()
Hello, world!

scala> :quit
$
```

# Scala scripting:

In a file named helloarg.scala:

```
// Say hello to the first argument
println("Hello, " + args(0) + "!")
```

```
$ scala helloarg.scala planet
Hello, planet!
```

# While loop:

```
var i = 0
while (i < args.length) {
  println(args(i))
  i += 1
}
```

```
$ scala printargs.scala Scala is fun
Scala
is
fun
```

# "Looping" with a for expression:

```
for (arg <- args)
  println(arg)
```

# "Looping" with foreach:

```
args.foreach(arg => println(arg))
```

```
$ scala pa.scala Concise is nice
Concise
Is
nice
```

```
args.foreach((arg: String) => println(arg))
```

```
args.foreach(println)
```

```
args foreach println
```

# The syntax of function literals in Scala

```
(x: Int, y: Int) => x + y
```

# Exercises

1. Print hello world from the interpreter.
2. Print hello world from a script.
3. In the interpreter, define a function that takes a string and an Int as parameters, and prints the string the Int number of times.
4. Write a script with a method that prints out three String arguments, each separated by one space on one line, but with each String reversed.
5. Now alter the script to also make each argument uppercase, then use :replay to try it out. See what happens?

# Exercises - extra credit

The Fibonacci Series in Mathematics can be calculated by the following formula:
fib(0) *equals* 0, fib(1) *equals* 1, fib(n) *equals* fib(n-1) + fib(n-2)

Write a script to complete the following definition:

```
def fib(v : Int) : Int = v match {
  case 0 => // Put something in here
  case 1 => // Put something in here
  case n => // Put something in here
}
```

Don't worry too much about the match syntax for now, just fill in the blanks, and test it using :load and :replay until it works.