Scala

Finally...

...something useful to do with the JVM.

Image source: http://www.tripadvisor.com/LocationPhotos-g187789-Lazio.html

Young

Developed in 2003 by Martin Odersky at EPFL

Martin also brought you `javac` and Java Generics

Don’t hold that against him, though

OO/Functional hybrid

Statically typed

Has a REPL (yay!)

Runs on the JVM and the CLR

Scala vs Java

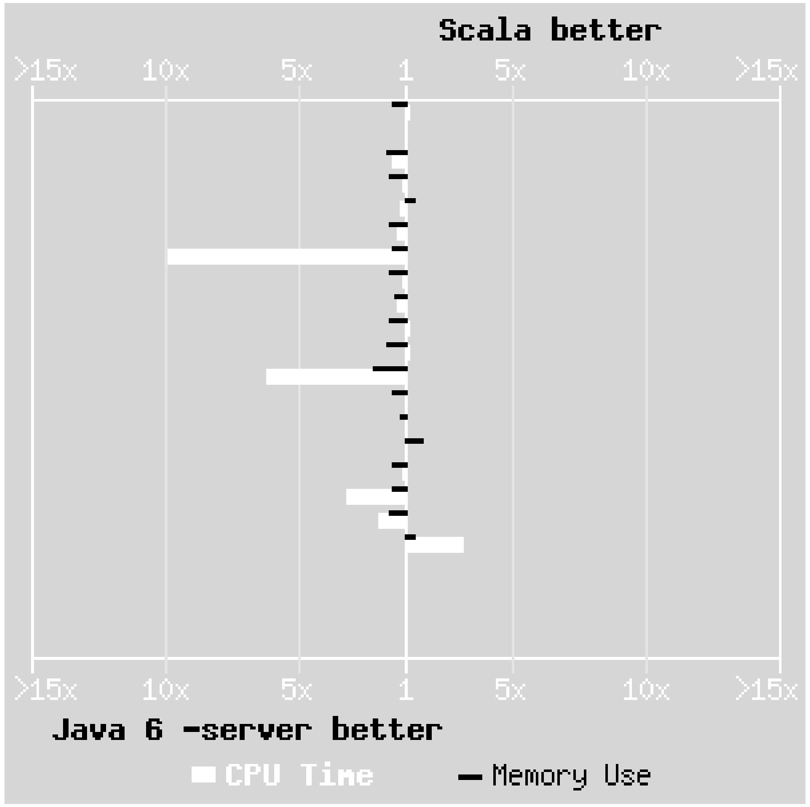


Image source: http://shootout.alioth.debian.org/gp4/scala.php

Scala vs Mono

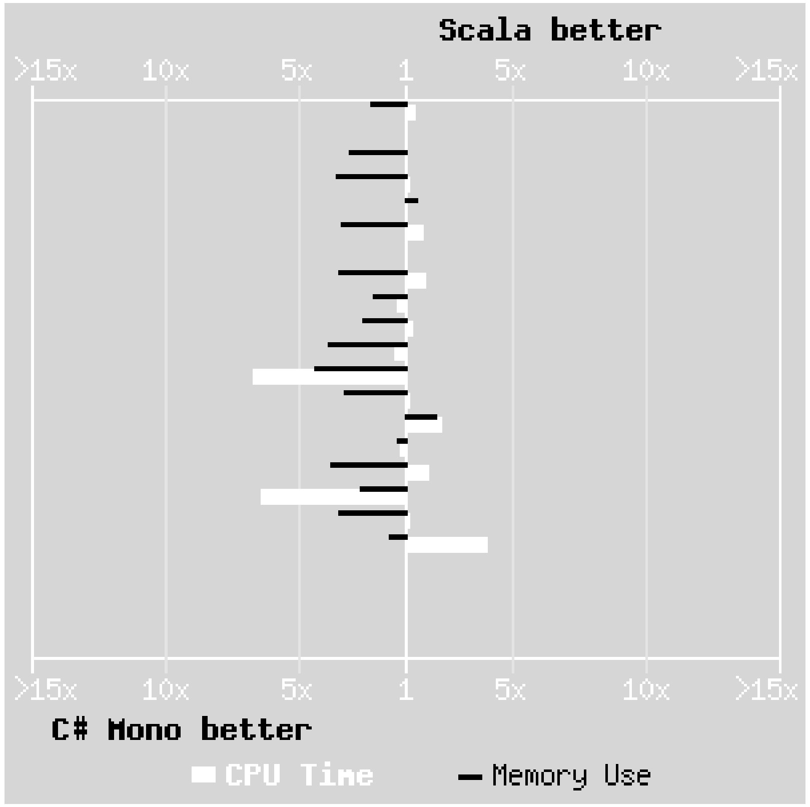


Image source: http://shootout.alioth.debian.org/gp4/benchmark.php?test=all&lang=scala&lang2=csharp

Can call existing Java code

Java can even call into Scala, too*

(*) most of the time

Same integer and float rules as Java

```
scala> 3 / 4
res3: Int = 0

scala>
scala> 3.0 / 4
res4: Double = 0.75

scala>
```

But, isn’t it statically typed?

type inference ==
awesome

```
scala> val msg = "Hello, Philly Lambda!"
msg: java.lang.String = Hello, Philly Lambda!

scala> val msg2 : String = "Its me again"
msg2: String = Its me again

scala>
```

```
c: Int = 10
```

The colon means “is type of”

Has both `values` and `variables`

`val`ues are immutable

```
scala> val c = 10000
c: Int = 10000

scala> c = 10001
line1$object.$iw.$iw.c = 10001
<console>:5: error: assignment to non-
variable
  val res0 = {c = 10001;c}
                ^


scala>
```

`var`iables are mutable

```
scala> var c = 10000
c: Int = 10000

scala> c = 10001
c: Int = 10001

scala> c
res2: Int = 10001

scala>
scala> println("Hello, world!")
Hello, world!
unnamed2: Unit = ()
```

`Unit == void`

Methods

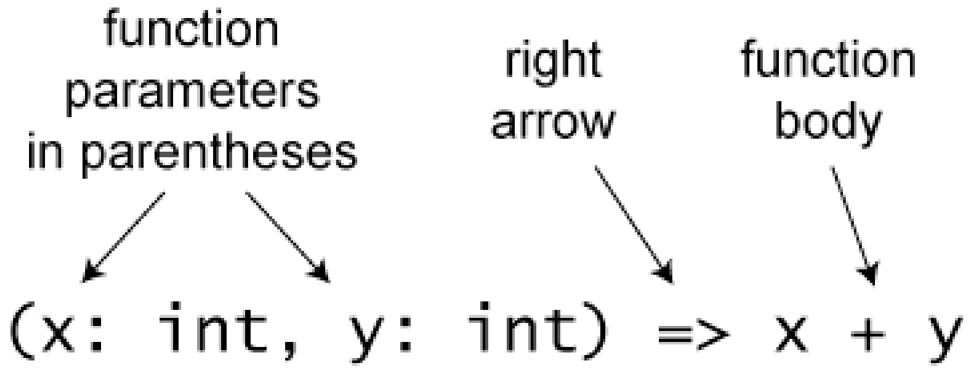
```
scala> def max(x: Int, y: Int) = if (x > y) x else y
max: (Int,Int)Int

scala> max(3, 5)
res5: Int = 5

scala>
```

Compiler cannot infer method parameter types

Sometimes it can infer the return parameter type*

(*) not if its recursive, though

Syntax of anonymous functions in Scala



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

Image source: http://www.artima.com/scalazine/articles/steps.html

You can write scripts in Scala, too

```
#!/bin/sh
exec scala $0 $@
!#

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

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

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

```
val ary = new Array[String](3)

val ary: Array[String] = new Array[String](3)
```

Has both `List` and `Tuple` built-ins a la Python

```
scala> val l1 = List(1,2)
l1: List[Int] = List(1, 2)

scala> val l2 = List(3,4)
l2: List[Int] = List(3, 4)

scala> l1 ::: l2
res0: List[Int] = List(1, 2, 3, 4)

scala>
scala> val x = List[String](1, 2, 3, "hello")
<console>:4: error: type mismatch;
 found   : Int(1)
 required: String
val x = List[String](1, 2, 3, "hello")
                     ^
<console>:4: error: type mismatch;
 found   : Int(2)
 required: String
val x = List[String](1, 2, 3, "hello")
                        ^
<console>:4: error: type mismatch;
 found   : Int(3)
 required: String
val x = List[String](1, 2, 3, "hello")
                           ^

scala>
scala> (1,2,3)
res14: (Int, Int, Int) = (1,2,3)

scala> ("hello",40,List('x','y'))
res15: (java.lang.String, Int, List[Char]) =
(hello,40,List(x, y))

scala> (1,2,3)._2
res16: Int = 2

scala>
```

Also has `Set` and `Map` classes

```
scala> val tvShows = Map(
     | 1 -> "World's Worst Cooking",
     | 2 -> "American Pariah",
     | 3 -> "Scala for n00bs",
     | 4 -> "Guy Steele is Looking for You",
     | 5 -> "699 Club"
     | )
tvShows: scala.collection.immutable.Map
[Int,java.lang.String] = Map(2 -> American
Pariah, 4 -> Guy Steele is Looking for You, 1
-> World's Worst Cooking, 3 -> Scala for
n00bs, 5 -> 699 Club)

scala>
```

Classes

```
class ConstructorShowoff(message: String) {
   var junk = “Can I haz Scala, plz?”

   def report() = println(message)

   def complete() = junk += “ Kthxbye”
}

val x = new ConstructorShowoff(“Yo, Adrienne”)
x.report
x.complete
println(x.junk)
```

```
class MoreHotness(message: String) {

   if (message == null)
      throw new NullPointerException(“message was null”)

}


class TwoConstructors(message: String, count: Int) {

   def this(message: String) = this(message, 1)

   def say() = {
      for (i <- 1 to count)
         println(message)
   }
}
```

No static fields or methods in Scala classes

Huh?

Instead, Scala has `singleton objects`

```scala
class ObjectWithCompanion(message: String) {
    def say() = {
        val whatToSay = ObjectWithCompanion.prepend(message)
        println(whatToSay)
    }
}

object ObjectWithCompanion {
    def prepend(x: String) = “Philly Lambda, “ + x
}
```

```
object PLApp {
    def main(args: Array[String]) {
        val o = ObjectWithCompanion(“Welcome to Scala!”)
        o.say()
    }
}
```

Traits and Mixins

```
trait Talky {
   def greet() = “Hi”
}

class WalMartGreeter extends Talky {
   override def greet() = “Welcome to Wal-Mart!”
}

class NewYorker extends Talky {
   override def greet() = “Fuck you”
}

var mouth: Talky = new NewYorker
println(mouth.greet())
```

```
trait Unsure extends Talky {
    override def greet() = super.greet() + “?”
}

val mouth: Talky = new NewYorker with Unsure
println(mouth.greet())
```

Something that will make Ed happy

```
def approximate(guess: Domain) : Domain =
    if (isGoodEnough(guess))
        guess
    else
        approximate(improve(guess))


def approximate(initialGuess: Domain) : Domain = {
    var guess = initialGuess
    while (!isGoodEnough(guess))
        guess = improve(guess)
    guess
}
```

They are the same

Scala supports TCO

Has pattern matching a la Prolog/Erlang, as well

Actors

```
scala> import scala.actors.Actor._
import scala.actors.Actor._

scala> val myActor = actor {
     | for (i <- 1 to 5)
     |  println("Anyone awake out there?")
     | Thread.sleep(1000)
     | }
myActor: scala.actors.Actor = scala.actors.Actor$$anon$0@7ddc70

scala> Anyone awake out there?
Anyone awake out there?
Anyone awake out there?
Anyone awake out there?
Anyone awake out there?


scala>
```

Uses the ! to send messages and pattern matching for `receive`

```
val echoActor = actor {
   while (true) {
      receive {
         case msg =>
            println(“received message: “ + msg)
      }
   }
}

echoActor ! “Uh, time to wrap it up, d00d”
```

Thanks for coming!

toby@cbcg.net