databricks™

Scala Crash Course

Holden Karau - @holdenkarau

Scala vs Java API vs Python

Spark was originally written in Scala, which allows concise function syntax and interactive use

Java API added for standalone applications

Python API added more recently along with an interactive shell.

This course: mostly Scala, some translations shown to Java & Python

Outline

Introduction to Scala & functional programming

A picture of a cat

Coffee Break*

*The coffee break may or may not be a lie.

Introduction to Scala

What is Scala?

Functions in Scala

Operating on collections in Scala

About Scala

High-level language for the JVM

- Object oriented + functional programming

Statically typed

- Comparable in speed to Java*
- Type inference saves us from having to write explicit types most of the time

Interoperates with Java

- Can use any Java class (inherit from, etc.)
- Can be called from Java code

Best way to Learn Scala

Interactive scala shell (just type scala)

Supports importing libraries, tab completing, and all of the constructs in the language

http://www.scala-lang.org/

Quick Tour of Scala

Declaring variables:

```scala
var x: Int = 7
var x = 7 // type inferred
val y = "hi" // read-only
```

Java equivalent:

```java
int x = 7;

final String y = "hi";
```

Functions:

```scala
def square(x: Int): Int = x*x
def square(x: Int): Int = {
  x*x
}
def announce(text: String) =
{
  println(text)
}
```

Java equivalent:

```java
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

databricks™

Scala functions (closures)

```
(x: Int) => x + 2 // full version
```

Scala functions (closures)

```
(x: Int) => x + 2 // full version

x => x + 2 // type inferred
```

databricks™

Scala functions (closures)

```
(x: Int) => x + 2 // full version

x => x + 2 // type inferred

_ + 2 // placeholder syntax (each argument must be used
exactly once)
```

Scala functions (closures)

```
(x: Int) => x + 2 // full version

x => x + 2 // type inferred

_ + 2 // placeholder syntax (each argument must be used
exactly once)

x => { // body is a block of code
    val numberToAdd = 2
    x + numberToAdd
}
```

databricks™

Scala functions (closures)

```
(x: Int) => x + 2 // full version

x => x + 2 // type inferred

_ + 2 // placeholder syntax (each argument must be used
exactly once)

x => { // body is a block of code
    val numberToAdd = 2
    x + numberToAdd
}

// Regular functions
def addTwo(x: Int): Int = x + 2
```

databricks™

Quick Tour of Scala Part 2

(electric boogaloo)

Processing collections with functional programming

```
val lst = List(1, 2, 3)
list.foreach(x => println(x)) // prints 1, 2, 3
list.foreach(println)         // same

list.map(x => x + 2)     // returns a new List(3, 4, 5)
list.map(_ + 2)          // same

list.filter(x => x % 2 == 1)// returns a new List(1, 3)
list.filter(_ % 2 == 1)     // same

list.reduce((x, y) => x + y) // => 6
list.reduce(_ + _)           // same
```

All of these leave the list unchanged as it is immutable.

Functional methods on collections

There are a lot of methods on Scala collections, just **google Scala Seq** or http://www.scala-lang.org/api/2.10.4/index.html#scala.collection.Seq

Method on Seq[T]	Explanation
map(f: T => U): Seq[U]	Each element is result of f
flatMap(f: T => Seq[U]): Seq[U]	One to many map
filter(f: T => Boolean): Seq[T]	Keep elements passing f
exists(f: T => Boolean): Boolean	True if one element passes f
forall(f: T => Boolean): Boolean	True if all elements pass
reduce(f: (T, T) => T): T	Merge elements using f
groupBy(f: T => K): Map[K, List[T]]	Group elements by f
sortBy(f: T => K): Seq[T]	Sort elements
…..	

databricks™

Cat picture from http://galato901.deviantart.com/art/Cat-on-Work-Break-173043455