Get Started Developing With Scala

Jason Swartz
Jason Swartz
Jason Swartz

twitter: @swartzrock
• Scala By The Bay
• Big-Data Scala
• OSCON discount: LEARNSCALELABYTHEBAY25
• scala.bythebay.io
Agenda

This Part
Setup
Scala Part 1
Break
Scala Part 2
Lunch
Let's get our environment ready
Setup

1. A terminal or other command-line shell
2. Scala
   - Homebrew: brew install scala
   - Debian: apt-get install scala
3. SBT
   - Homebrew: brew install sbt
   - Debian: apt-get install sbt
Let's Try It Out!
It's time to run

> scala
println("Hello, Scala")
Welcome to Scala version 2.11.1 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_05).
Type in expressions to have them evaluated.
Type :help for more information.

scala> println("Hello, Scala")
Hello, Scala

scala>
Okay that's enough

Setting Up
Now let's learn some Scala
Now the learning begins
Part 1

Introduction
New Stuff

The Scala Language
The Java Virtual Machine
First-Class Functions
Static Types & Generics
Immutable Data
Pattern Matching
New Stuff

The Scala Language

The Java Virtual Machine

First-Class Functions

Static Types & Generics

Immutable Data

Pattern Matching
New Stuff

The Scala Language
The Java Virtual Machine
First-Class Functions
Static Types & Generics
Immutable Data
Pattern Matching
New Stuff

The Scala Language
The Java Virtual Machine
First-Class Functions
Static Types & Generics
Immutable Data
Pattern Matching
New Stuff

The Scala Language
The Java Virtual Machine
First-Class Functions
Static Types & Generics
Immutable Data
Pattern Matching
New Stuff

The Scala Language
The Java Virtual Machine
First-Class Functions
Static Types & Generics
Immutable Data
Pattern Matching
New Stuff

The Scala Language
The Java Virtual Machine
First-Class Functions
Static Types & Generics
Immutable Data
Pattern Matching
We Have So Much To Cover Today
Java

- Compiled
- Object-Oriented
- Static Types
- Fast (eventually)
- Generic Types
- Optimizing Compiler
- First-Class Functions
- Pattern Matching
Java 1.5

- Generic Types
- Optimizing Compiler
- First-Class Functions
- Pattern Matching
- First-Class Functions
- Pattern Matching
- Immutable Data
- Awesome New Language
Java

String name = "Phil";
Java

String name = "Phil";

Scala

var name: String = "Phil";
Java

String name = "Phil";

Scala

var name: String = "Phil"
Java

String name = "Phil";

Scala

var name = "Phil"
Java

```java
final String name = "Phil";
```
Java

```java
final String name = "Phil";
```

Scala

```scala
val name: String = "Phil";
```
Java

final String name = "Phil";

Scala

val name = "Phil"
Java

```java
int doubler(int amount) {
    return amount * 2;
}
```
Java

```java
int doubler(int amount) {
    return amount * 2;
}
```

Scala

```scala
def doubler(amount: Int): Int = {
    return amount * 2
}
```
Java

```java
int doubler(int amount) {
    return amount * 2;
}
```

Scala

```scala
def doubler(amount: Int): Int = {
    amount * 2
}
```
Java

```java
int doubler(int amount) {
    return amount * 2;
}
```

Scala

```scala
def doubler(amount: Int): Int = amount * 2
```
Java

```java
int doubler(int amount) {
    return amount * 2;
}
```

Scala

```scala
def doubler(amount: Int) = amount * 2
```
public class Node {
    public String name;
    public Node(String name) { this.name = name; }
    public void String toString() {
        return name.toString();
    }
    public boolean equals(Node n) {
        return name.equals(n.name);
    }
}
Scala

case class Node(name: String)
How would you describe Scala?
Scala is Mysterious
Scala is Mysterious
Scala is a JVM language.
Scala is Type Safe
Scala is Object-Oriented
Scala is Functional Programming
(whatever *that* means)
Scala is Expressive
Scala is Expression-Oriented
Scala is Limber
Scala is built for
Speed
Scala is built for Performance
Scala is built for Productivity
Okay, that's enough about Scala
Now let's learn some Scala
Part 2

The Basics
Time to REPL with Scala
Scala REPL and SBT Shell

scala>

sbt>

O'REILLY®
oscon
Scala REPL and SBT Shell

scala

sbt console

scala>

>

scala>
Are we ready to REPL?
println("Hello, Scala")
Welcome to Scala version 2.11.1 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_05).
Type in expressions to have them evaluated. Type :help for more information.

scala> println("Hello, Scala")
Hello, Scala

scala>
"Hello, Scala"
scala> println("Hello, Scala")
Hello, Scala

scala> "Hello, Scala"
res1: String = Hello, Scala
scala> println("Hello, Scala")
Hello, Scala

scala> "Hello, Scala"
res1: String = Hello, Scala

A Handy Value Containing The Result For Later Reuse

The Type Of The Result

Your Result, In Case You Forgot
Strings

"Hello, " + "World"
"Hello, World" == "Hello, World"
"Hello, World".size
"Hello, World".replaceAll("W.*", "Java")
Multiline Strings

Hello,
World

html.replaceAll("\n\n\n", "\n\n\n")
More Types

12: Int

3.1416: Double

ture: Boolean

List(1, 3, 20): List[Int]
Lists

List(1, 3, 20): List[Int]

List('a', 'b', 'c')

List("hi", "there")

List(x: A, y: A, z: A)
Lists

List(1, 3, 20): List[Int]
List('a', 'b', 'c'): List[Char]
List("hi", "there")
List(x: A, y: A, z: A)
Lists

List(1, 3, 20): List[Int]

List('a', 'b', 'c'): List[Char]

List("hi", "there"): List[String]

List(x: A, y: A, z: A)
Lists

List(1, 3, 20): List[Int]
List('a', 'b', 'c'): List[Char]
List("hi", "there"): List[String]
Generics

defn: Code or data that takes a type as a parameter specifying a type to use at runtime

List[A] could be List[Int] or List[Char] or List[String]
Generics

defn: Code or data that takes a type as a parameter specifying a type to use at runtime

List[A] could be List[Int] or List[Char] or List[String] or List[List[Int]]
List Operations

List(1, 2) ++ 3
List Operations

List(1, 2) ++ 3
List(1, 2) ++ List(3, 4)
List Operations

List(1, 2) :+ 3
List(1, 2) ++ List(3, 4)
List(2, 4, 1, 3).sorted
## Scala Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>Base type</td>
<td>n/a</td>
</tr>
<tr>
<td>Boolean</td>
<td>Either true or false</td>
<td>false</td>
</tr>
<tr>
<td>Char</td>
<td>Unicode character</td>
<td>'A'</td>
</tr>
<tr>
<td>Double</td>
<td>8-byte floating point</td>
<td>9.23</td>
</tr>
<tr>
<td>Int</td>
<td>4-byte integer</td>
<td>10</td>
</tr>
<tr>
<td>List</td>
<td>Sequence of items</td>
<td>List(1,2,3)</td>
</tr>
<tr>
<td>Long</td>
<td>8-byte integer</td>
<td>327L</td>
</tr>
<tr>
<td>String</td>
<td>Text</td>
<td>&quot;Hello&quot;</td>
</tr>
</tbody>
</table>
Syntax: Defining A Value

val identifier[: type] = expression
Syntax: Defining A Value

val identifier[:: type] = expression

val count = 20 + 5
val eolChar = '|
val names = List("Oona", "Lily")
Syntax: Defining A Value

val identifier[:: type] = expression

val count: Int = 20 + 5
val eolChar: Char = '|'
val names: List[String] = List("Oona", "Lily")
That was Types & Values
How about we try out your new skills?
Setup (again)

1. A terminal or other command-line shell
2. Scala
   - Homebrew: brew install scala
   - Debian: apt-get install scala
3. SBT
   - Homebrew: brew install sbt
   - Debian: apt-get install sbt
It's time to try SBT
"build.sbt"

name := "My Project"

version := "1.0"

scalaVersion := "2.11.7"
"build.sbt"

name := "My Project"

version := "1.0"

scalaVersion := "2.11.7"

> sbt compile

[success]
object Hello extends App {
    println("Hello, SBT")
}
"Hello.scala"

```scala
object Hello extends App {
  println("Hello, SBT")
}
```

> sbt compile
[success]
"Hello.scala"

```scala
object Hello extends App {
  println("Hello, SBT")
}
```

> sbt compile
[success]

> sbt run
Hello, SBT
> sbt compile
[success]

> sbt run
Hello, SBT
Exercises

1. The formula for converting Centigrade to Fahrenheit is
   \[ f = c \times \frac{9}{5} + 32 \]
   In the Scala REPL, convert 22.5° Centigrade to the equivalent temperature in Fahrenheit
2. Take the Fahrenheit result from the exercise above, halve it, and convert it back to Centigrade.
3. Create a file named "HelloWorld.scala" with a single println. Run it from the Scala REPL with this command:
   :load HelloWorld.scala
Part 3

Expressions
scala> println("Hello, Scala")
Hello, Scala

scala> "Hello, Scala"
res1: String = Hello, Scala
How can that be valid code?
Expressions are valid commands
Expression

defn: A unit of code that returns a value after it has been executed.
Expression

defn: A unit of code that returns a value after it has been executed.

"Hello, Scala"
Expression

defn: A unit of code that returns a value after it has been executed.

"Hello, Scala" true
Expression

defn: A unit of code that returns a value after it has been executed.

"Hello, Scala"   true   23 + 42 / 2
Expression

defn: A unit of code that returns a value after it has been executed.

"Hello, Scala"   true   23 + 42 / 2

loadUsersFromDatabase()
val x = 25

x * 2
Expression Blocks

```
{
    val x = 25
    x * 2
}
```
Expression Blocks

```scala
val y = {
  val x = 25
  x * 2
}
```
Expression Blocks

```scala
val y = {
  val x = 25
  x * 2
}
val greeting = { "Hello" }
```
Expression Blocks

```scala
val y = {
  val x = 25
  x * 2
}
val greeting = { "Hello" }
val sum = {{ { 2 } } }
```
Expression Block

defn: An expression composed of one or more expressions surrounded by curley braces.
More Expressions
Syntax: If-Else expressions

if (boolean) expression else expression
Syntax: If-Else expressions

```java
if (boolean) expression else expression
```

```java
if (x > 10) {
    println("more")
} else {
    println("less")
}
```
Syntax: If-Else expressions

```java
if (boolean) expression else expression
```

```java
if (x > 10) {
    println("more")
} else {
    println("less")
}
val msg = if (x > 10) { "more" } else { "less" }
```
Syntax: If-Else expressions

if (boolean) expression else expression

if (x > 10) {
    println("more")
} else {
    println("less")
}

val msg = if (x > 10) { "more" } else { "less" }
val msg = if (x > 10) "more" else "less"
Syntax: If-Else expressions

```java
if (x < 0) {
    "less than"
}
else if (x > 0) {
    "greater than"
}
else {
    "about the same"
}
```
Syntax: If-Else expressions

```java
if (x < 0) {
    "less than"
}
else
    if (x > 0) {
        "greater than"
    }
    else {
        "about the same"
    }
```
Expressions are Recursive
val sum = { { { 2 } } }
Syntax: If-Else expressions

if (boolean) expression
else expression
Syntax: If-Else expressions

if (boolean) expression-block
else expression-block
Syntax: If-Else expressions

```java
if (x < 0) {
    "less than"
}
else if (x > 0) {
    "greater than"
}
else {
    "about the same"
}
```
Expressive Pattern Matching
Syntax: Match expressions

expression match {
    case pattern => expression
}

Syntax: Match expressions

```haskell
expression match {  
    case pattern => expression
}
```

```haskell
val hex = color match {  
    case "red" => "#f00"  
    case "green" => "#0f0"  
    case "blue" => "#0ff"
}
```
Syntax: Match expressions

expression match {
  case pattern => expression
}

val valid = parts match {
  case List(12, 24) => true
  case List(18, 32) => true
  case List(22, 28) => false
}
Syntax: Match expressions

expression match {
    case pattern => expression
}

val msg = status match {
    case 200 => "okay"
    case 400 => "not okay"
}
Syntax: Match expressions

expression match {
    case pattern => expression
}

val msg = 500 match {
    case 200 => "okay"
    case 400 => "not okay"
}
Syntax: Match expressions

expression match {
  case pattern => expression
}

val msg = 500 match {
  case 200 => "okay"
  case 400 => "not okay"
}

scala.MatchError: 500
Pattern Matching with Value Binding
Syntax: Value Binding

expression match {
  case pattern => expression
  case value => expression
}

Syntax: Value Binding

expression match {
    case pattern => expression
    case value => expression
}

val msg = 500 match {
    case 200 => "okay"
    case 400 => "not okay"
    case x => "hmm got this: " + x
}
Syntax: Value Binding

expression match {
    case pattern => expression
    case value => expression
}

val valid = parts match {
    case List(12, x ) => true
    case List(18, 32) => true
    case List(13, 28) => false
}
Pattern Matching with Pattern Guards
expression match {
  case pattern => expression
  case value => expression
}
expression match {
    case pattern => expression
    case value if boolean => expression
    case value => expression
}

Syntax: Pattern Guards

expression match {
  case value if boolean => expression
}

---
Syntax: Pattern Guards

expression match {
    case value if boolean => expression
}

val msg = status match {
    case 200 => "okay"
    case x if x < 500 => "odd, got this: " + x
    case x => "error! got this error status: " + x
}
Congrats on mastering Expressions
How about we try out your new skills?
Exercises

1. Write a `match` expression that takes a string and returns it if the string is non-empty, or else returns "n/a".

2. Given a double value, write an `if-else` expression to return "greater" if it is more than zero, "same" if it equals zero, and "less" if it is less than zero. Can you write this as a `match` expression?
Part 4

Functions
What is a Function anyways?
Function

defn: An expression with a name & input parameters
val now = System.currentTimeMillis
val seconds = now / 1000
"Num seconds since 1970 = " + seconds
```java
{ 
    val now = System.currentTimeMillis
    val seconds = now / 1000
    "Num seconds since 1970 = " + seconds
}
```
```scala
def numSeconds: String = {
  val now = System.currentTimeMillis
  val seconds = now / 1000
  "Num seconds since 1970 = " + seconds
}
```
def numSeconds: String = {
    val now = System.currentTimeMillis
    val seconds = now / 1000
    "Num seconds since 1970 = " + seconds
}

val since1970: String = numSeconds()
val since1970: String = {
  val now = System.currentTimeMillis
  val seconds = now / 1000
  "Num seconds since 1970 = " + seconds
}
```scala
def numSeconds(then: Long): String = {
  val now = System.currentTimeMillis
  val seconds = (now - then) / 1000
  "Num seconds since then = " + seconds
}
```
```scala
def numSeconds(then: Long): String = {
  val now = System.currentTimeMillis
  val seconds = (now - then) / 1000
  "Num seconds since then = " + seconds
}

val sinceMay2014 = numSeconds(14000000000000L)
```
Functions Are Expressions
Functions Are
Expressions
with names
Functions Are Expressions

with names and inputs
def name = expression
def name = expression

def greeting = "Hello, World"
val message = greeting + ", here I am!"

println(message)
Syntax: Functions

```python
def name = expression

def greeting = "Hello, World"
def message = greeting + ", here I am!"

println(message)
```
Syntax: Functions

```python
def name[: type] = expression

def greeting: String = "Hello, World"
def message: String = greeting + ", here I am!"

println(message)
```
Syntax: Functions

def name(name1: type1[, , ]) = expression

def divide(a: Int, b: Int) = {
    if (b == 0) 0
    else a / b
}

val speed = divide(miles, hours)
def name(name1: type1[, ,]): type = express

def divide(a: Int, b: Int): Int = {
    if (b == 0) 0
    else a / b
}

val speed: Int = divide(miles, hours)
Now You Know Functions
How about we try out your new skills?
Exercises

1. The formula for computing the area of a circle given its radius is
   \[\text{area} = \text{math.Pi} \times (\text{square of the radius})\]
   Write a function that returns the area of a circle given its radius.

2. Write a function that takes a milliseconds value and returns a string describing the duration in days, hours, minutes and seconds. What's the optimal type for the input value?
Part 5

First-Class Functions
Functions are Expressions
Functions are also Data
val myFunction
myFunction()
val myFunction
myFunction()

val myCopy = myFunction
myCopy()
val myFunction
myFunction()

val myCopy = myFunction
myCopy()

def invoke(someFunction) = someFunction()
invoke( myCopy )
What does this imply?
Functions as data are Storable
Functions as data are Passable
Functions as data are Typed
If Scala data has a Type...
What is the type of a function?
def buzz(x: Int): String = {
    "the value is " + x
}
def buzz(x: Int): String = {
  "the value is " + x
}

val fizz: (Int) => String = buzz
def buzz(x: Int): String = {
    "the value is " + x
}

val fizz: (Int) => String = buzz

fizz(3)
Syntax: Function Types

\((\text{type}[, \text{type}, ...]) \Rightarrow \text{output-type}\)
Syntax: Function Types

(type[, type, ...]) => output-type

def hello(name: String) = "Hello, " + name

val h: String => String = hello
h("World")
Leave The Function At Home
Syntax: Function Literals

(id: type) => expression
Syntax: Function Literals

(id: type) => expression

val doubler = (x: Int) => x * 2
val doubled = doubler(22)
Syntax: Function Literals

(id: type) => expression

val l = (name: String) => {
    val nameLength = name.length
    "Your name is " + nameLength + " letters long!"
}
val doubler = (x: Int) => x * 2
val doubled = doubler(22)
Syntax: Placeholder Syntax

val doubler = (x: Int) => x * 2
val doubled = doubler(22)

val doubler = (x: Int) => _ * 2
val doubled = doubler(22)
val h: String => String = "Hello, " + _
h("World")
val h: String => String = "Hello, " + _
h("World")

val f: (Int, Int) => Int = _ + _
f(4, 6)
Remember, functions are Passable
Higher-Order Functions

defn: A function that takes another function as input
Higher-Order Functions

defn: A function that takes another function as input or as a return value
def reverser(s: String) = s.reverse
val rev = reverser(userName)
def stringSafe(s: String, f: String => String) = {
  if (s != null) f(s) else s
}

def reverser(s: String) = s.reverse

val rev = stringSafe(userNamex, reverser)
Can we make this Simpler?
def stringSafe(s: String, f: String => String) = {
  if (s != null) f(s) else s
}

// def reverser(s: String) = s.reverse

// val rev = stringSafe(userFullName, reverser)
val rev = stringSafe(userFullName, s => s.reverse)
def stringSafe(s: String, f: String => String) = {
  if (s != null) f(s) else s
}

// def reverser(s: String) = s.reverse

// val rev = stringSafe(userUserName, reverser)
val rev = stringSafe(userUserName, _.reverse)
Now You Know

Higher-Order Functions
How about we try out your new skills?
Exercises

1. The format for function literals is:
   \((\text{id}: \text{type}) \Rightarrow \text{expression}\)
   Write a function literal that takes two integers and returns the higher number.

2. Write a higher-order function that takes 4 parameters: 3 integers and a \((\text{Int, Int}) \Rightarrow \text{Int}\) function argument. It should use the function argument with the integers to pick a single winner and return it.

   Try invoking the function with the function literal created in the first exercise.
Part 6

Collections
Collections

defn: Data structures for collecting values of a given type
Immutable Collections

defn: Collections which cannot be modified after they are created.
Lists

Immutable linked lists
List Operations

List(1, 2) :: 3
List(1, 2) ++ List(3, 4)
List(2, 4, 1, 3).sorted
List Operations

scala> val a = List(1, 2)
a: List[Int] = List(1, 2)

scala> val b = a :: 3
b: List[Int] = List(1, 2, 3)

scala> a
res16: List[Int] = List(1, 2)
More List Operations

```scala
define rgb: List[String] = List("red", "green", "blue")
rgb(1) // "red"
rgb.head // "red"
rgb.tail // List("green", "blue")
```
val rgb: List[String] = List("red", "green", "blue")
rgb(1)  // "red"
rgb.head // "red"
rgb.tail // List("green", "blue")
rgb.tail.head // "green"
rgb.tail.tail // "blue"
val rgb: List[String] = List("red", "green", "blue")

val pgb = "purple" :: rgb.tail
More List Operations

```scala
val rgb: List[String] = List("red", "green", "blue")
val pgb = "purple" :: rgb.tail
scala> println(rgb)
List(red, green, blue)
```
Recursive immutable linked lists
Lists of a Higher Order
val colors = List("red", "green", "blue")

colors.map( (c: String) => c.size )
map()

val colors = List("red", "green", "blue")

colors.map( (c: String) => c.size )

colors.map( _.size )
val colors = List("red", "green", "blue")

colors.reduce((a,i) => a + ", " + i)
val nums = List(52.3, 273.1, 92.9, 22.8)

nums.reduce((a, i) => if (a > i) a else i)
val colors = List("red", "green", "blue")

colors.foreach( c => println("Roses are "+c) )
val colors = List("red", "green", "blue")
val shorties = colors.filter(_.size < 5)
Lists + Higher Order Functions = Magic
This is why I just ♥ Scala
How about we try out your new skills?
Exercises

1. You can use this command to create a list of the first 100 integers: `(1 to 100).toList`
   Can you convert this into a list of the first 20 odd numbers?

2. Write a function that takes a list of strings and returns the longest string in the list
Part 7

Classes
Now it's time we got Classy
Scala is Object-Oriented
Class

defn: A bunch of data plus functions which act on that data
Class

defn: A bunch of data plus functions which act on that data
ok, methods which act on that data
class name {
    expressions,
    values &
    methods
}

Syntax: Classes
Syntax: Classes

class name {
  expressions,
  values &
  methods
}

new name()
class User {
    val name: String = "Yubaba"
    def greet = "Hello from " + name
}

class User {
    val name: String = "Yubaba"
    def greet = "Hello from " + name
}

val u = new User()
println( u.greet )
println( "name = " + u.name )
Class

defn: A bunch of data plus functions which act on that data

ok, methods which act on that data
Class

defn: A bunch of data plus functions which act on that data
ok, *methods* which act on that data
ok, the data may be passed as *parameters*
Syntax: Classes

class name(name: type[, ...]) {
    expressions,
    values &
    methods
}

Syntax: Classes

class name(name: type[, ...]) {
    expressions,
    values &
    methods
}

new name(value)
class User(n: String) {
    val name: String = n
    def greet = "Hello from " + name
}
class User(n: String) {
    val name: String = n
    def greet = "Hello from " + name
}

val u = new User("Zeniba")
println( u.greet )
println( "name = " + u.name )
Syntax: Classes

class name(val name: type[, ...]) {
    expressions,
    values &
    methods
}
Syntax: Classes

class name(val name: type[, ..., ...]) {
    expressions,
    values &
    methods
}

ew name(value)
class User(val name: String) {
    def greet = "Hello from " + name
}

```scala
class User(val name: String) {
    def greet = "Hello from " + name
}

val u = new User("Zeniba")
println( u.greet )
println( "name = " + u.name )
```
Subtyping & Polymorphism
Syntax: Extending Classes

class Child extends Parent {

  def method {}

  override def method { super.method() }

}
Syntax: Extending Classes

class A {
    def hi = "Hi from A"
}

new A().hi // Hi from A
Syntax: Extending Classes

class A {
    def hi = "Hi from A"
}

new A().hi // Hi from A

class B extends A {
    override def hi = { "B says " + super.hi }
}

new B().hi // B says Hi from A
What's better than Classes?
Case Classes
Case Class

defn: A class with special features for holding data
Case Class

defn: A class with special features for holding data where class parameters are automatically fields
Case Class

defn: A class with special features for holding data where class parameters are automatically fields and is easily printable
Case Class

defn: A class with special features for holding data where class parameters are automatically fields and is easily printable and comparable
Syntax: Case Classes

case class Name()
case class Name()

case class User(name: String)
val u = User("Hadrian")
Syntax: Case Classes

```scala
case class Name()

case class User(name: String)
val u = User("Hadrian")
println(u) // User(Hadrian)
```
case class Name()

case class User(name: String)
val u = User("Hadrian")
println(u) // User(Hadrian)

u == User("Royce") // false
It's Time For Singleton's
Singleton
defn: A class that may have zero or one instances
Object

defn: A class that may have zero or one instances
Syntax: Objects

object name
object HtmUtils {
    def removeMarkup(input: String) = {
        input.replaceAll(".*<\/?\w[^>]*>.*","").replaceAll("<\.*>\","")
    }
}
object name

object HtmlUtils {
    def removeMarkup(input: String) = {
        input.replaceAll("[\/*\?\|\^\$][\w\s]+\*"">"",""")
            .replaceAll("<\.*>","")
    }
}

HtmlUtils.removeMarkup( htmlText )
We have covered Classes
and Case Classes
plus Objects
It's Time For Singleton's
Wait! I have a much better idea!
How about we try out your new skills?
Exercises

1. A case class can be created with the following syntax:

```scala
case class Thing(a: Int, b: String)
```

Create a video gaming console class that can track the make, model, debut date, and physical media formats supported. Can you compare two instances of this class?

2. Write a function that takes a list of instances of the video gaming console class and prints out the name of the most recent console.
And Now For Something Completely Different
Class Project - MathAPI

2. Follow the instructions to compile and run the server
3. Fix the broken "max" and "average" services
4. Run "sbt run" after each change to try out your changes.
Q & A
Get Started Developing With Scala

Jason Swartz
Jason Swartz

@swartzrock
- Scala By The Bay
- Big-Data Scala
- OSCON discount: LEARNSCALABYTHEBAY25
- scala.bythebay.io
Thanks Everyone!