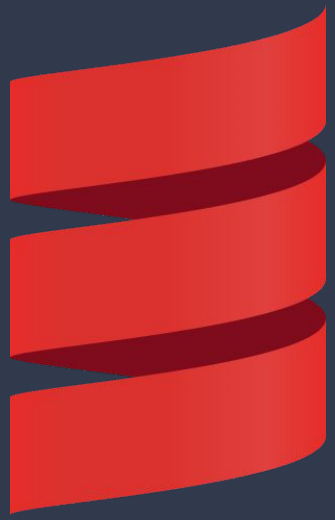
Learning Scala, Why?

M ª Cristina Heredia Gómez



Young but popular

- Scala =**Sca**lable + **la**nguage
- First public release in 2004
- Martin Odersky
- Compiles to Java byte code
- runs on JVM
- addresses the needs of the modern devs
- Twitter, Foursquare, The guardian and LinkedIn are using Scala

Why Scala?

1. **Scalability**: from small scripts to big data
2. **JVM and JS language**: exploits the JVM, tools & libraries for Java and has a js port
3. **Statically typed**: to create robust code but with type inference and more flexibility
4. **Mixed Paradigm OOP-FP**: object-oriented programming improving java object model & functional programming, the best tool for concurrency, big data and correct code
5. **Sophisticated type system**: extends Java type system with more flexible generics & type inference
6. **Concise, elegant syntax**: program the same, write less, ease to read

Scala examples

```java
// Java code
class Person {
  private int age;
  private String name;

  public Person(int age, String name) {
    this.age = age;
    this.name = name;
  }
}
```

```scala
// Scala code
class Person(age:Int,
             name:String)
```


Scala
def products = orders.flatMap(o => o.products)
public List<Product>getProducts(){
List<Product> products = new ArrayList<Product>();
for (Order order : order) {
products.addAll(order.getProducts());
}
return products;
}
Java
toptal

Scala examples

```
// src/main/java/progscala2/bigdata/HadoopWordCount.javaX
...
class WordCountMapper extends MapReduceBase
      implements Mapper<IntWritable, Text, Text, IntWritable> {

    static final IntWritable one  = new IntWritable(1);
    // Value will be set in a non-thread-safe way!
    static final Text word = new Text;

    @Override
    public void map(IntWritable key, Text valueDocContents,
        OutputCollector<Text, IntWritable> output, Reporter reporter) {
    String[] tokens = valueDocContents.toString.split("\\s+");
      for (String wordString: tokens) {
        if (wordString.length > 0) {
          word.set(wordString.toLowerCase);
          output.collect(word, one);
        }
      }
    }
  }

  class WordCountReduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text keyWord, java.util.Iterator<IntWritable> counts,
      OutputCollector<Text, IntWritable> output, Reporter reporter) {
    int totalCount = 0;
    while (counts.hasNext) {
      while (counts.hasNext) {
        totalCount += counts.next.get;
      }
      output.collect(keyWord, new IntWritable(totalCount));
    }
}
```

```
// src/main/scala/progscala2/bigdata/WordCountScalding.scalaX
import com.twitter.scalding._
class WordCount(args : Args) extends Job(args) {
  TextLine(args("input"))
    .read
    .flatMap('line -> 'word) {
      line: String => line.trim.toLowerCase.split("""\s+""")
    }
    .groupBy('word){ group => group.size('count) }
    .write(Tsv(args("output")))
}
```

Variable definition & collections

- **var** and **val**: mutable value or not
- **Lists**: all elements the same
- **Tuples**: elements could be diferent
- **Maps**
- **Sets**

Special forms of function calling

Going back to class Person example

For **classes** and **functions**:

- One can specify default values for params
- One can avoid respecting the params order by using its name

For **functions**:

the above also special forms of calling functions, but also:

- Repeated params - lists of variable length arguments with *

Pattern Matching

```
for {
  x <- Seq(1, 2, 2.7, "one")
} {
  val str = x match {
    case i: Int     => "an int: "+i
    case d: Double  => "a double: "+d
    case s: String  => "an string: "+s
    case _ => "unexpected value"
  }
  println(str)
}
```

- Like familiar statements in C-like language
- More flexible

Traits

- Scala replacement of Java's Interfaces
- Allow to define methods (Until Java 8 you couldn't)
- Can define *instance* fields
- Enable true composition of behaviour

```
trait Iterator[A] {
  def hasNext: Boolean
  def next(): A
}

class IntIterator(to: Int) extends Iterator[Int] {
  private var current = 0
  override def hasNext: Boolean = current < to
  override def next(): Int =  {
    if (hasNext) {
      val t = current
      current += 1
      t
    } else 0
  }
}

val iterator = new IntIterator(10)
iterator.next()  // prints 0
iterator.next()  // prints 1
```

High order functions

Are functions that take one or more functions as arguments.

Examples: applying twice a function

- Scala

```scala
def twice(f: Int => Int) = f compose f
twice(_ + 3)(7) // 13
```

- Java 8+

```java
Function<Function<Integer, Integer>, Function<Integer, Integer>>
twice = f -> f.andThen(f);
twice.apply(x -> x + 3).apply(7); // 13
```

Common FP functions

- `filter(p: (A) ⇒ Boolean)`
- `find(p: (A) ⇒ Boolean)`
- `flatMap[B](f: (A) ⇒ GenTraversableOnce[B])`
- `reduce[A1 >: A](op: (A1, A1) ⇒ A1)`
- `fold[A1 >: A](z: A1)(op: (A1, A1) ⇒ A1)`
- `aggregate[B](z: ⇒ B)(seqop: (B, A) ⇒ B, combop: (B, B) ⇒ B)`
- `exists(p: (A) ⇒ Boolean)`
- `forall(p: (A) ⇒ Boolean)`

Thanks!

@_musicalnote

mrcrstnherediagmez@gmail.com

github.com/CristinaHG/Scala-Intro

References

- Image for toptal : https://www.toptal.com/scala/why-should-i-learn-scala
- Programming Scala - Dean Wampler and Alex Payne