Scala Clinic

Scala basics...

Jim Powers

Patch.com

8 February 2012

What is Scala?

What is Scala?

Scala is a multi-paradigm programming language designed to integrate features of object-oriented programming and functional programming. The name Scala is a portmanteau of *scalable* and *language*, signifying that it is designed to grow with the demands of its users. James Strachan, the creator of Groovy, described Scala as a possible successor to Java.

http://en.wikipedia.org/wiki/Scala_(programming_language)

Why Scala?

- It's not Java?

Why Scala?

- It's not Java?
- Inter-operates with Java eco-system well

Why Scala?

- It's not Java?
- Inter-operates with Java eco-system well
- Many features to aid building large systems

Why Scala?

- It's not Java?
- Inter-operates with Java eco-system well
- Many features to aid building large systems
- Good performance

Why Scala?

- It's not Java?
- Inter-operates with Java eco-system well
- Many features to aid building large systems
- Good performance
- Probably the best language on the JVM

Why Scala?

- It's not Java?
- Inter-operates with Java eco-system well
- Many features to aid building large systems
- Good performance
- Probably the best language on the JVM
- Robust and active community

Why Scala?

- It's not Java?
- Inter-operates with Java eco-system well
- Many features to aid building large systems
- Good performance
- Probably the best language on the JVM
- Robust and active community
- Getting better all the time

Why Scala?

- It's not Java?
- Inter-operates with Java eco-system well
- Many features to aid building large systems
- Good performance
- Probably the best language on the JVM
- Robust and active community
- Getting better all the time
    - Scala 2.10 to be released this quarter

Why JVM?

- Tons of stuff written for it (some of it good)

Why JVM?

- Tons of stuff written for it (some of it good)
- Fewer surprises running code in different environments

Why JVM?

- Tons of stuff written for it (some of it good)
- Fewer surprises running code in different environments
- Good performance (thousands of man-years)

My Opinions on Scala

- It's mostly nice, kinda. - I'm not a rabid fanboi

My Opinions on Scala

- It's mostly nice, kinda. - I'm not a rabid fanboi
- Better than Java

My Opinions on Scala

- It's mostly nice, kinda. - I'm not a rabid fanboi
- Better than Java
- Better than most alternatives on the JVM

My Opinions on Scala

- It's mostly nice, kinda. - I'm not a rabid fanboi
- Better than Java
- Better than most alternatives on the JVM
- Good run-time performance

My Opinions on Scala

- It's mostly nice, kinda. - I'm not a rabid fanboi
- Better than Java
- Better than most alternatives on the JVM
- Good run-time performance
- Smart people working on the language and tools

My Opinions on Scala

- It's mostly nice, kinda. - I'm not a rabid fanboi
- Better than Java
- Better than most alternatives on the JVM
- Good run-time performance
- Smart people working on the language and tools
- Compiler is slow

My Opinions on Scala

- It's mostly nice, kinda. - I'm not a rabid fanboi
- Better than Java
- Better than most alternatives on the JVM
- Good run-time performance
- Smart people working on the language and tools
- Compiler is slow
- Will be happier when macros land

Getting Started

- Get Java JDK 1.6 (1.7 if you want to experiment)

Getting Started

- Get Java JDK 1.6 (1.7 if you want to experiment)
- Download it (2.9.1 current) `http://scala-lang.org`

Getting Started

- Get Java JDK 1.6 (1.7 if you want to experiment)
- Download it (2.9.1 current) http://scala-lang.org
- Install SBT https://github.com/harrah/xsbt

Getting Started

- Get Java JDK 1.6 (1.7 if you want to experiment)
- Download it (2.9.1 current) http://scala-lang.org
- Install SBT https://github.com/harrah/xsbt
- TextMate/Sublime Text/Vim/Emacs/JEdit

Getting Started

- Get Java JDK 1.6 (1.7 if you want to experiment)
- Download it (2.9.1 current) http://scala-lang.org
- Install SBT https://github.com/harrah/xsbt
- TextMate/Sublime Text/Vim/Emacs/JEdit
- Eclipse http://scala-ide.org/index.html

Getting Started

- Get Java JDK 1.6 (1.7 if you want to experiment)
- Download it (2.9.1 current) http://scala-lang.org
- Install SBT https://github.com/harrah/xsbt
- TextMate/Sublime Text/Vim/Emacs/JEdit
- Eclipse http://scala-ide.org/index.html
- Intelli/J IDEA

  http://confluence.jetbrains.net/display/SCA/Scala+Plugin+Nightly+Builds+for+Nika

Getting Started

- Get Java JDK 1.6 (1.7 if you want to experiment)
- Download it (2.9.1 current) http://scala-lang.org
- Install SBT https://github.com/harrah/xsbt
- TextMate/Sublime Text/Vim/Emacs/JEdit
- Eclipse http://scala-ide.org/index.html
- Intelli/J IDEA

  http://confluence.jetbrains.net/display/SCA/Scala+Plugin+Nightly+Builds+for+Nika
- Plays well with most JVM tools

On-Line Learning Resources

- Scala Documentation Project http://docs.scala-lang.org/

On-Line Learning Resources

- Scala Documentation Project http://docs.scala-lang.org/
- Official Scala Web Site http://scala-lang.org

On-Line Learning Resources

- Scala Documentation Project http://docs.scala-lang.org/
- Official Scala Web Site http://scala-lang.org
- Scala-user Google Group

  https://groups.google.com/forum/?fromgroups#!forum/scala-user

On-Line Learning Resources

- Scala Documentation Project http://docs.scala-lang.org/
- Official Scala Web Site http://scala-lang.org
- Scala-user Google Group

  https://groups.google.com/forum/?fromgroups#!forum/scala-user
- Stackoverflow http://stackoverflow.com/tags/scala/info

On-Line Learning Resources

- Scala Documentation Project http://docs.scala-lang.org/
- Official Scala Web Site http://scala-lang.org
- Scala-user Google Group

  https://groups.google.com/forum/?fromgroups#!forum/scala-user
- Stackoverflow http://stackoverflow.com/tags/scala/info
- Scala Koans https://bitbucket.org/dickwall/scala-koans

On-Line Learning Resources

- Scala Documentation Project http://docs.scala-lang.org/
- Official Scala Web Site http://scala-lang.org
- Scala-user Google Group

  https://groups.google.com/forum/?fromgroups#!forum/scala-user
- Stackoverflow http://stackoverflow.com/tags/scala/info
- Scala Koans https://bitbucket.org/dickwall/scala-koans
- Debasish Ghosh's Blog http://debasishg.blogspot.com/

On-Line Learning Resources

- Scala Documentation Project http://docs.scala-lang.org/
- Official Scala Web Site http://scala-lang.org
- Scala-user Google Group

  https://groups.google.com/forum/?fromgroups#!forum/scala-user
- Stackoverflow http://stackoverflow.com/tags/scala/info
- Scala Koans https://bitbucket.org/dickwall/scala-koans
- Debasish Ghosh's Blog http://debasishg.blogspot.com/
- Tony Morris' Blog http://blog.tmorris.net/

On-Line Learning Resources

- Scala Documentation Project http://docs.scala-lang.org/
- Official Scala Web Site http://scala-lang.org
- Scala-user Google Group

  https://groups.google.com/forum/?fromgroups#!forum/scala-user
- Stackoverflow http://stackoverflow.com/tags/scala/info
- Scala Koans https://bitbucket.org/dickwall/scala-koans
- Debasish Ghosh's Blog http://debasishg.blogspot.com/
- Tony Morris' Blog http://blog.tmorris.net/
- Jim McBeath's Blog http://jim-mcbeath.blogspot.com/

Books

- A pretty comprehensive list can be found here

  http://www.scala-lang.org/node/959

Books

- A pretty comprehensive list can be found here

  http://www.scala-lang.org/node/959

- Martin Odersky's Book

  http://www.artima.com/shop/programming_in_scala_2ed

Books

- A pretty comprehensive list can be found here

  http://www.scala-lang.org/node/959
- Martin Odersky's Book

  http://www.artima.com/shop/programming_in_scala_2ed
- Josh Seureth's Book http://www.manning.com/suereth/

Community

- Scala Bug-tracker (JIRA)

  https://issues.scala-lang.org/secure/Dashboard.jspa

Community

- Scala Bug-tracker (JIRA)

  `https://issues.scala-lang.org/secure/Dashboard.jspa`
- Google Groups (there are a bunch)

Community

- Scala Bug-tracker (JIRA)

  https://issues.scala-lang.org/secure/Dashboard.jspa
- Google Groups (there are a bunch)
- Scala Documentation Project http://docs.scala-lang.org/

Community

- Scala Bug-tracker (JIRA)

  https://issues.scala-lang.org/secure/Dashboard.jspa
- Google Groups (there are a bunch)
- Scala Documentation Project http://docs.scala-lang.org/
- Scala Improvemet Process

  http://docs.scala-lang.org/sips/index.html

Community

- Scala Bug-tracker (JIRA)

  https://issues.scala-lang.org/secure/Dashboard.jspa
- Google Groups (there are a bunch)
- Scala Documentation Project http://docs.scala-lang.org/
- Scala Improvemet Process

  http://docs.scala-lang.org/sips/index.html
- IRC: #scala on freenode

Community

- Scala Bug-tracker (JIRA)

  https://issues.scala-lang.org/secure/Dashboard.jspa
- Google Groups (there are a bunch)
- Scala Documentation Project http://docs.scala-lang.org/
- Scala Improvemet Process

  http://docs.scala-lang.org/sips/index.html
- IRC: #scala on freenode
- Meetups/Hackathons

Frame of Mind

It is a logical impossibility to make a language more powerful by omitting features, no matter how bad they may be.

– John Huges, *Why Functional Programming Matters*, 1990

Frame of Mind

Scala seems designed on the principle that if we can't have nice things, we can at least have lots and lots of meh ones.

– Bryan O'Sullivan, `https://twitter.com/#!/bos31337/status/155102828774428672`

Frame of Mind

The more interesting your types are, the less fun they are to write down.

– Benjamin Pierce

Language Features

- Strongly Typed
  - Statically typed

Language Features

- Strongly Typed
  - Statically typed
  - Local type inference

```
def foo(x:Int) = List(x) // ->> foo(x:Int):List[Int]
val x = foo(9) // -> x:List[Int]
```

Language Features

- Strongly Typed
  - Statically typed
  - Local type inference

```
def foo(x:Int) = List(x) // ->> foo(x:Int):List[Int]
val x = foo(9) // -> x:List[Int]
```

  - Parameterized types (generics)

```
class Foo[X,Y]
trait Bar[A,M[_]] // ->> 'M' is a type constructor
```

Language Features

- Strongly Typed
  - Statically typed
  - Local type inference

```
def foo(x:Int) = List(x) // ->> foo(x:Int):List[Int]
val x = foo(9) // -> x:List[Int]
```

  - Parameterized types (generics)

```
class Foo[X,Y]
trait Bar[A,M[_]] // ->> 'M' is a type constructor
```

  - Type aliases

```
type MyFoo = Foo[Int,Double]
type MyFooX[X] = Foo[X,Double]
type MyFooXY[X,Y] = Foo[X,Y]
```

Language Features

- Strongly Typed
  - Statically typed
  - Local type inference

```
def foo(x:Int) = List(x) // ->> foo(x:Int):List[Int]
val x = foo(9) // -> x:List[Int]
```

  - Parameterized types (generics)

```
class Foo[X,Y]
trait Bar[A,M[_]] // ->> 'M' is a type constructor
```

  - Type aliases

```
type MyFoo = Foo[Int,Double]
type MyFooX[X] = Foo[X,Double]
type MyFooXY[X,Y] = Foo[X,Y]
```

  - "Higher-kinded" Types

Language Features

- Strongly Typed
  - Statically typed
  - Local type inference

```
def foo(x:Int) = List(x) // ->> foo(x:Int):List[Int]
val x = foo(9) // -> x:List[Int]
```

  - Parameterized types (generics)

```
class Foo[X,Y]
trait Bar[A,M[_]] // ->> 'M' is a type constructor
```

  - Type aliases

```
type MyFoo = Foo[Int,Double]
type MyFooX[X] = Foo[X,Double]
type MyFooXY[X,Y] = Foo[X,Y]
```

  - "Higher-kinded" Types
  - Specialized Generics

Language Features

- Common Types
  - Primatives: `Byte`, `Int`, `Long`, `Short`, `Double`, `Float`, `Char`, `Boolean`

Language Features

- Common Types
  - Primatives: `Byte`, `Int`, `Long`, `Short`, `Double`, `Float`, `Char`, `Boolean`
  - Arrays: `Array[_](size)`

Language Features

- Common Types
  - Primatives: `Byte`, `Int`, `Long`, `Short`, `Double`, `Float`, `Char`, `Boolean`
  - Arrays: `Array[_](size)`
  - Strings

Language Features

- Common Types
  - Primatives: `Byte`, `Int`, `Long`, `Short`, `Double`, `Float`, `Char`, `Boolean`
  - Arrays: `Array[_](size)`
  - Strings
  - Functions

```
// Functions are represented with type T1 => T2
val f = (_:Int).toString // f:Int => String
```

Language Features

- Common Types
  - Primatives: `Byte`, `Int`, `Long`, `Short`, `Double`, `Float`, `Char`, `Boolean`
  - Arrays: `Array[_](size)`
  - Strings
  - Functions

```
// Functions are represented with type T1 => T2
val f = (_:Int).toString // f:Int => String
```

  - List

```
val x = List(1,2,3) // x:List[Int]
val y = List(1,"2",(3,4)) // y:List[Any]
val z = 1::2::3::Nil // z:List[Int]

// Pattern matching
val w = x match {
  case Nil => -1
  case h::t => h // h:Int, t:List[Int]
}
```

Language Features

- Common Types
  - Vector: `Vector[_]`(v_0,v_1,v_3,...)

Language Features

- Common Types
  - Vector: `Vector[_]`(v_0,v_1,v_3,...)
  - Map: `Map[_,_]`(k_0 `->` v_0,k_1 `->` v_1,k_3 `->` v_3,...)

Language Features

- Common Types
  - Vector: `Vector[_]`(v_0,v_1,v_3,...)
  - Map: `Map[_,_]`(k_0 `->` v_0,k_1 `->` v_1,k_3 `->` v_3,...)
  - Set: `Set[_]`(k_0,k_1,k_3,...)

Language Features

- Common Types
  - Vector: `Vector[_]`(v_0,v_1,v_3,...)
  - Map: `Map[_,_]`(k_0 -> v_0,k_1 -> v_1,k_3 -> v_3,...)
  - Set: `Set[_]`(k_0,k_1,k_3,...)
  - Stream

```
def from(n: Int): Stream[Int] =
   Stream.cons(n, from(n + 1))
def sieve(s: Stream[Int]): Stream[Int] =
  Stream.cons(s.head, sieve(s.tail filter { _ % s.head != 0 }))
def primes = sieve(from(2))

primes take 10 print // -> 2, 3, 5, 7, 11, 13, 17, 19, 23, 29
```

Language Features

- Common Types
  - Vector: `Vector[_]`(v_0,v_1,v_3,...)
  - Map: `Map[_,_]`(k_0 -> v_0,k_1 -> v_1,k_3 -> v_3,...)
  - Set: `Set[_]`(k_0,k_1,k_3,...)
  - Stream

```
def from(n: Int): Stream[Int] =
   Stream.cons(n, from(n + 1))
def sieve(s: Stream[Int]): Stream[Int] =
  Stream.cons(s.head, sieve(s.tail filter { _ % s.head != 0 }))
def primes = sieve(from(2))

primes take 10 print // -> 2, 3, 5, 7, 11, 13, 17, 19, 23, 29
```

  - Unit: ()

Language Features

- Common Types
  - Vector: `Vector[_](`v_0`,`v_1`,`v_3`,...)`
  - Map: `Map[_,_](`k_0` -> `v_0`,`k_1` -> `v_1`,`k_3` -> `v_3`,...)`
  - Set: `Set[_](`k_0`,`k_1`,`k_3`,...)`
  - Stream

```
def from(n: Int): Stream[Int] =
   Stream.cons(n, from(n + 1))
def sieve(s: Stream[Int]): Stream[Int] =
  Stream.cons(s.head, sieve(s.tail filter { _ % s.head != 0 }))
def primes = sieve(from(2))

primes take 10 print // -> 2, 3, 5, 7, 11, 13, 17, 19, 23, 29
```

  - Unit: ()
  - Tuples *a maximum size of 22*

```
val x = (1,"a") // x:Tuple2[Int,String]
val y:(Int,String) = (2,"b")
```

Language Features

- Common Types
  - Option

```scala
val x = Some(1) // x:Some[Int]
val y:Option[Int] = None

def foo(t:Option[Int]):Unit = t match {
  case None => println("got nothing")
  case Some(_) => println("got something")
}

foo(x) // ->> got something
foo(y) // ->> got nothing
```

Language Features

- Common Types
  - Either

```scala
val x = Left[Int,String](1) // x:Left[Int,String]
val y = Right[Int,String]("hello") // y:Right[Int,String]
val z:Either[Int,String] = Right("ZZZZ")

def foo(t:Either[Int,String]):Unit = t match {
    case Left(x) => println("Left(%d)".format(x))
    case Right(x) => println("Right(%s)".format(x))
  }

foo(x) // ->> Left(1)
foo(y) // ->> Right(hello)
foo(z) // ->> Right(ZZZZ)
```

Language Features: Object-Oriented

Classes and Traits

Example: *Simple*

```
class Foo
trait Bar
```

Language Features: Object-Oriented

Classes and Traits

Example: *Single inheritance*

```scala
// Any number of traits
trait Bar
trait Baz
class Foo extends Bar with Baz

// Only one class, but with any number of traits
trait Bad
class Woo extends Foo with Bad

// Traits can inherit from other traits
trait Car extends Bar with Baz with Bad
```

Language Features: Object-Oriented

Classes and Traits

Traits do not have constructors

Language Features: Object-Oriented

Classes and Traits

Example: *Classes have canonical constructor*

```
// Trivial
class Foo
val x = new Foo() // Does nothing, but works

// Define a private value in the constructor
class Foo(x:Int) // Type annoation on arguments *always* required
val x = new Foo(8) // The '8' is trapped inside a Foo, never to escape

// Define a read-only value as an argument
class Foo(val x:Int)
val x = new Foo(8)
x.x // ->> returns 8
x.x = 9 // ->> Will not compile: cannot set a 'val'

// Define a read/write value
class Foo(var x:Int)
val x = new Foo(8)
x.x // ->> returns 8
x.x = 9 // ->> Works! x.x == 9
```

Language Features: Object-Oriented

Classes and Traits

Example: *Classes have canonical constructor*

```
// More general example
class Foo(x:Int,y:Int) { // Defines the beginning of the
  private val s = x + y  // constructor body
  def sum = s
}
class Bar(x:Int,y:Int) extends Foo(x,y) {
  private val d = x - y
  def diff = d
}
```

Language Features: Object-Oriented

Classes and Traits

Example: *Classes can have alternate constructors*

```
import java.lang.{Integer => JInt}
class Foo(x:Int,y:Int) {
  def this(xs:String,ys:String) = // alternate constructor
    this(JInt.parse(xs.trim),JInt.parse(ys.trim))
  private val s = x + y
  def sum = s
}
```

Language Features: Object-Oriented

Classes and Traits

Example: *Classes and Traits can be nested*

```
class Foo(x:Int,y:Int) {
  class Bar {
    val d = x - y
    def diff = d
  }
  private val s = x + y
  def sum = s
}

// Usage
val f = new Foo(1,2)
f.sum // ->> 3
val g = new f.Bar
g.diff // ->> -1
```

Language Features: Object-Oriented

Classes and Traits

Example: *Classes and Traits can be nested*

```
trait Baz {
  val x:Int; val y:Int // Abstract values
  trait Bad {
    def product = x * y
  }
  def diff = x - y
}

class Foo(val x:Int,val y:Int) extends Baz {
  def bad = new Bad {} // type Baz#Bad
  def sum = x + y
}

// Usage
val f = new Foo(1,2)
val b = f.bad
b.product // ->> 2
```

Language Features: Object-Oriented

Classes and Traits

Example: *Traits can carry implementations*

```
trait Bar {
  def foo:Int // Abstract member
  val bar = "bar"
  var goo = foo + bar.length
}
```

Language Features: Object-Oriented

Classes and Traits

Example: *Mixins*

```
trait Foo { this:Bar => // 'self' annotation with type
  override def bar(x:Int) = x + 2
}

class Bar {
  def bar(x:Int) = x + 1
}

val b = new Bar with Foo
b.bar(10) // ->> 12
```

The principle hack used to implement the *Cake Pattern*

Language Features: Object-Oriented

Classes and Traits

Example: *Ad-hoc Enrichment*

```
class Foo {
  private val x:Long = 10
  def getX = x
}

def genFoo(flip:Boolean) =
  if (flip) new Foo
  else new Foo {
    private val x:Long = System.nanoTime
    override def getX = x
  }

val f1 = genFoo(true)
val f2 = genFoo(false)

f1.getX // ->> 10
f2.getX // ->> Example: 1268696837761878
```

Language Features: Object-Oriented

Structural Types

Example: *Structural Types*

```
def foo(x:{val bar:Int}) = 5 + x.bar

class Bar {
  val bar = 5
}

class Bad {
  val bar = 9
}

class Baz {
  val baz = 9
}

foo(new Bar())
foo(new Bad())
foo(new Baz()) // << Does not compile
```

Language Features: Object-Oriented

Anonymous Classes

Example: *Anonymous Classes*

```
val f = new {
    val x = 5
  }

f.x // ->> 5
```

Language Features: Object-Oriented

Self-naming

Example: *Self-naming*

```
// Self-naming class
class Foo(val x:Int) { self=> // Now 'this' is called 'self'
  def dup = new Foo(self.x)
}

// Self-naming trait
trait Bar { self=> // Now 'this' is called 'self'
  val x:Int
  def dup = new Bar {
    val x:Int = self.x
  }
}
```

Language Features: Object-Oriented

Traits can Self-type

Example: *Self-typing*

```
class Foo(val x:Int)

trait Bar { self:Foo=> // Bar can only be mixed
  def bar = self.x+5    // into a subclass of Foo
}

class Baz(x:Int) extends Foo(x) with Bar

val b = new Baz(9)
b.bar // ->> 14
```

Why would you do this? – self-types constrain inheritance without exposing an is-a relationship.

Language Features: Object-Oriented

First-class Modules

Example: *Classes as Modules*

```
class Foo[T](val x:T, show:T => String) {
  type TheType = T
  def foo:String = show(x)
}

object FooTest {
  def main(args:Array[String]) {
    val f = new Foo(50,(_:Int).toString)
    val g:f.TheType = 100
    import f._
    println(foo) // ->> "50"
  }
}
```

Language Features: Object-Oriented

First-class Modules

Example: *Objects as Modules*

```
object Foo { // Declares a singleton object
  def myPrintln(x: Any) {
    print("MY: ")
    println(x)
  }
}

object Bar {
  import Foo._
  def printTest {
    myPrintln("This is a test")
    Foo.myPrintln("This is a test1")
  }
}
```

Language Features: Object-Oriented

First-class Modules

Example: *Package Objects*

```scala
// File: package.scala
package foo

package object bar {
  def printBar = println("bar")
}

// File: Foo.scala
package foo.bar

class Foo {
  def foo = printBar
}
```

Language Features: Object-Oriented

First-class Modules

Example: *Visibility Control*

```
package my.foo

class Foo {
  private val x = 5 // Only visible inside the class
  protected def foo = "foo" // visible to sub-types
  def bar = foo // Public
  final val tt = 99 // cannot be over-ridden
  private[foo] pFoo = new Integer(45) // private outside the module my.foo
}
```

Language Features: Object-Oriented

First-class Modules

Example: *Imports*

```
// simple symbol import
import java.io.File
// wildcard import
import java.io._
// multiple imports from same parent namespace
import java.io.{File,FileInputStream}
// nested imports
import java.io.File, File._
// rename symbol
import java.io.{File => JFile}
```

Language Features: Object-Oriented

Simulated "Algebraic Data Types"

Example:

```scala
// This can also be a class
sealed trait Foo
case class IntFoo(x:Int) extends Foo
case class DoubleFoo(x:Double) extends Foo
case object DeadFoo extends Foo

val x:Foo = IntFoo(4) // Notice no 'new' here

val y = x match {
    case IntFoo(_) => "got int"
    case DoubleFoo(_) => "got double"
    case DeadFoo => "He's dead, Jim."
  }

// Compiler will generate a warning that the match is not exhaustive
val z = x match {
    case IntFoo(_) => "got int"
  }

// Compiler happy now
val w = x match {
    case IntFoo(_) => "got int"
    case _ => "don't care"
  }
```

Language Features: Object-Oriented

Simulated “Algebraic Data Types”

Example: *Nifty Trick*

```
// Nifty trick to subset matchers
sealed trait Foo
sealed trait Bar
case class IntFoo(x:Int) extends Foo
case class DoubleFoo(x:Double) extends Foo with Bar
case object DeadFoo extends Foo with Bar

val x:Bar = DeadFoo // Notice no 'new' here

val y = x match {
    case DoubleFoo(_) => "got double"
    case DeadFoo => "He's dead, Jim."
  }
```

Language Features: Object-Oriented

Case Classes

Example: *Copy Synthetic*

```
case class Name(first:String,middle:Option[String],last:String)

// Name(Thomas,Some(Alva),Edison)
val n1 = Name("Thomas",Some("Alva"),"Edison")
// Name(Thomas,Some(Tupac),Edison)
val n2 = n1.copy(middle = Some("Tupac"))
```

Language Features: Functional

First-class functions!

Example:

```scala
// Anonymous Functions
val f:Int => String = (x => x.toString)
val f:Int => String = {x => x.toString}
val f:Int => String = {
  x => x.toString
}

// Using the all powerful _
val g:Int => String = _.toString

// Using the all powerful _ blech type-inferencing
val h = (_:Int).toString

// Closures
def foo(x:Int):Int => Int = (_ + x)

// Nested
def fact(n:Int):Int = {
  def factN(n:Int,out:Int):Int =
    if (n <= 1) out
    else factN(n-1,out*n)
  factN(n,1)
}
```

Language Features: Functional

Function “Objects”

Example:

```
// You can use singleton objects
object Foo {
  def apply(x:Int) = x.toString
}

// Or instances
class Bar {
  def apply(x:Int) = x.toString
}

val b = new Bar
b(5) // ->> "5"
```

Language Features: Functional

Partial Functions

Example:

```
val f1:PartialFunction[Option[String],String] = {
  case Some("George") => "you're somebody"
  case Some("Sam") => "you're somebody"
  case Some("Desmond") => "you're somebody"
}

f1(Some("George")) // ->> "you're somebody"
f1(Some("Tom")) // ->> Exception!
```

Language Features: Functional

Partial Functions

Example: *Partial function chaining*

```
val f1:PartialFunction[Option[String],String] = {
  case Some("George") => "you're somebody"
  case Some("Sam") => "you're somebody"
  case Some("Desmond") => "you're somebody"
}
val f2:PartialFunction[Option[String],String] = {
  case Some("Tom") => "you're Tom"
  case Some(_) => "you're nobody"
  case None => "you're not even there"
}
val f3 = f1 orElse f2
f3(Some("George")) // ->> "you're somebody"
f3(Some("Tom")) // ->> "you're Tom"
f3(None) // "you're not even there"
```

Language Features: Functional

Partial Functions

Example: *Guarded*

```
val coolNames = Set("george","sam","desmond")

val f1:PartialFunction[Option[String],String] = {
  case Some(x) if coolNames(x.toLowerCase) => "you're somebody"
}

f1(Some("George")) // ->> "you're somebody"
f1(Some("Tom")) // ->> Exception!
```

Language Features: Functional

Parameter Groups

Example:

```
def time[T](message:String)(block: => T):T = {
  val start = System.nanoTime
  val result = block
  val end = System.nanoTime
  println("%s: %d".format(message,end-start))
  result
}

// Notice that the second argument is a block
time("Testing Thread.sleep") {
  Thread.sleep(1000)
}
```

Language Features: Functional

By Reference, Value and Name arguments

Example:

```
def foo(x:Int, // Int's are value types
        y:String, // Strings are reference types
        z: => String):String = // Something that eventally will result in a string
  if (x < 0) y
  else z // Exression only forced if this branch evaluated
```

Language Features: Functional

Variable-sized Argument Lists

Example:

```
// 0 or more indicated with a '*'
def foo(prefix:String,args:Int*):Unit =
  println("%s: %s".format(prefix,List(args:_*))) // "splat" with ':_*'
```

Language Features: Functional

Methods and Functions

Example: *Methods*

```
object Namer {
  // Default arguments
  def fullName(first:String, last:String, middle:Option[String] =
None):String =
    middle map (m => "%s %s %s".format(first,m,last)) getOrElse "%s
%s".format(first,last)
  // Variable arguments
  def spaniardName(first:String,rest:String*):String =
    "%s, %s".format(first,rest.mkString(", "))
}
fullName("Jim","Powers") // ->> "Jim Powers"
fullName("Thomas","Edison",Some("Alva")) // ->> "Thomas Alva Edison"
// Named parameters
fullName(last = "Einstein",first = "Albert") // ->> "Albert Einstein"
```

Language Features: Functional

Methods and Functions

Example: *Functions*

```
object Namer {
  // Default arguments
  def fullName(first:String, last:String, middle:Option[String] =
None):String =
    middle map (m => "%s %s %s".format(first,m,last)) getOrElse "%s
%s".format(first,last)
  // Variable arguments
  def spaniardName(first:String,rest:String*):String =
    "%s, %s".format(first,rest.mkString(", "))
}
```

```
import Namer._
// <method> _ -> function
// fn: (String, String, Option[String]) => String
val fn = fullName _
// sn: (String, String*) => String
val sn = spaniardName _
```

Language Features: Other

Var and Val

Example:

```
val x = new StringBuilder()
x = new StringBuilder() // ->> Won't compile
x.append("updated")
x.toString // ->> "updated"
var y = new StringBuilder()
y = new StringBuilder() // ->> Works fine!
y.append("updated")
y.toString // ->> "updated"
```

Language Features: Other

Lazy Values

Example:

```
import java.util.Date

class Foo {
  val dt1 = new Date
  lazy val dt2 = new Date
  def dumpDates {
    println("dt1: %s".format(dt1))
    println("dt2: %s".format(dt2))
  }
}

val f = new Foo
// Wait a few seconds
f.dumpDates
/** Example output:
 *     dt1: Tue Feb 07 22:48:31 EST 2012
 *     dt2: Tue Feb 07 22:48:39 EST 2012
 */
```

Language Features: Other

If expressions

Example:

```
// If expressions yield a value
val x = if (fuBar == 9) "nine"
        else "not nine"

// Chained if elses
val y = if (fuBar == 1) "one"
        else if (fuBar == 2) "two"
        else "something else"

// imperative if
if (fuBar == 10) println("fuBar is equal to ten")

// Multi-line expressions are bracketed
val z = if (fuBar == 0) {
          val r = rand.next()
          r.toString
        } else "non-zero"
```

Language Features: Other

While loops

Example:

```
// Simple while
var i = 0
while (i < 100) {
  println(i)
  i += 1
}

// do-while
var i = 0
do {
  println(i)
  i += 1
} while (i < 100)
```

Language Features: Other

For-comprehensions

Example: *Monadic For*

```
// Simple
for (i <- 1 to 100) yield i+1
// translates to
Range.inclusive(1,100).map(i => i+1)
// Guarded
for (i <- 1 to 100 if i%2==0) yield i+1
// translates to
Range.inclusive(1,100).withFilter(i => i%2 == 0).map(i => i+1)
// Nested
for (i <- 1 to 100; j <- i to 100) yield i+j
// translates to
Range.inclusive(1,100).flatMap(i => Range.inclusive(i,100).map(j =>
i+j))
```

Language Features: Other

For-comprehensions

Example: *Imperative For*

```
for { i <- 0 to 100
      j <- i to 100 } {
      println("i+j = %d".format(i+j))
    }

// Translates to
Range.inclusive(1,100).foreach { i =>
  Range.inclusive(i,100).foreach { j =>
    println("i+j = %d".format(i+j))
  }
}
```

Language Features: Other

Pattern-matching

Example: *Value Matching*

```
// use match for simple values
val x = 10

val y = x match {
  case 0 => "zero"
  case 1 => "one"
  case 2 => "two"
  case 3 => "three"
  case 4 => "four"
  case 5 => "five"
  case 6 => "six"
  case 7 => "seven"
  case 8 => "eight"
  case 9 => "nine"
  case 10 => "ten"
}
```

Language Features: Other

Pattern-matching

Example: *"Otherwise" Match*

```
// "otherwise" simple values
val x = 10
val y = x match {
  case 0 => "zero"
  case 1 => "one"
  case 2 => "two"
  case 3 => "three"
  case 4 => "four"
  case 5 => "five"
  case _ => "something other than [0-5]"
}
```

Language Features: Other

Pattern-matching

Example: *Alternates Match*

```
// Alternates allowed
val x = 10

val y = x match {
  case 0 => "none"
  case 1 => "single"
  case 2 => "couple"
  case 3 => "few"
  case 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 => "several"
  case 12 => "dozen"
  case _ => "many"
}
```

Language Features: Other

Pattern-matching

Example: *Extractor Match*

```
// Extractor match
val x:Option[Int] = Some(10)

val y = x match {
  case None => "none"
  case Some(z) => z.toString
}
```

Language Features: Other

Pattern-matching

Example: *Mixed Extractor/Value Match*

```
// Mixed Extractor/Value match
val x:Option[Int] = Some(10)

val y = x match {
  case null => "Nulls are *so* Java."
  case None => "none"
  case Some(z) => z.toString
}
```

Language Features: Other

Pattern-matching

Example: *Match with Guard*

```
// Match with guard
val x:Option[Int] = Some(10)
val y = x match {
  case None => "none"
  case Some(z) if z > 5 => "something greater than five"
  case Some(z) if z <= 5 => "something less than or equal to five"
}
```

Language Features: Other

Pattern-matching

Example: *Trick for matching multiple criteria*

```
val x:Map[String,String] = // ...
val y = (x.get("first"),x.get("last")) match {
  case (None,Some(_)) | (Some(_),None) => "clearly a celebrity"
  case _ => "Normal Joe"
}
```

Language Features: Other

Pattern-matching

Example: *Type Matching*

```scala
trait Foos
class IntFoo(val x:Int) extends Foos
class DoubleFoo(val x:Double) extends Foos
class StringFoo(val x:String) extends Foos

def show(y:Foos):String = y match {
  // If you are going to check for null, you MUST do it first
  case null => "Do I have a sign saying 'Dead Null Storage'?"
  case a:IntFoo => "IntFoo(%d)".format(a.x)
  case a:DoubleFoo => "DoubleFoo(%f)".format(a.x)
  case a:StringFoo => "StringFoo(%s)".format(a.x)
  // Order matters. _ must always be last
  case _ => "Say 'What?' one more time! I dare you! I double-dare you!"
}
```

Language Features: Other

Pattern-matching

Example: *"@" Matching*

```
val x:Map[String,String] = // ...
val y = x.get("first") match {
  case None => Some("<NO NAME>")
  case x@Some(_) => x
}
```