

Demystifying Scala

Kelley Robinson

Engineering Team Lead, Sharethrough

@kelleyrobinson

Demystifying Scala





📍 Lausanne

Demystifying Scala

Kelley Robinson

Engineering Team Lead, Sharethrough

@kelleyrobinson





Kelley Robinson

Engineering Team Lead, Sharethrough

@kelleyrobinson



Introduction to Scala



Why Scala



How to get started



Introduction to Scala



Why Scala



How to get started

A woman with short brown hair, wearing a dark blue button-up jacket and a gold necklace, stands in a room with a large bookshelf in the background. She is holding a glass of wine in her right hand and resting her left hand on the back of a green armchair. The bookshelf is filled with many books of various colors.

**Ina Garten a.k.a.
The Barefoot Contessa**

Syntax Intro


```
"How easy is that?".isEmpty  
// Boolean = false
```

```
val chefs = List("Ina", "Martha", "Giada")  
// List[String] = List(Ina, Martha, Giada)
```

```
chefs(0)  
// String = Ina
```




Many ways to write a function...

```
def isStoreBoughtFine(yourName: String): Boolean = {  
    val areYouAnyoneButIna = yourName != "Ina Garten"  
    areYouAnyoneButIna  
}
```


Many ways to write a function...

```
def isStoreBoughtFine(yourName: String): Boolean = {  
    yourName != "Ina Garten"  
}
```


Many ways to write a function...

```
def isStoreBoughtFine(yourName: String): Boolean =  
    yourName != "Ina Garten"
```


Many ways to write a function...

```
def isStoreBoughtFine(yourName: String) =  
    yourName != "Ina Garten"
```


Types provide safety

```
def isStoreBoughtFine(yourName: String): Boolean =  
    "Ina Garten"
```

```
// error: type mismatch;  
// found   : String  
// required: Boolean
```


Calling a function

```
def isStoreBoughtFine(yourName: String): Boolean
```

```
// ok
```

```
isStoreBoughtFine("Kelley")
```

```
// oh no!
```

```
isStoreBoughtFine(42)
```

```
// error: type mismatch;
```

```
// found    : Int(42)
```

```
// required: String
```


Data is immutable by default

```
val barefootContessa = "Ina Garten"
```

```
val barefootContessa = "Kelley Robinson"
```

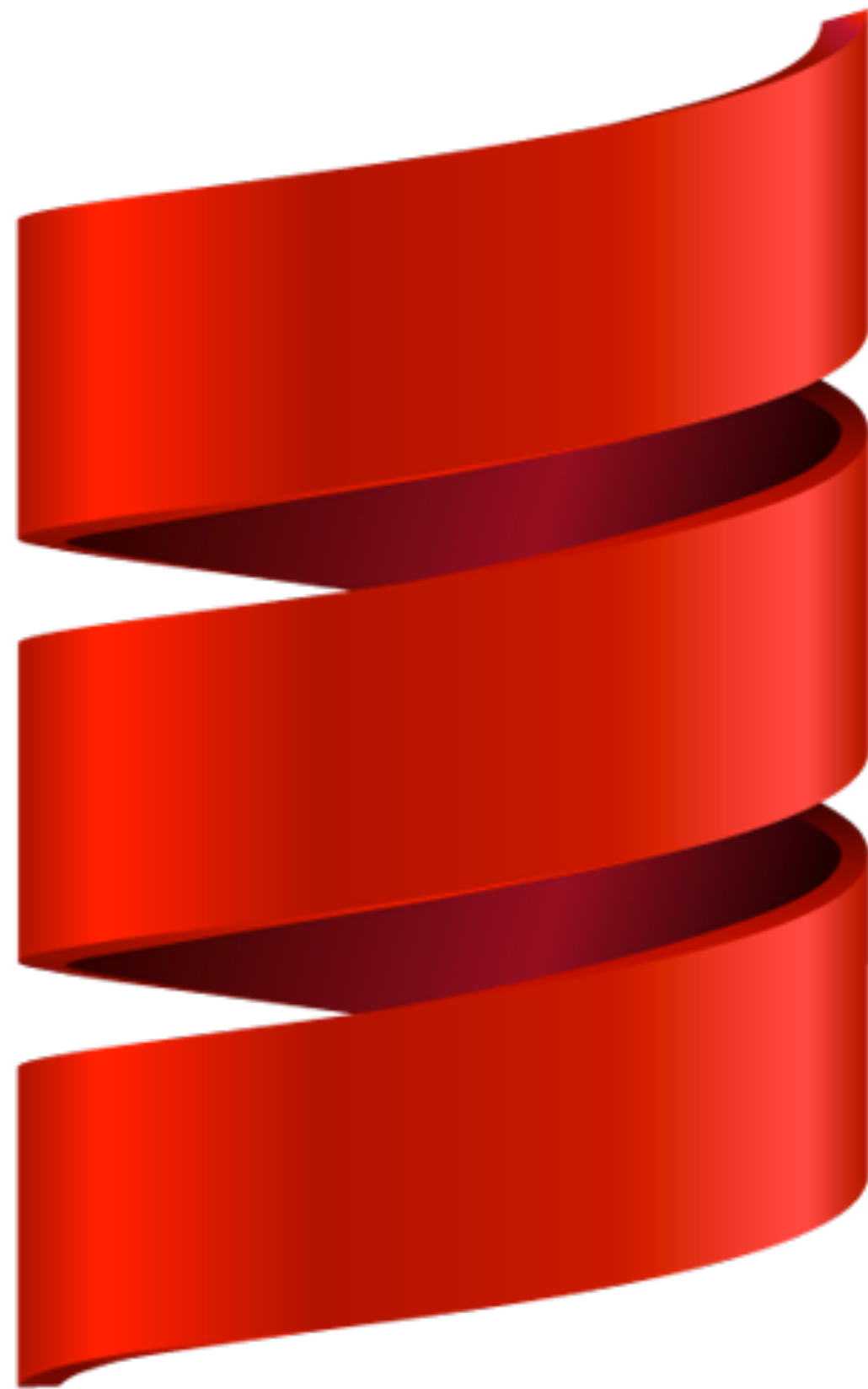
```
// <console> error: barefootContessa is already defined
```

```
val yaskween = barefootContessa.toUpperCase
```

```
println(barefootContessa)
```

```
// Ina Garten
```


Scala in a Nutshell



Object-Oriented Meets Functional

Have the best of both worlds.

Functional Programming

FP: Everything is an expression

- think in values, not side-effects
- statement is an expression with Unit return type

FP: Avoid Side Effects

Changes outside of function scope

- Modifies state
- Not predictable



FP: Referential transparency

An expression can be replaced by
its value without changing the
program behavior



FP: Immutable data

Avoid modifying state



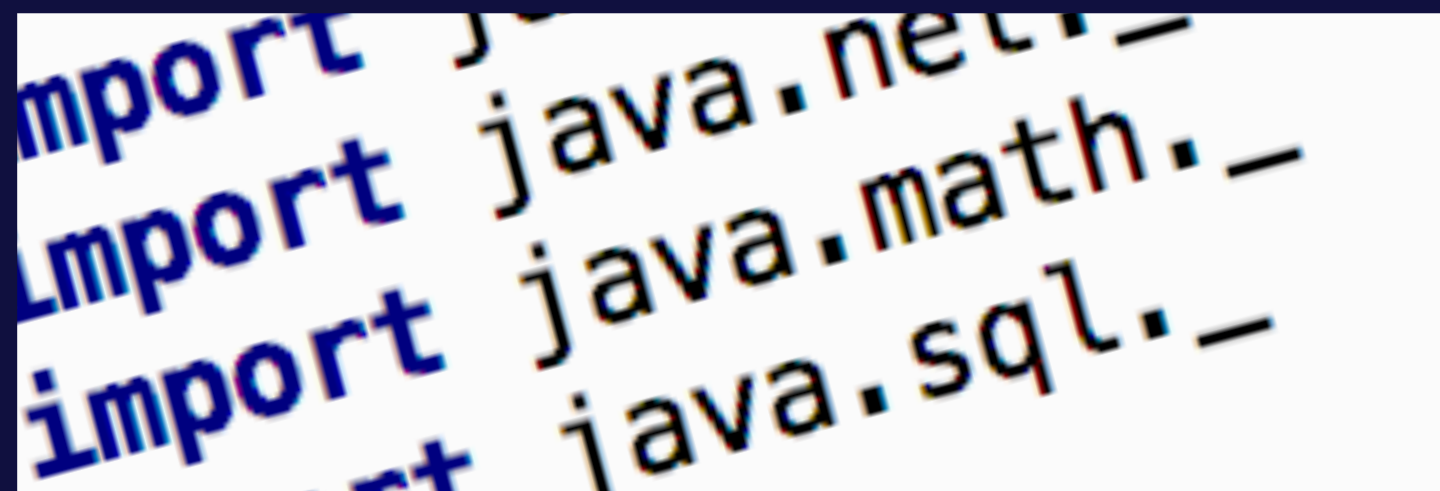
FP: Expressions

The programmer benefits

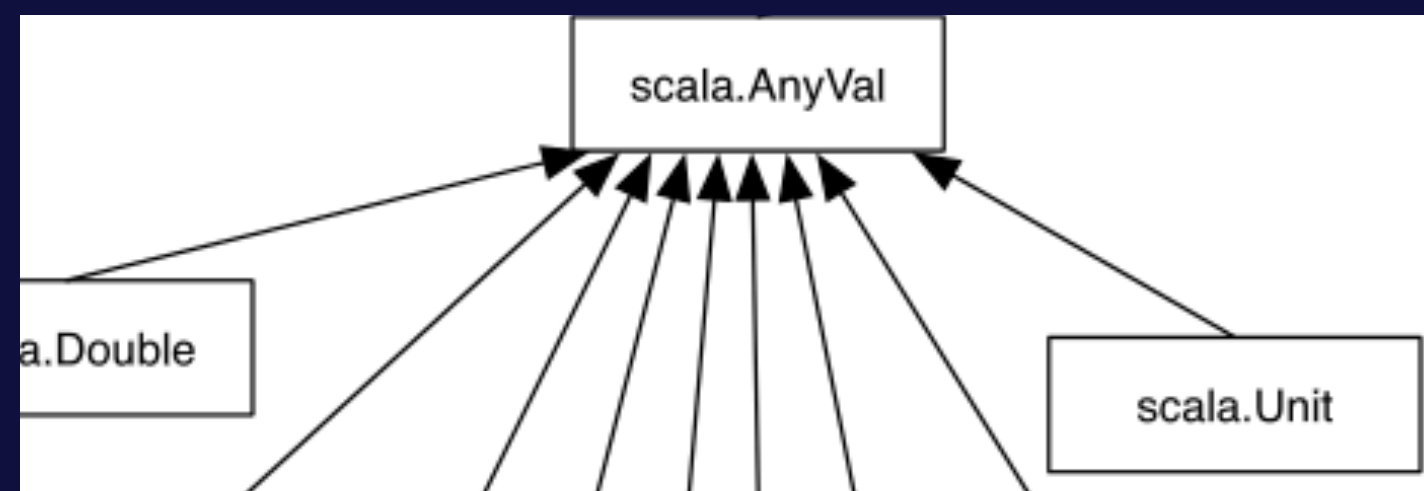
- Understanding
- Predictability
- Unit testing



Scala in a Nutshell



SEAMLESS JAVA INTEROP



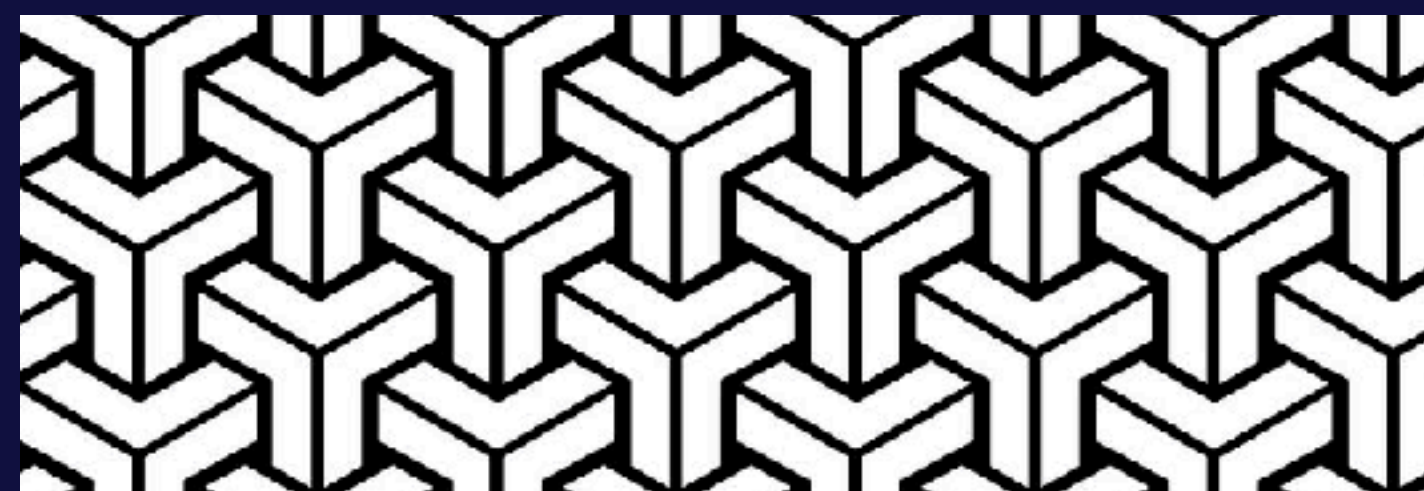
TYPE INFERENCE



CONCURRENCY & DISTRIBUTION



TRAITS



PATTERN MATCHING



HIGHER-ORDER FUNCTIONS



Seamless Java Interop

All the benefits of the Java
Virtual Machine (JVM)

Type Inference

Let the type system
work for you

Traits

Flexible interfaces


```
trait CelebrityChef

trait FoodNetworkStar extends CelebrityChef {
  val showName: String
}

trait BestSellingAuthor {
  val bookName: String
}

object InaGarten extends FoodNetworkStar
  with BestSellingAuthor {
  val bookName: String = "Cooking for Jeffrey"
  val showName: String = "Barefoot Contessa"
}
```



```
trait CelebrityChef
```

```
trait FoodNetworkStar extends CelebrityChef
```

```
object InaGarten extends FoodNetworkStar with BestSellingAuthor
```

```
object RoyChoi extends CelebrityChef
```

```
def learnToCook(instructor: CelebrityChef) = { ... }
```


Pattern Matching

“Switch” on steroids

Pattern Matching

```
def matchTest(x: Int): String =  
  x match {  
    case 1 => "one"  
    case 2 => "two"  
    case _ => "many"  
  }
```

```
println(matchTest(3))  
// "many"
```


Pattern Matching

```
sealed trait Ingredient
case object Produce extends Ingredient
case object Meat extends Ingredient
case object DryGood extends Ingredient

def storageLocation(i: Ingredient): String =
  i match {
    case Produce => "Refrigerator"
    case Meat    => "Freezer"
    case DryGood => "Pantry"
  }
```


Higher-order Functions

Be expressive and succinct


```
trait Ingredient { val shelfLife: Int }
```

Scala

```
val ingredients: Array[Ingredient]  
val (eatNow, eatLater) = ingredients.partition(i => i.shelfLife < 7)
```

Java

```
List<Ingredient> ingredients;  
List<Ingredient> eatNow = new ArrayList<Ingredient>(ingredients.size());  
List<Ingredient> eatLater = new ArrayList<Ingredient>(ingredients.size());  
for (Ingredient ingredient : ingredients) {  
    if (ingredient.getShelfLife() < 7)  
        eatNow.add(ingredient);  
    else  
        eatLater.add(ingredient);  
}
```



```
trait Ingredient { val shelfLife: Int }
```

Scala

```
val ingredients: Array[Ingredient]
```

```
val (eatNow, eatLater) = ingredients.partition(i => i.shelfLife < 7)
```

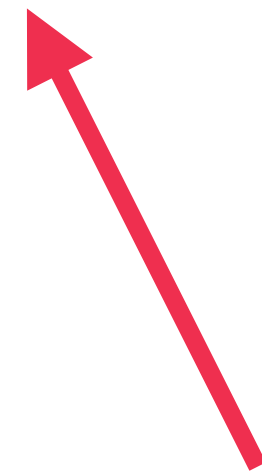
Partition takes a function as its argument

```
def partition(p: Int => Boolean)
```


These are equivalent

```
partition(i => i.shelfLife < 7)
```

```
partition(_.shelfLife < 7)
```



underscore is a convenience operator



Introduction to Scala



Why Scala



How to get started



Sharethrough chooses Scala to build industry's first native ad management system

“Sharethrough developers were quickly productive with Scala, which can be attributed to their familiarity with Java, Scala’s expressiveness and lack of boilerplate code.”

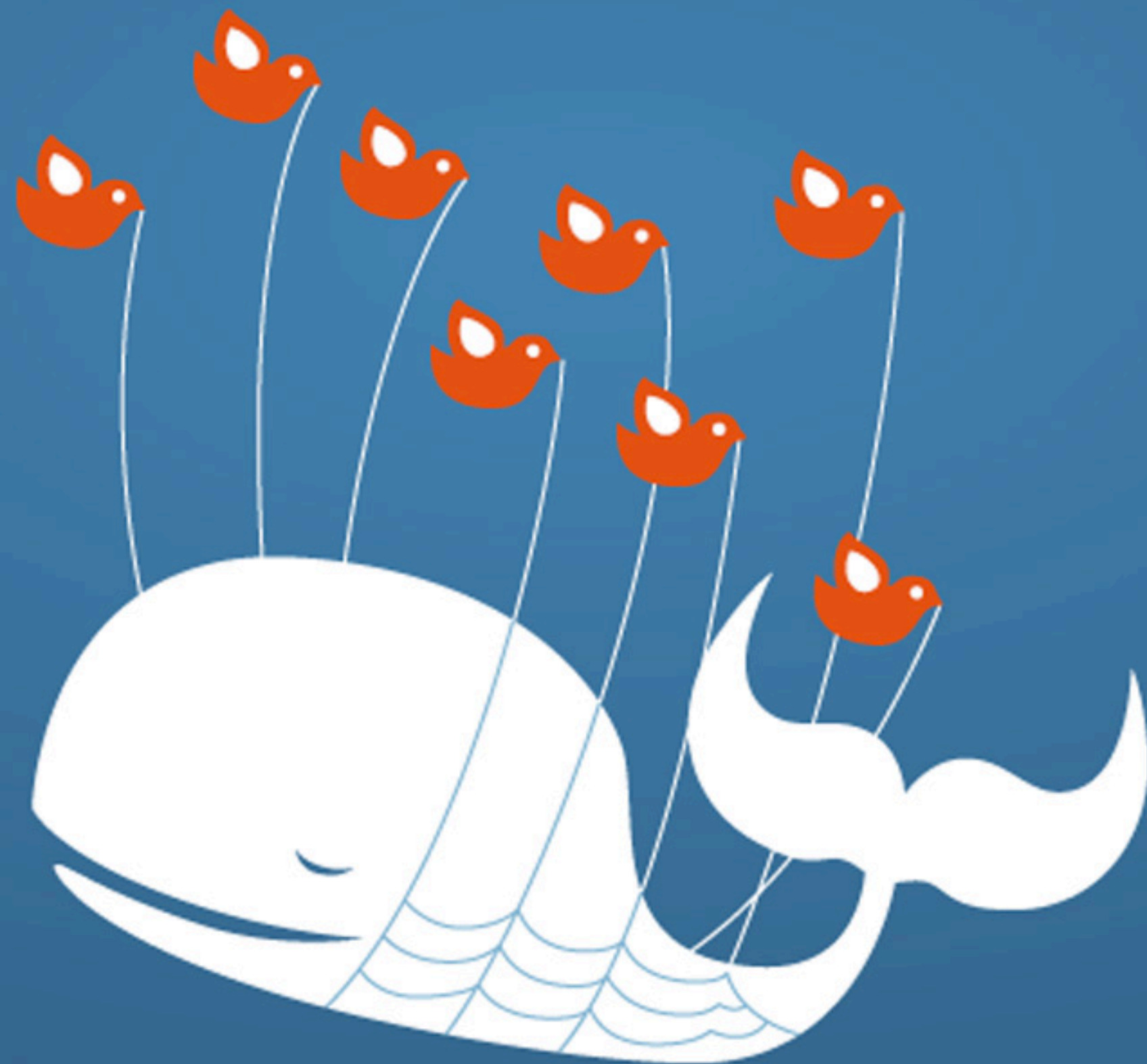
Twitter is over capacity.

Please wait a moment and try again. For more information, check out **Twitter Status**.

[Bahasa Indonesia](#) [Bahasa Melayu](#) [Deutsch](#) [English](#) [Español](#) [Filipino](#) [Français](#) [Italiano](#) [Nederlands](#) [Português](#) [Türkçe](#)

[Русский](#) [हिन्दी](#) [日本語](#) [简体中文](#) [繁體中文](#) [한국어](#)

© 2012 Twitter [About](#) [Help](#) [Status](#)



**Kelley Robinson**

@kelleyrobinson

 Follow

Hey [#scala](#) devs - I'm collecting your 140 character answer to the question:

Why Scala?

4:26 PM - 14 Jan 2017



148



118



Caitie McCaffrey

@caitie

 Follow

[@kelleyrobinson](#) accessible functional programming on the
jvm

4:58 PM - 14 Jan 2017 · San Francisco, CA



25



Shane Delmore

@ShaneDelmore

 **Follow**

[@kelleyrobinson](#) most functional, easiest to refactor
language I can get paid to write

7:01 PM - 14 Jan 2017



15

**Travis Brown**

@travisbrown

 **Follow**

[@kelleyrobinson](#) *because* it's a disaster—but a disaster with a lot of the right raw materials lying around. e.g. [meta.plasm.us/posts/2015/07/...](https://meta.plasm.us/posts/2015/07/11/roll-your-own-scala/)

11:09 PM - 14 Jan 2017 · Evanston, IL



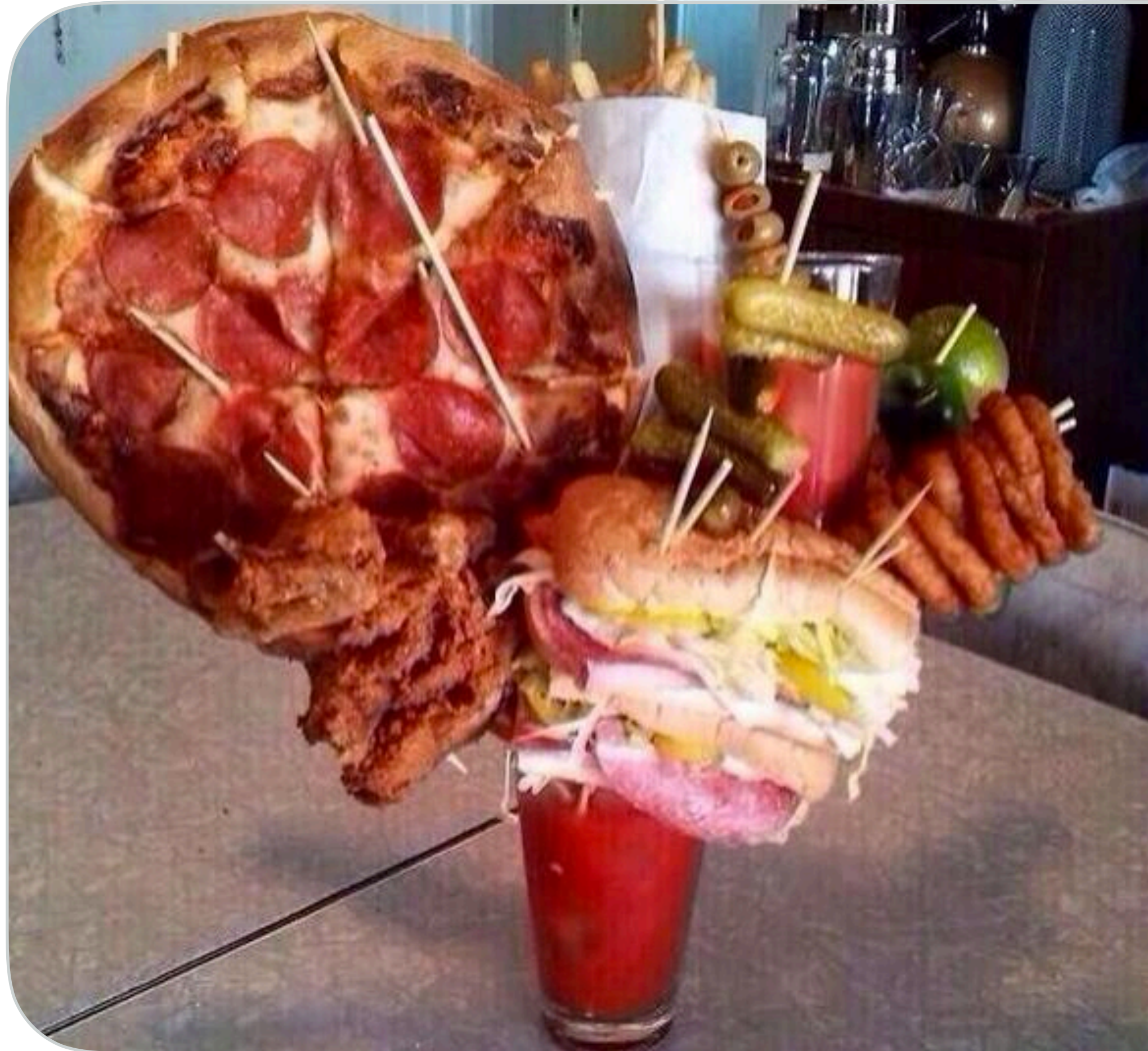
2



24



Flexibility requires discipline



Ian Chan
@chanian

Scala

10:37 PM - 1 May 2014



888



628

Sad truths about Scala



Scala Functional Spectrum



Java



Haskell




```
// source code yikes
object Task {
  implicit val taskInstance:
    Nondeterminism[Task] with Catchable[Task]
    with MonadError[(type  $\lambda[\alpha, \beta]$  =
      Task[ $\beta$ ])# $\lambda$ , Throwable] = new
    Nondeterminism[Task] with Catchable[Task]
    with MonadError[(type  $\lambda[\alpha, \beta]$  =
      Task[ $\beta$ ])# $\lambda$ , Throwable] { ... }
}
```


These are equivalent

```
partition(i => i.shelfLife < 7)
```

```
partition(_.shelfLife < 7)
```



underscore is a convenience operator

Symbolic operators

// this is valid code

```
def @*%--@\\&#&(foo: String): String = foo
```

// exists for this purpose

```
def +(s1: String, s2: String): String = s1.concat(s2)
```


“Modern” Functional Programming is cluttered with jargon

Sad truths about Scala



**Scala is still
pretty awesome**

Scala is still pretty awesome

- Powerful
- Flexible
- Creative



Introduction to Scala



Why Scala

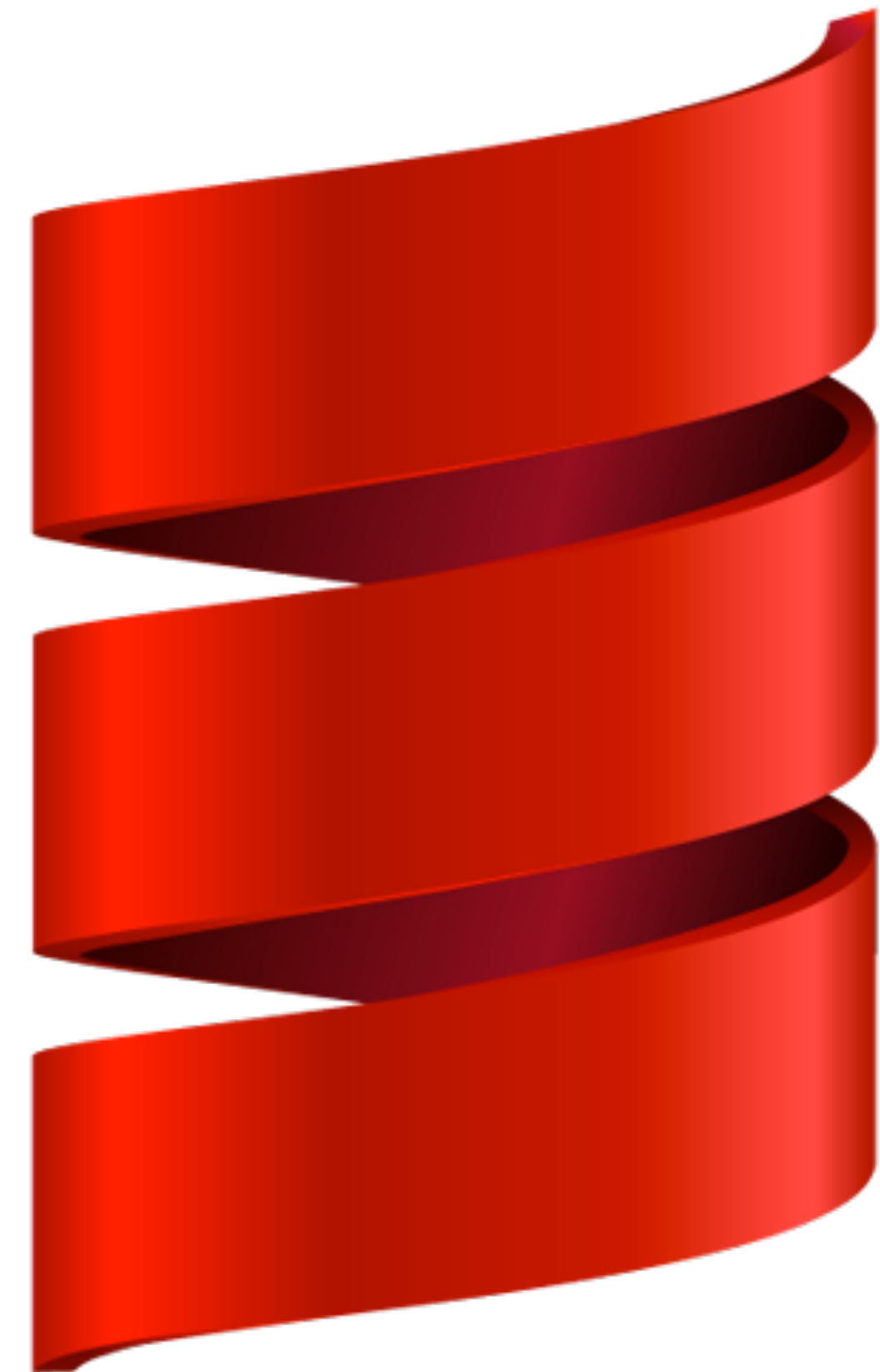


How to get started

Scala REPL

Core Language

<https://www.scala-lang.org/download/install.html>





Functional Programming Principles in Scala

Scala Center + Coursera

<https://www.coursera.org/learn/progfun1>

Scala Exercises

47 Degrees

https://www.scala-exercises.org/scala_tutorial/terms_and_types



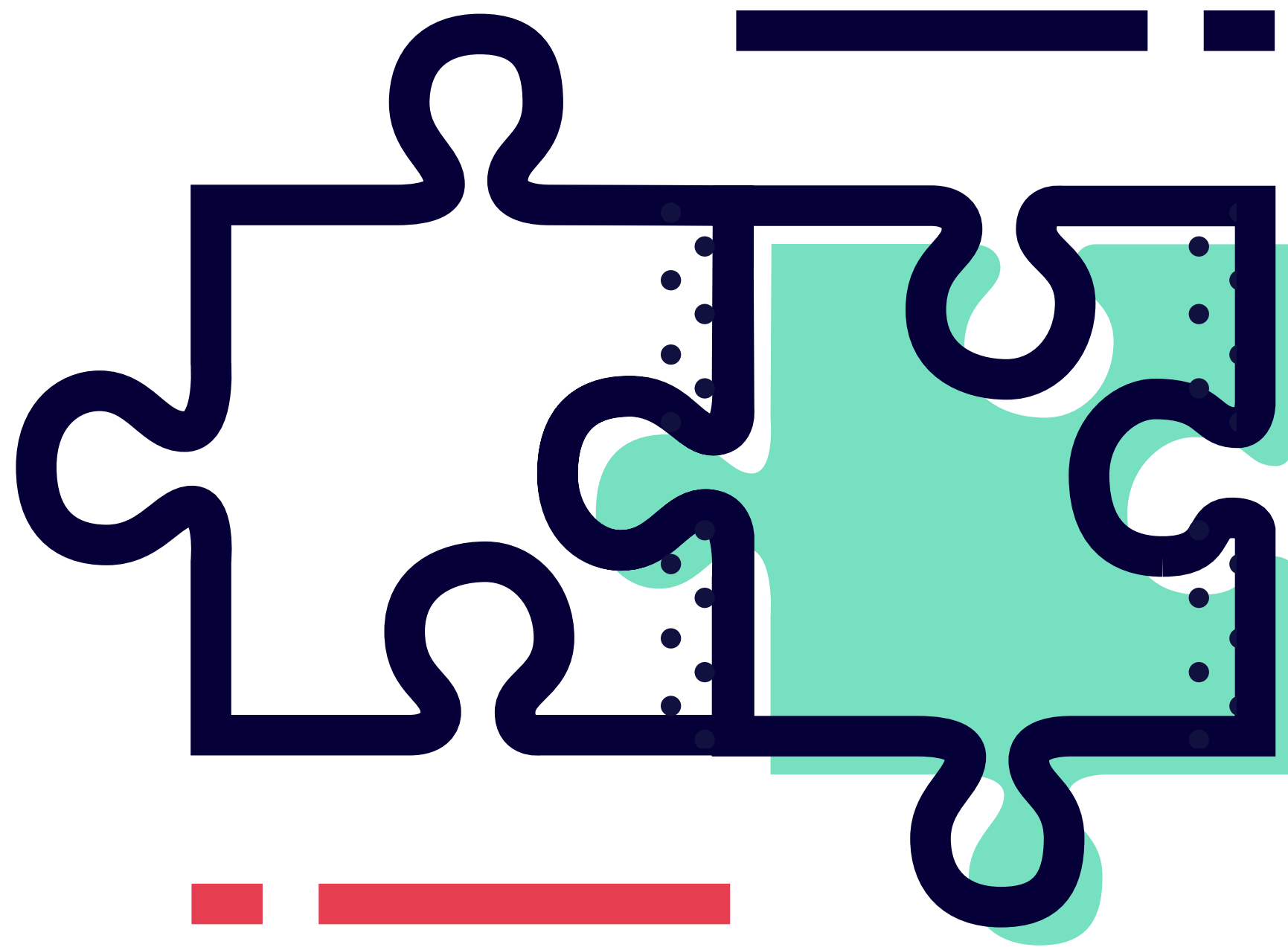
ScalaBridge

Introductory programming
workshops for underrepresented
groups

- Curriculum is free and open source
- Teaching is a great way to learn

<https://scalabridge.gitbooks.io/curriculum/content/setup/setup.html>





- **Don't be intimidated**
- **Have fun**
- **Be disciplined**

A scenic view of a historic European city, likely Bern, Switzerland. The foreground shows the rooftops and walls of old buildings. A large, ornate cathedral with multiple spires is prominent in the middle ground. In the background, a range of rugged, snow-capped mountains rises above a body of water. The sky is blue with some light clouds.

Thank You!

[@kelleyrobinson](#)

hello@krobinson.me