Scala

António Roldão, Ph.D. CQF

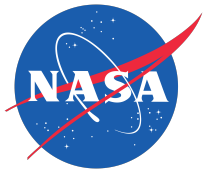
April/2015

Overview

- Origin of Scala
- Why Scala
- Remarkable Scala Projects
- Instrument Pricing Example
- The not so Good Parts
- Conclusion

Scala

- Design started around 2001 in Lausanne
- Academic Project at École Polytechnique
- Author is Martin Odersky
  - Previously worked on Funnel and Generic Java
  - Teaches two modules on Coursera
- Supported Commercially by TypeSafe Inc.
- Used at Many Reputable Institutions:



Morgan Stanley

Why Scala

- Interoperability with Java
- Sugar Syntax
- Type Safety
- Object Oriented
- Functional Programming
- Compiler Extensibility
- Rich Syntax
- Singletons Out-of-the-Box
- Macros / Meta-programming
- Lazy Values
- Dot Notation / Space
- Closures
- Operator Overloading
- Mixins
- Immutable “val”
- Pattern Matching
- Native Tuples
- Implicits
- XML Literals
- … etc.

Remarkable Scala Projects

Pricing Example

```
def BlackScholesBody( call: GlobalFloatP, put: GlobalFloatP, S: Float,
      X: Float, T: Float, R: Float , V: Float ) = {

   def sqrtT = SQRT(T)

   val d1 = (LOG(S / X) + (R + 0.5f * V * V) * T) / (V * sqrtT)
   val d2 = d1 - V * sqrtT
   val CNDD1 = CND(d1)
   val CNDD2 = CND(d2)

   val expRT = EXP(- R * T)
   call.d = (S * CNDD1 - X * expRT * CNDD2)
   put.d = (X * expRT * (1.0f - CNDD2) - S * (1.0f - CNDD1))
}
```

source: https://github.com/derekwhite/firepile

Pricing Example

```
def CND(float d): Float = {

   final val  A1 = 0.31938153f
   final val  A2 = -0.356563782f
   final val  A3 = 1.781477937f
   final val  A4 = -1.821255978f
   final val  A5 = 1.330274429f
   final val RSQRT2PI = 0.39894228040143267793994605993438f

   val K = 1.0f / (1.0f + 0.2316419f * fabs(d))

   val  cnd = RSQRT2PI * EXP(- 0.5f * d * d) *
      (K * (A1 + K * (A2 + K * (A3 + K * (A4 + K * A5)))))

   if(d > 0) cnd = 1.0f - cnd

   cnd
}
```

source: https://github.com/derekwhite/firepile

Not So Good Parts

- Slow Compilation
- Type System Requires More Context
- Functional Syntax Can be Hard to Follow
- Some Operators Depend on Context
- Non-intuitive Operators (for example: </>, =:=)
- No One Obvious Way to Express a Certain Algorithm
- Stuck in the Middle
- Still Too Verbose for a Relatively New Language
- Maps to Native JVM values, with Overflows, etc.

Not Ideal for Scientific Computing

- Slow and Silently Produces Wrong Values:

```
$ time scala -e 'println(1<<100)'
16

real  0m0.643s
user 0m0.302s
sys   0m0.066s
```

```
$ time python -c "print 1<<100"
1267650600228229401496703205376

real  0m0.019s
user 0m0.011s
sys   0m0.006s
```

```
$ time scala -e 'println(-2%10)'
-2

real  0m0.596s
user 0m0.311s
sys   0m0.069s
```

```
$ time python -c "print -2%10"
8

real  0m0.040s
user 0m0.012s
sys   0m0.011s
```

Conclusion

- Scala is a great academic language to explore new programing concepts
- Great to implement and test new concepts using compiler plugins

- Extends Java with new programing paradigms (e.g. functional)
- Transparently inter-operate with the vast amount of libraries from the Java eco-system

- Found a great position for Big Data Systems (for example: Spark)

- Can be very hard to read and maintain
- Compilation step severely limits the size of projects
- Some fundamental parts are dubious

Thank You