GREG HAZEN

100 DAYS OF SCALA

OUTLINE

- Scala 101
- Who is Scala Right For?
- Learning More

OUTLINE

- Who are we?
- Why Scala?

memeguy.com

SCALA 101 - HELLO WORLD

```scala
object HelloWorld {
  def main(args: Array[String]): Unit = {
    println("Hello, World!")
  }
}
```

SCALA 101 - HELLO WORLD

```scala
object HelloWorld {
  def main(args: Array[String]): Unit = {
    println(getGreeting)
  }

  def getGreeting: String = "Hello, World!"
}
```

```scala
import org.scalatest.FlatSpec

class HelloWorldSpec extends FlatSpec {
  "getGreeting" should "return Hello, World" in {
    assert(HelloWorld.getGreeting == "Hello, World!")
  }
}
```

SCALA 101 - VARIABLES

```scala
var mutableVariable: Int = 0
val immutableVariable: Int = 0

def incrementMutableVariable: Int = {
  mutableVariable = mutableVariable + 1
  mutableVariable
}
// Won't compile
// def incrementImmutableVariable = immutableVariable = immutableVariable + 1
```

```scala
assert(incrementMutableVariable == 1)
assert(incrementMutableVariable == 2)
assert(incrementMutableVariable == 3)
assert(incrementMutableVariable == 4)
```

"...val and var are just two different tools in your toolbox..."

SCALA 101 - FUNCTIONS

```scala
def add(a: Int, b: Int)~~: Int~~ = a + b

// Note Scala is purely object-oriented
def addIsActuallyAFunction(a: Int, b: Int)~~: Int~~ = a.+(b)
```

```scala
assert(add(1, 2) == 3)

assert(add(1, 2) == addIsActuallyAFunction(1, 2))
```

```scala
def placeholders = (_: Int) + (_: Int)
def placeholdersWithParams = (a: Int, b: Int) => a + b
```

```scala
assert(placeholders(1, 2) == 3)
assert(placeholdersWithParams(1, 2) == 3)
```

How do you keep your team consistent?

SCALA 101 - CLASSES

```scala
class MyClass(val a: Int, b: String) {
  val c: Int = 3
}
```

```scala
val classInstance = new MyClass(1, “b")

assert(classInstance.a == 1)
// Doesn't compile
// assert(classInstance.b == "b")
assert(classInstance.c == 3)
```

```scala
case class MyCaseClass(a: Int, b: String) {
  val c: Int = 3
}
```

```scala
val caseClassInstance = new MyCaseClass(1, "b")

assert(caseClassInstance.a == 1)
assert(caseClassInstance.b == "b")
assert(caseClassInstance.c == 3)

val anotherCaseClass = MyCaseClass(1, "b")
assert(caseClassInstance == anotherCaseClass)

assert("MyCaseClass(1,b)" == caseClassInstance.toString)
```

```scala
object SingletonObject {
  val a: Int = 1
  val b: String = "b"
}
```

```scala
assert(SingletonObject.a == 1)
assert(SingletonObject.b == “b")
```

```scala
trait MyFirstTrait {
  def getName: String = "Trait 1"
}

trait MySecondTrait {
  val number = 50
}

class MyMixin extends MyFirstTrait with MySecondTrait
```

```scala
val mixin: MyMixin = new MyMixin
assert(mixin.getName == "Trait 1")
assert(mixin.number == 50)
```

```scala
trait MyFirstTrait {
  def getName: String = "Trait 1"
}

trait MySecondTrait {
  val number = 50
}

class MyMixinOverride extends MyFirstTrait with MySecondTrait {
  override def getName: String = "My Mixin”
  // Note this doesn't compile
  // override number = 75
}
```

```scala
assert((new MyMixinOverride).getName == "My Mixin")
```

```scala
trait MyFirstTrait {
  def getName: String = "Trait 1"
}

trait MySecondTrait {
  def getName: String = "Trait 2"
}

// Note this creates a compiler error
// class MyMixinError extends MyFirstTrait with MySecondTrait
```

```scala
class MyMixinError extends MyFirstTrait with MySecondTrait {
  override def getName: String = super.getName
}
```

```scala
assert((new MyMixinError).getName == “Trait 2”)
```

```scala
val arrayOfSize3 = new Array[Int](3)
// Note you can edit what's in the array
arrayOfSize3(0) = 0
arrayOfSize3(1) = 1
arrayOfSize3(2) = 2

val arrayOfGivenElements = Array.apply(0, 1, 2)

val arrayWithoutApply = Array(0, 1, 2)
```

```scala
assert(arrayOfSize3 sameElements arrayOfGivenElements)

assert(arrayOfSize3(0) == 0)
// Have to use `sameElements` instead
val arrayComparison = arrayOfSize3 == arrayOfGivenElements
assert(!arrayComparison)
assert(arrayOfSize3.head == 0)
assert(arrayOfSize3.tail sameElements Array(1, 2))
```

```scala
val listOfGivenElements = List(0, 1, 2, 3)

val listOfAppends = 0 :: 1 :: 2 :: 3 :: Nil

val listA = List(0, 1)
val listB = List(2, 3)
val listOfAppendedLists = listA ::: listB
```

```scala
assert(listOfGivenElements == listOfAppends).contains(i))
assert(listOfGivenElements == listOfAppendedLists)
assert(listOfGivenElements(0) == 0)
assert(listOfGivenElements(1) == 1)
assert(listA == List(0, 1))2) == 2)
assert(listB == List(2, 3)) ) == 3)

assert(listOfGivenElements.head == 0)
```

```scala
val arrayOfSize3 = Array.apply(0, 1, 2)
val listOfGivenElements = List(0, 1, 2, 3)
```

```scala
assert(arrayOfSize3.sum == 3)
assert(listOfGivenElements.sum == 6)

assert(arrayOfSize3.count(_ > 1) == 1)
assert(listOfGivenElements.count(_ > 1) == 2)
assert(listOfGivenElements.count(i => i > 1) == 2)

assert(listOfGivenElements.filter(_ > 1) == List(2, 3))

assert(listOfGivenElements.map(_ + 10) == List(10, 11, 12, 13))
```

```scala
val tuple2 = ("first", "second")
val tuple5 = ("one", 2, "three", 4, "five")
```

```scala
assert(tuple2._1 == "first")
assert(tuple2._2 == "second")

assert(tuple2 == ("first", "second"))

assert(tuple5 == ("one", 2, "three", 4, "five"))
```

```scala
val map = Map[Int, String](
  0 -> "zero",
  1 -> "one",
  2 -> "two"
)

val mapFromList = List(
  (0, "zero"),
  (1, "one"),
  (2, "two")
).toMap
```

```scala
assert(map(0) == "zero")
assert(map(1) == "one")
assert(map(2) == "two")

assert(map == mapFromList)
```

```scala
val mutableSet = mutable.Set[Int](1)
mutableSet += 2
mutableSet += 3

val immutableSetVal1 = immutable.Set[Int](1)
// Does not compile
// immutableSetVal1 += 2
val immutableSetVal2 = immutableSetVal1 + 2
val immutableSetVal3 = immutableSetVal2 + 3

var immutableSetVar = immutable.Set[Int](1)
immutableSetVar += 2
immutableSetVar += 3
```

“For some problems, mutable collections work better, while for others, immutable collections work better…”

SCALA 101 - OPTIONS

```scala
val myNullOption = Option(null)
val myValueOption = Option("Options are cool")
```

```scala
assert(myNullOption == None)
assert(myNullOption.isEmpty)

assert(myNullOption.get == "Options are cool")
assert(myNullOption.isDefined)
```

SCALA 101 - OPTIONS

```scala
def sayHello(name: Option[String]): String = name match {
  case None => "Hello, Stranger"
  case n => s"Hello, ${n.get}"
}
```

```scala
assert(sayHello(Option("Greg")) == "Hello, Greg”)

assert(sayHello(None) == "Hello, Stranger")
```

```scala
def sayHelloToAll(names: List[String]): String = names match {
  case Nil => "Anyone there?"
  case List(n) => s"Hello, $n!"
  case List("Greg", "Tim") => "Shouldn't you guys be presenting?!"
  case _ => s"Welcome everyone including ${names.mkString(", ")}"
}
```

```scala
assert(sayHelloToAll(List()) == "Anyone there?")

assert(sayHelloToAll(List("Greg")) == "Hello, Greg!")

assert(sayHelloToAll(List("Greg", "Tim"))
       == "Shouldn't you guys be presenting?!")

assert(sayHelloToAll(List("Larry", "Curly", "Mo"))
      == "Welcome everyone including Larry, Curly, Mo")
```

```scala
def sayHelloToAll(names: List[String]): String = names match {
  case Nil => "Anyone there?"
  case List(n) => s"Hello, $n!"
  case List("Greg", "Tim") => "Shouldn't you guys be presenting?!"
  case List("Greg", "Tim", n) =>
    s"Only $n is waiting for you to present!"
  case "Greg" :: "Tim" :: n =>
    s"${n.mkString(", ")} are all waiting for you to present!"
  case _ => s"Welcome everyone including ${names.mkString(", ")}"
}
```

```scala
assert(sayHelloToAll(List("Greg", "Tim", "Larry"))
       == "Only Larry is waiting for you to present!")

assert(sayHelloToAll(List("Greg", "Tim", "Larry", "Curly", "Mo"))
       == "Larry, Curly, Mo are all waiting for you to present!")
```

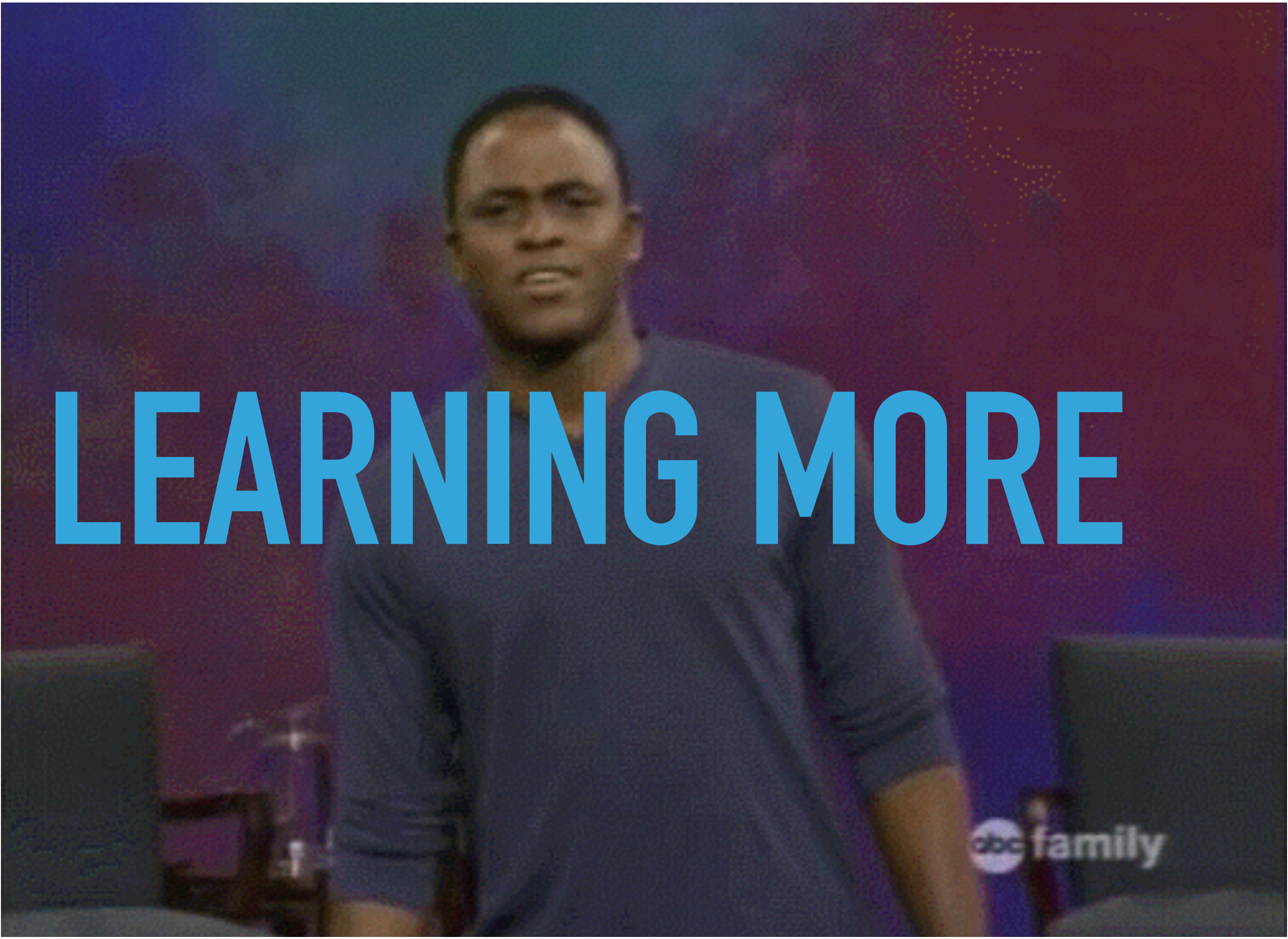
WHO IS SCALA RIGHT FOR?

WHO IS SCALA RIGHT FOR?

- Investment in learning
  - Scala support groups
  - How do you keep everyone getting better without intimidating them?
- It's really difficult for green and experienced developers who are both new to Scala to pair
- Green developers need to learn the basics before taking advantage of the concise manner of Scala
- Staffing and Recruiting

WHO IS SCALA RIGHT FOR?

- Maintenance
- Lots of strong feeling for and against Scala
  - Some people are super into Scala because of functional programming and really higher logic that makes the learning curve for green developers nearly impossible
- Challenges with SBT
- So many ways to do the same thing; how do you get consistency across the team?


LEARNING MORE
abc family

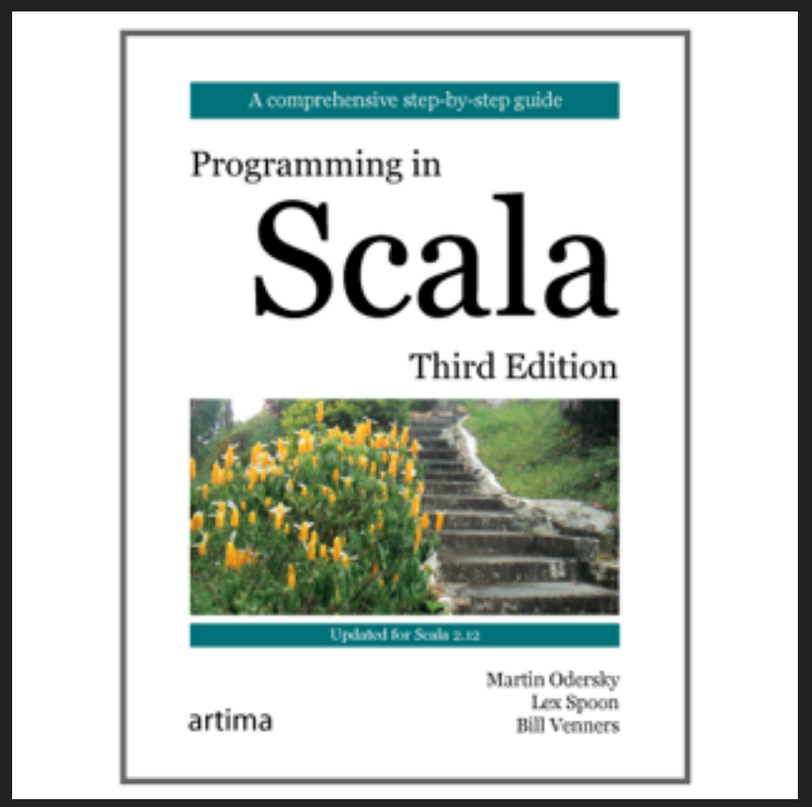
REFERENCES

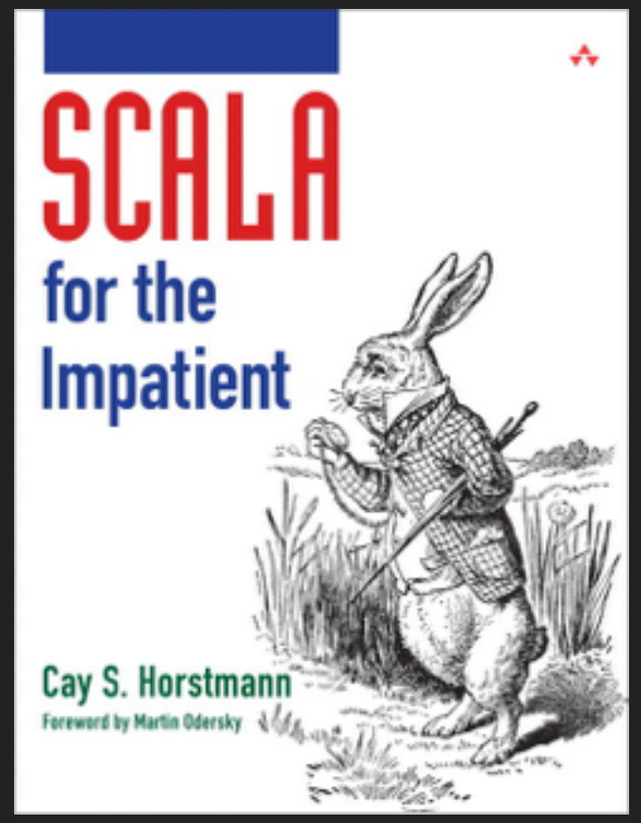
- Scala: Deeply Functional, Purely Object-Oriented
  ADRIAAN MOORS
  ORACLE.COM/JAVAMAGAZINE
  JANUARY/FEBRUARY 2017

- Programming in Scala, 3rd Edition
  Martin Odersky, Lex Spoon, Bill Venners

OTHER WAYS TO LEARN

- Books
  - Scala: Deeply Functional, Purely Object-Oriented
  - Programming in Scala, 3rd Edition
  - Scala for the Impatient
- Several free courses on Coursera
- Twitter's Scala School





```scala
object Salutations {
  def getSalutationFor(timing: String): String = timing match {
    case "start" => "Welcome to 100 Days of Scala!"
    case "middle" => "Are you still awake?"
    case "end" => "Go fourth and use Scala!"
    case _ => "Uh oh..."
  }
  def main(args: Array[String]): Unit = {
    println(getSalutationFor("end"))
  }
}
```