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

Functions:
def square(x: Int): Int = x*x
def square(x: Int): Int = {
  x*x
}
def announce(text: String) = {
  println(text)
}
Scala functions (closures)

(x: Int) => x + 2  // full version
Scala functions (closures)

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

x => x + 2 // type inferred
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
}
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
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](http://www.scala-lang.org/api/2.10.4/index.html#scala.collection.Seq)

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