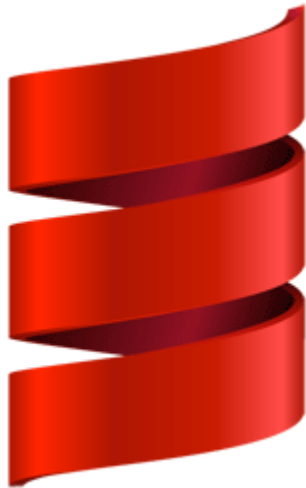




An Introduction to



Scala

OBJECT - FUNCTIONAL
- ORIENTED

Apostolos N. Papadopoulos
Assistant Professor

The Very Basics

“If I were to pick a language to use today other than Java, it would be Scala.”

—James Gosling
(father of Java)



The Very Basics

Scala = **SCA**lable **L**anguage

i.e., designed to grow with the demands of its users

Development started in 2001 by **Martin Odersky**
and his team at EPFL

First release in **January 2004**

Current version **2.11.7** (June 2015)



The Very Basics

Scala is a general-purpose programming language that runs on [Java Virtual Machine](#) (JVM) and .NET

Expresses common programming patterns in a **concise, elegant, and type-safe** way.

Scala supports both the **object-oriented** and the **functional** programming model.

The Very Basics

Scala is object-oriented:

- Encapsulation
- Inheritance
- Polymorphism
- All predefined types are objects
- All user-defined types are objects
- Objects communicate by **message exchange**

The Very Basics

Scala is functional:

- Functions are first-class values
- Can define a function in another function
- Can map input values to output values
- Can do lazy evaluation
- Supports pattern matching
- Higher-order functions

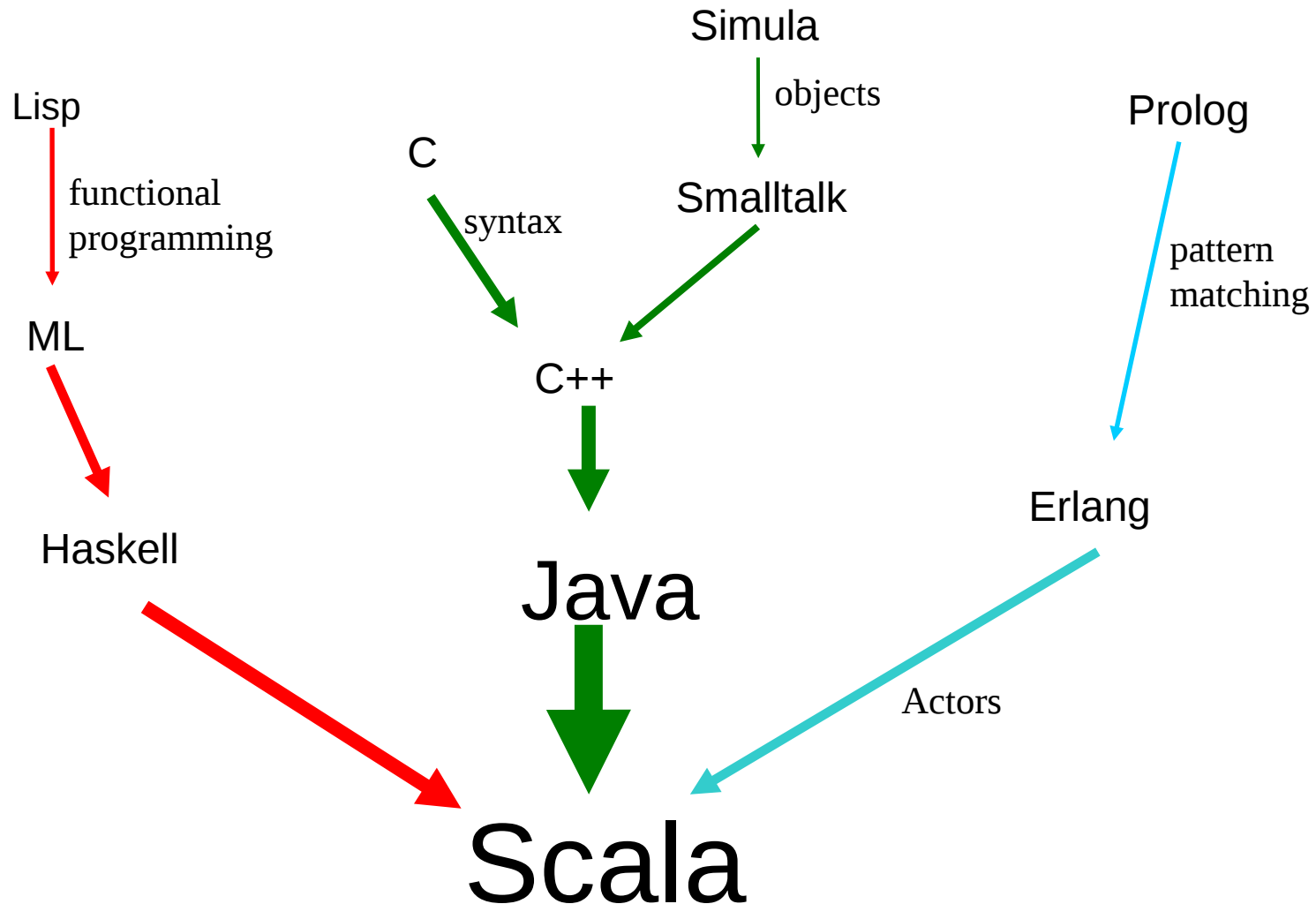
Scala is **not a PURE** functional language however.

The Very Basics

Scala has been inspired by other programming languages:

- Scala's object model was pioneered by **Smalltalk** and taken up subsequently by **Ruby**
- Scala adopts a large part of the syntax of **Java** and **C#**
- Its idea of universal nesting (almost every construct in Scala can be nested inside any other construct) is also present in **Algol**, **Simula**, and, more recently in **Beta**
- Its uniform access principle for method invocation and field selection comes from **Eiffel**
- Its approach to functional programming is quite similar in spirit to the **ML family** of languages, which has **SML**, **OCaml**, and **F#** as prominent members
- It adopts the Actor model for concurrent computation from **Erlang**

The Very Basics



The Very Basics

Scala is a **statically typed language** like Java.

In static typing, a variable is bound to a particular type for its lifetime. Its type can't be changed and it can only reference type-compatible instances.

That is, if a variable refers to a value of type A , you can't assign a value of a different type B to it, unless B is a subtype of A , for some reasonable definition of “subtype.”

This is different than in dynamically typed languages such as Ruby, Python, Groovy, JavaScript, Smalltalk and others.

Scala vs Java

Java code:

```
class Book {  
    private String author;  
    private String title;  
    private int    year;  
  
    public Book(String author, String title, int year) {  
        this.author = author;  
        this.title  = title;  
        this.year   = year;  
    }  
  
    public void setAuthor(String author) { this.author = author; }  
    public void String getAuthor() { return this.author; }  
    public void setTitle(String title) { this.title = title; }  
    public void String getTitle() { return this.title; }  
    public void setYear(int year) { this.age = year; }  
    public void int getYear() { return this.year; }  
}
```

Scala code:

```
class Book (var author: String, var title: String, var year: Int)
```

Scala vs Java

Lets check it out:

```
class Book (var author: String, var title: String, var year: Int)
```

Assume the previous class declaration is in the file
Book.scala

We run the Scala compiler using

```
scalac Book.scala
```

Then, we use the class disassembler

```
$JAVA_HOME/bin/javap -private Book.class
```

Scala vs Java

```
public class Book {  
    private java.lang.String author;  
    private java.lang.String title;  
    private int year;  
    public java.lang.String author();  
    public void author_$eq(java.lang.String);  
    public java.lang.String title();  
    public void title_$eq(java.lang.String);  
    public int year();  
    public void year_$eq(int);  
    public Book(java.lang.String, int);  
}
```

The First Scala Program

A very simple program in Scala:

```
/* Our first Scala program */  
object HelloWorld {  
  /* The main function */  
  def main(args: Array[String]) {  
    println("Hello, world!")  
  }  
}
```

To define a singleton we use the keyword **object**

Scala Data Types

Byte	8 bit signed value, Range from -128 to 127
Short	16 bit signed value. Range -32768 to 32767
Int	32 bit signed value. Range -2147483648 to 2147483647
Long	64 bit signed value. -9223372036854775808 to 9223372036854775807
Float	32 bit IEEE 754 single-precision float
Double	64 bit IEEE 754 double-precision float
Char	16 bit unsigned Unicode character. Range from U+0000 to U+FFFF
String	a sequence of Chars

Simple Functions

// count from 1 to 10

```
def countTo(n: Int) {  
    for (i <- 1 to 10) {  
        println(i)  
    }  
}
```

// return true if a number is even, false otherwise

```
def isEven(n: Int) = {  
    val m = n % 2  
    m == 0  
}
```

Immutable vs Mutable

An **immutable** element cannot change its value. Immutable elements help in creating more robust parallel programs because concurrency is much more easier for immutable values.

A **mutable** element can change its value. Scala supports both immutable and mutable elements and it is up to the program to use these features.

e.g.,

```
val pi = 3.14 // pi is immutable, is defined as a val (value)
```

```
var sal = 10,000 // sal is mutable, it is defined as a var (variable)
```


Imperative vs Functional

Imperative programming is the way we program in C, C++, Java and similar languages. It is heavily based on **mutable elements** (i.e., variables) and we need to specify every single step of an algorithm.

Scala supports imperative programming, but the real power of the language is the **functional** perspective which is based on **immutable elements**.

Interactive Scala

The REPL (Read – Evaluate – Print Loop)

It is a Scala **Shell**

Useful for fast testing, without the need to write complete programs. Runs each scala command immediately.

Interactive Scala

A screenshot of the Scala REPL

```
apostol@dell:~$ scala
Welcome to Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_40).
Type in expressions to have them evaluated.
Type :help for more information.

scala> val msg = "Hello World"
msg: String = Hello World

scala> msg.split(" ").foreach(println)
Hello
World

scala> █
```

First Steps in Scala

Define a list of pets:

```
val pets = List("dog", "cat", "parrot")
```

Print the contents of the list:

```
pets.foreach(println)
```

Print the length of each string:

```
pets.foreach(pet => println(pet.length))
```

First Steps in Scala

Split a sentence to words

```
val msg = "Hello World"
```

```
msg.split(" ").foreach(println)
```

Result:

Hello

World

First Steps in Scala

```
val msg = "Polytechnique"  
msg.drop(3).take(2).capitalize
```

Result:

Yt

First Steps in Scala

Iterate over string characters

```
val msg = "hello polytechnique"
```

```
msg.map(c => c.toUpperCase)
```

Result:

HELLO POLYTECHNIQUE

First Steps in Scala

Filtering strings

```
val msg = "This is a text"
msg.filter(_ != 'i').map(c => c.toUpperCase)
msg.filter(_ != 'i').map(_.toUpperCase)
```

} *equivalent*

Result:

THS S A TEXT

First Steps in Scala

Random numbers

```
val r = scala.util.Random
```

```
r.nextInt
```

```
r.nextInt(100)
```

```
r.nextFloat
```

```
r.nextDouble
```

```
// An array of random numbers
```

```
val vector = Array.fill(10){r.nextInt(9)}
```

First Steps in Scala

List examples

// List of Strings

```
val fruit: List[String] = List("apples", "oranges", "pears")
```

// List of Integers

```
val nums: List[Int] = List(1, 2, 3, 4)
```

// An empty List

```
val empty: List[Nothing] = List()
```

// A two-dimensional List

```
val dim: List[List[Int]] = List(List(1, 0, 0),  
                                List(0, 1, 0),  
                                List(0, 0, 1)  
)
```

First Steps in Scala

Set examples

```
var users = Set("Mary", "John", "Fred", "Julia")
```

```
users("Mary") // returns true
```

```
users("Ted") // returns false
```

```
users += "Jack" // inserts "Jack" into the users
```

```
users -= "Fred" // removes "Fred" from users
```

What if we had declared

```
val users = Set("Mary", "John", "Fred", "Julia")
```

First Steps in Scala

Duplicate elimination

```
val nums: List[Int] = List(1, 2, 1, 3, 4, 3)  
nums.distinct
```

Result

```
List (1, 2, 3, 4)
```

First Steps in Scala

Folding

A mechanism to traverse the contents of container applying some function **as we go**

```
val myList = List(1,2,3,4,5)  
myList.foldLeft(0)((a,b) => a+b)
```

Result

???

First Steps in Scala

Folding

A mechanism to traverse the contents of container applying some function **as we go**

```
val myList = List(1,2,3,4,5)  
myList.foldLeft(0)((a,b) => a+b)
```

Result

```
List (1, 2, 3, 4)
```

First Steps in Scala

Exercise: multiply all elements of a List using folding

```
val myList = List(1,2,3,4,5)
```

First Steps in Scala

Exercise: multiply all elements of a List using folding

```
val myList = List(1,2,3,4,5)  
myList.foldLeft(1) ((a,b) => a*b)
```


Counting Lines of a File

```
object LineCount {  
  def main(args: Array[String]) {  
    val inputFile = "leonardo.txt"  
    val src = scala.io.Source.fromFile(inputFile)  
    val counter = src.getLines().map(line => 1).sum  
    println("Number of lines in file: "+counter)  
  }  
}
```

WordCount v1: idea

Use a hashmap, which stores (word,counter) pairs.

The hashmap is updated in every word occurrence. Either a new word is inserted with counter=1, or the counter is incremented.

WordCount v1: code

```
import scala.io.Source

object WordCount {
  def main(args: Array[String]) {

    val lines = Source.fromFile("leonardo.txt").getLines.toArray
    val counts = new collection.mutable.HashMap[String, Int].withDefaultValue(0)
    lines.flatMap(line => line.split(" ")).foreach(word => counts(word) += 1)

    println(counts)

  }
}
```

WordCount v2: idea

Group words on linked lists.

Each linked list is responsible to store a single word. Using groupBy all same words are grouped together in the same linked list.

The number of occurrences of a word equals the length of the linked list.

WordCount v2

```
import scala.io.Source
object WordCount {
  def main(args: Array[String]) {

    val counts = Source.fromFile("leonardo.txt").
      getLines().
      flatMap(_.split("\\W+")).
      toList.
      groupBy((word: String) => word).
      mapValues(_.length)

    println(counts)
  }
}
```

WordCount v3: idea

Use flatMap and foldLeft.

This is more efficient, since we avoid the costly **groupByKey** operation.

WordCount v3: code

```
import scala.io.Source

object WordCount {

  def main(args: Array[String]) {

    val counts = Source.fromFile("leonardo.txt").
      getLines().
      flatMap(_.split("\\W+")).
      foldLeft(Map.empty[String, Int]){
        (count, word) => count + (word -> (count.getOrElse(word, 0) + 1))
      }

    println(counts)

  }

}
```

Quicksort v1

```
var xs: Array[Double]

def swap(i: Int, j: Int) {
    val t = xs(i); xs(i) = xs(j); xs(j) = t
}

def sort1(l: Int, r: Int) {
    val pivot = xs((l + r) / 2)
    var i = l
    var j = r
    while (i <= j) {
        while (xs(i) < pivot) i += 1
        while (xs(j) > pivot) j -= 1
        if (i <= j) {
            swap(i, j); i += 1; j -= 1
        }
    }
    if (l < j) sort1(l, j)
    if (j < r) sort1(i, r)
}

sort1(0, xs.length - 1)
```


Quicksort v2

```
object QuickSort {  
  
  def quick(xs: Array[Int]): Array[Int] = {  
  
    if (xs.length <= 1) xs  
    else {  
      val pivot = xs(xs.length / 2)  
      Array.concat(quick(xs filter (_ < pivot)),  
        xs filter (_ == pivot), quick(xs filter (_ > pivot)))  
    }  
  }  
}
```

Array Example

```
object ArrayDemo {  
  def mySquare(arr: Array[Int]): Array[Int] = {  
    arr.map(elem => elem * elem)  
  }  
  def myCube(arr: Array[Int]): Array[Int] = {  
    arr.map(elem => elem*elem*elem)  
  }  
  
  def main(args: Array[String]) {  
    // fill the array with random numbers  
    val vector = Array.fill(10){scala.util.Random.nextInt(9)}  
  
    println(vector.mkString(", "))  
    println(mySquare(vector).mkString(", "))  
    println(myCube(vector).mkString(", "))  
  }  
}
```

Traits

- Similar to **interfaces** in Java
- They may have implementations of methods
- But cannot contain state
- Can be multiply inherited from

Trait Example

```
trait Similarity {  
  def isSimilar(x: Any): Boolean  
  def isNotSimilar(x: Any): Boolean = !isSimilar(x)  
}
```

```
class Point(xc: Int, yc: Int) extends Similarity {  
  var x: Int = xc  
  var y: Int = yc  
  def isSimilar(obj: Any) =  
    obj.isInstanceOf[Point] &&  
    obj.asInstanceOf[Point].x == x  
}
```

```
object TraitsTest extends Application {  
  val p1 = new Point(2, 3)  
  val p2 = new Point(2, 4)  
  val p3 = new Point(3, 3)  
  println(p1.isNotSimilar(p2))  
  println(p1.isNotSimilar(p3))  
  println(p1.isNotSimilar(2))  
}
```

Actors

A strong aspect of Scala is its ability to develop concurrent programs.

The language supports the Actor model (adopted from Erlang)

What is an actor?

Actors are normal objects that are created by instantiating subclasses of the **Actor class**.

Actors

Actors may collaborate by message exchange.

If actor A1 sends a message to actor A2, the message is stored in the mailbox of A2 and it will be processed in turn.

When A2 finishes processing of the current message, handles the next one from the mailbox.

Actor Example

```
import akka.actor.Actor
import akka.actor.ActorSystem
import akka.actor.Props
```

```
class HelloActor extends Actor {
  def receive = {
    case "hello" => println("hello back at you")
    case _       => println("huh?")
  }
}
```

```
object Main extends App {
  val system = ActorSystem("HelloSystem")
  val helloActor = system.actorOf(Props[HelloActor], name = "helloactor")
  helloActor ! "hello"
  helloActor ! "buenos dias"
}
```

Reference: Scala Cookbook

Resources

Recommended links for Scala programming

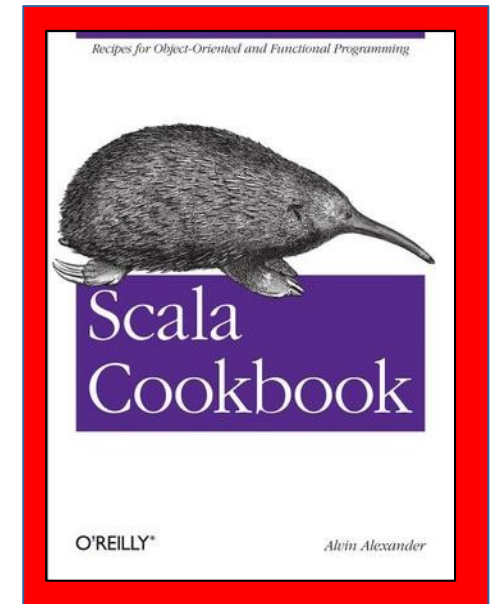
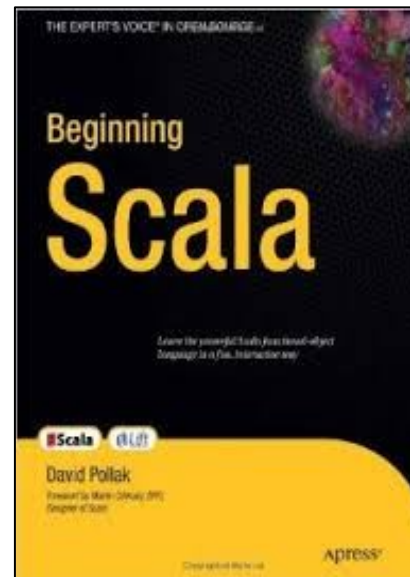
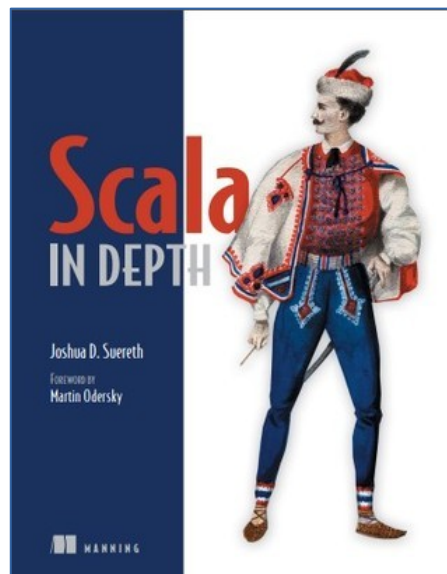
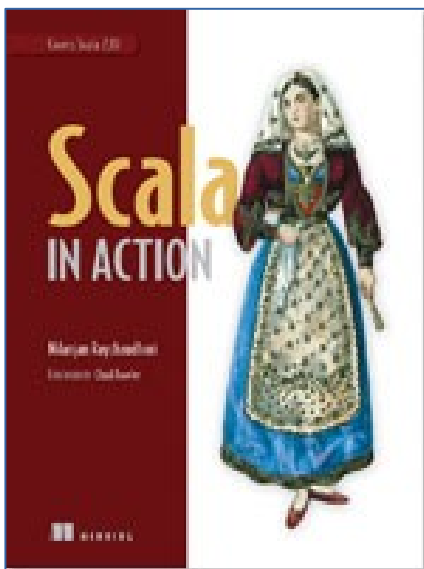
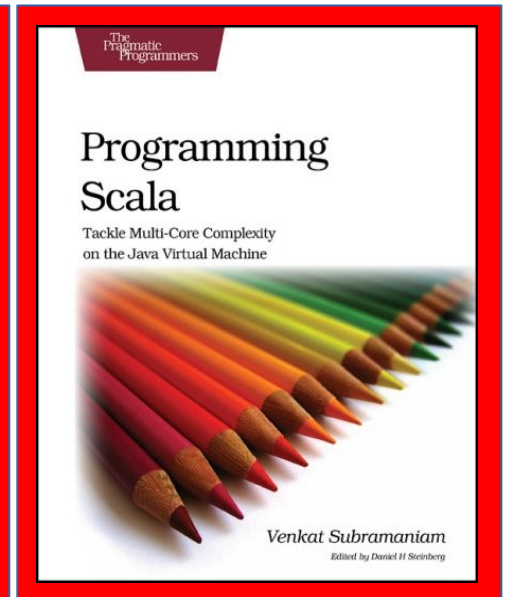
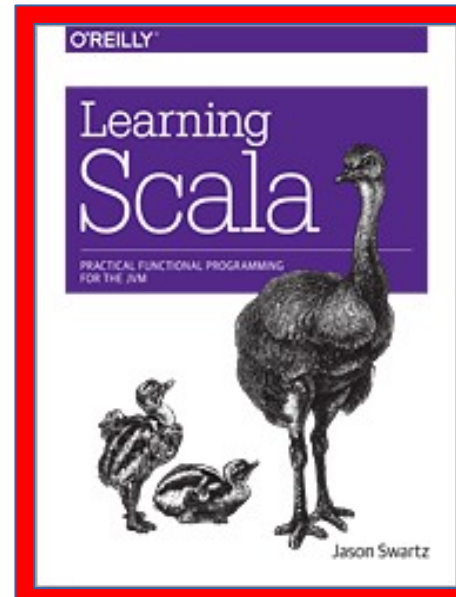
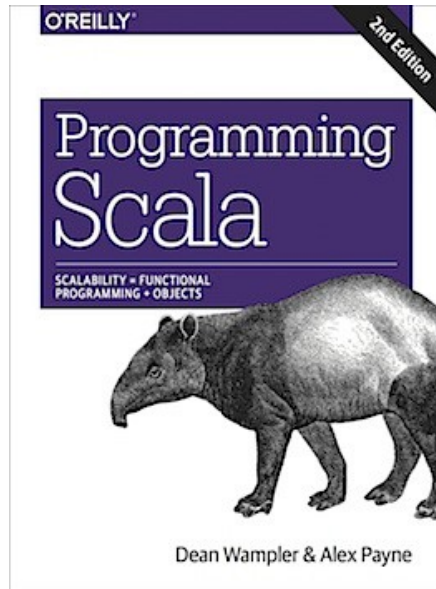
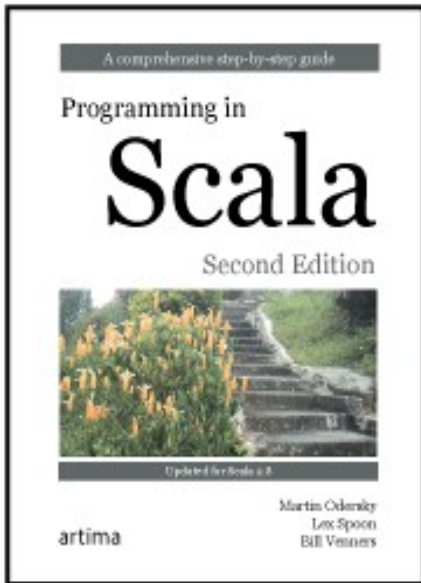
Official Scala Website

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

Scala School

https://twitter.github.io/scala_school

Resources



Tools to Learn Scala

Use an applet to test some Scala code in your browser (visit <http://www.simplyscala.com>)

Download a scala version, install it and use the **REPL** for testing.

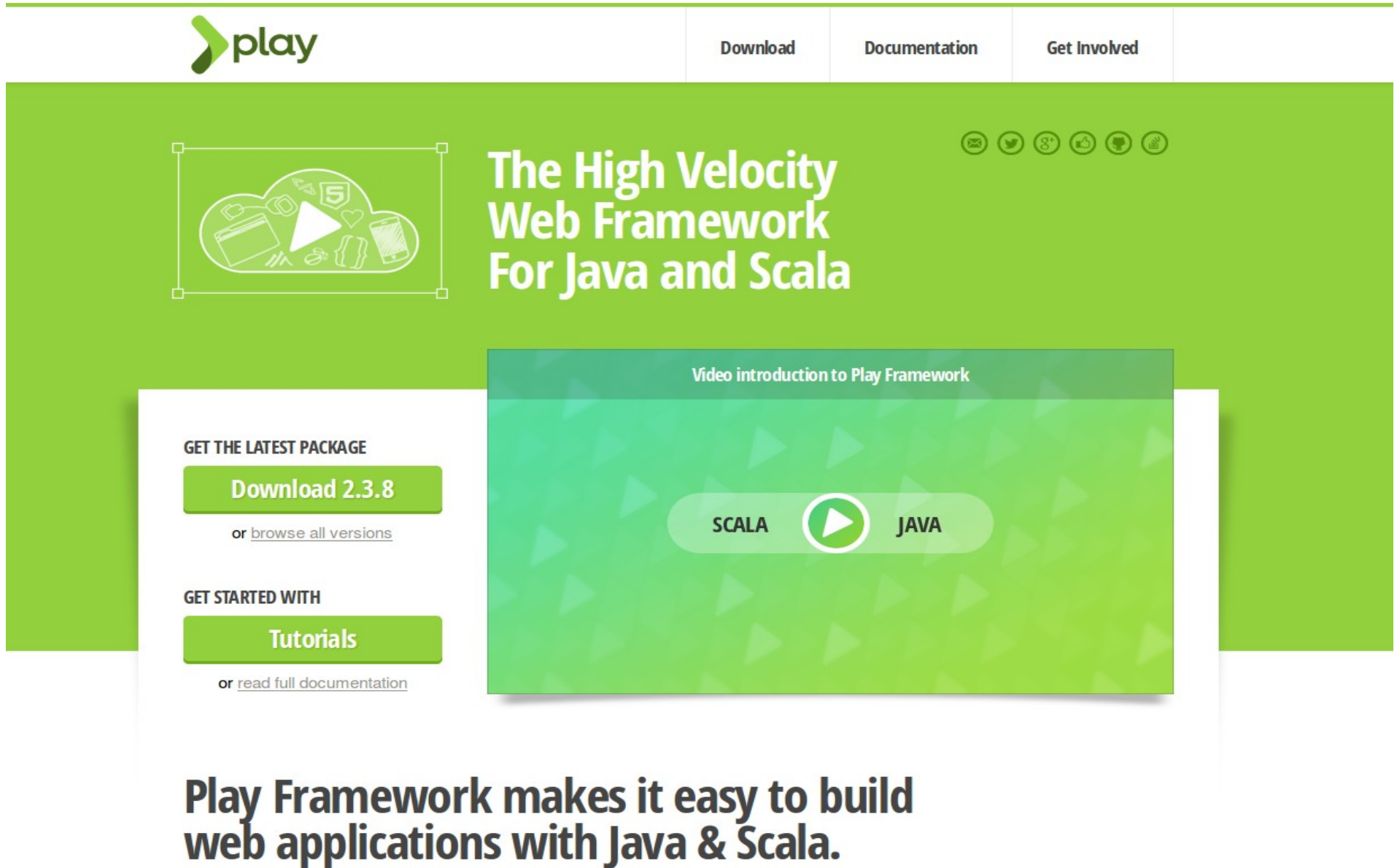
Try **scala-notebook**, a web-based interface to test scala code (it has more features than the applet).

Download the tool **Typesafe Activator** (<http://www.typesafe.com/get-started>) and use Scala through your browser.

Use Scala from an IDE, like **Netbeans**, **Eclipse** or **IntelliJ**

You can use Linux, Windows or Mac as long as you have a recent (at least 1.6) JDK installed in your system.

More Scala



The screenshot shows the Play Framework website with a green header and a large green hero section. The header contains the Play logo and navigation links for Download, Documentation, and Get Involved. The hero section features a cloud icon with various symbols and the text 'The High Velocity Web Framework For Java and Scala'. Below this, there are two main content areas: one for downloading the latest package (2.3.8) and another for getting started with tutorials. A video introduction to the Play Framework is also featured, with buttons for Scala and Java. At the bottom, a bold statement reads: 'Play Framework makes it easy to build web applications with Java & Scala.'

play

Download Documentation Get Involved

The High Velocity Web Framework For Java and Scala

GET THE LATEST PACKAGE

Download 2.3.8

or [browse all versions](#)

GET STARTED WITH

Tutorials

or [read full documentation](#)

Video introduction to Play Framework

SCALA JAVA

Play Framework makes it easy to build web applications with Java & Scala.

Even More Scala



Scala.js

[Archive](#)

[Categories](#)

[Pages](#)

[Tags](#)

Scala.js the Scala to JavaScript compiler

Scala.js compiles Scala code to JavaScript, allowing you to write your web application entirely in Scala! Take a look at the [project gallery](#) to see what kind of things you can build with Scala.js.

Get started

[Start the Tutorial](#)

[Try it in the Browser](#)

The easiest way to get started is to follow our [tutorial](#). You can also fork the [bootstrapping skeleton](#) and follow the instructions in its readme or [try it out in the browser](#). There's also an e-book [Hands-on Scala.js](#) which contains a lot of introductory material to help you get started.

We also have a [standalone distribution](#) that doesn't require SBT.

Note that Scala.js is not part of the Typesafe Reactive platform. Thus, although we consider Scala.js production-ready, Typesafe does not provide any commercial support for it.



Scala.js logo

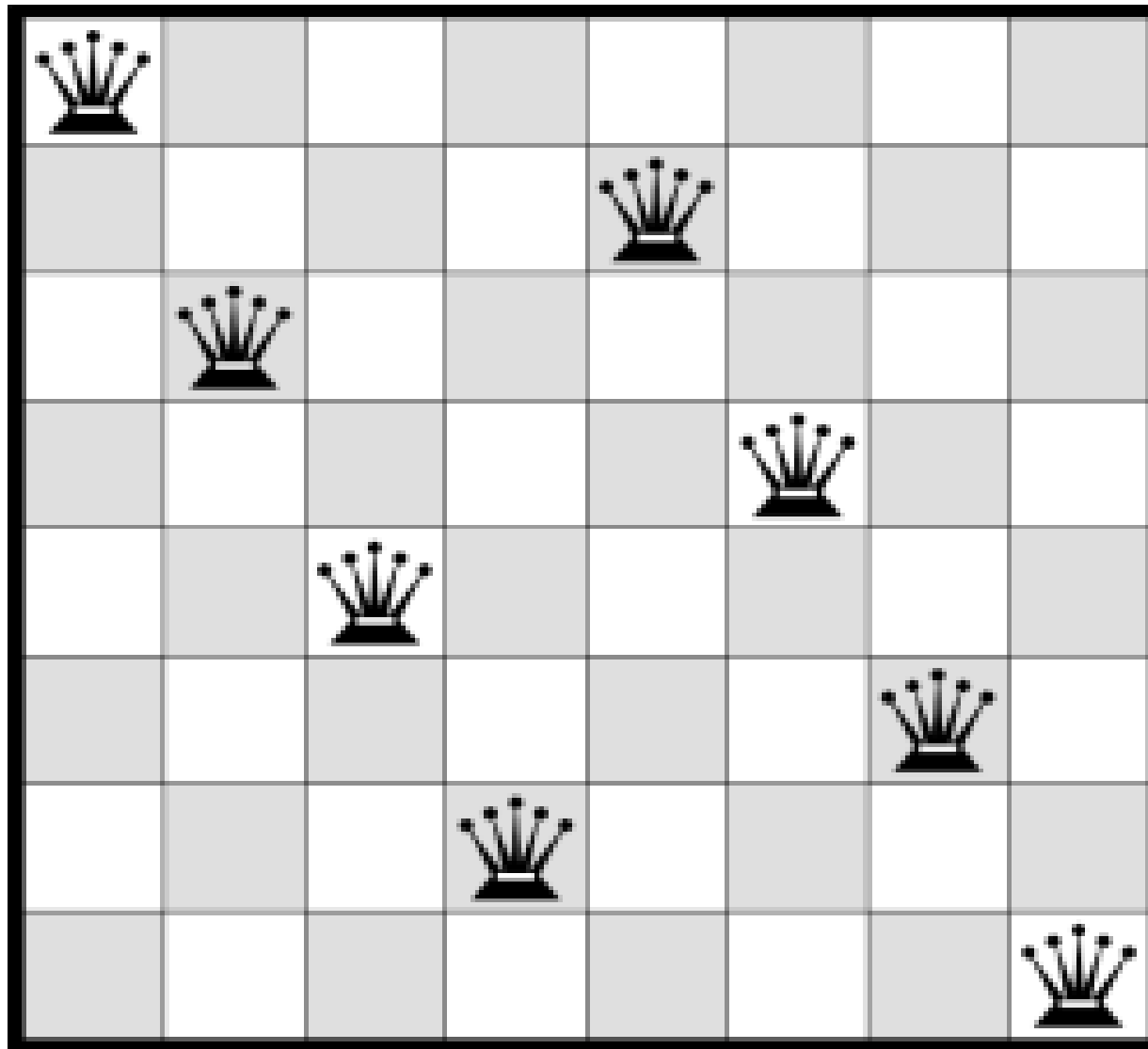
Noteworthy features

- Support all of Scala (including macros!), modulo [a few semantic differences](#)
- Very good [interoperability with JavaScript code](#). For example, use jQuery and HTML5 from your Scala.js code, either in a typed or untyped way. Or create Scala.js objects and call their methods from JavaScript.
- [Integrated with sbt](#) (including support for dependency management and incremental compilation)
- Can be used with your favorite IDE for Scala
- Generates [Source Maps](#) for a smooth debugging experience (step through your Scala code from within your browser supporting source maps)
- Integrates [Google Closure Compiler](#) for producing minimal code for production. Compiled blobs range from 170-400kb
- Produces (very) efficient JavaScript code ([benchmarks](#))

Thank You

Questions ?

8 Queens



8 Queens

// Example: List(4, 2, 7, 3, 6, 8, 5, 1) means that the queen in the first column is in row 4,
// the queen in the second column is in row 2, etc

```
object Queens {  
  def eightQueens = {  
    def validDiagonals(qs: List[Int], upper: Int, lower: Int): Boolean = qs match {  
      case Nil => true  
      case q :: tail => q != upper && q != lower && validDiagonals(tail, upper + 1, lower - 1)  
    }  
    def valid(qs: List[Int]): Boolean = qs match {  
      case Nil => true  
      case q :: tail => validDiagonals(tail, q + 1, q - 1)  
    }  
    def eightQueensR(curQueens: List[Int], remainingCols: Set[Int]): List[List[Int]] =  
      if (!valid(curQueens)) Nil  
      else if (remainingCols.isEmpty) List(curQueens)  
      else remainingCols.toList.flatMap(c => eightQueensR(c :: curQueens, remainingCols - c))  
    eightQueensR(Nil, Set() ++ (1 to 8))  
  }  
}
```