

Are you interested in
functional programming
and looking for an
alternative to Haskell?

Explore a new world...

Scala

putting the fun into functional programming

An interactive introduction to the
language with Keoni D'Souza





Do you want to look
at a language that
allows you to
implement FP
nicely?



Do you want to look at a language that allows you to implement FP nicely?



Do you want to look at a language that has a similar syntax to one you've seen already?



Do you want
at a language
allows you
implement
nicely?

Do you want

Do you want to look
at a language that
companies actually
use?



Do you want to look
at a language that
allows you to
implement FP
nicely?



Do you
(not) like
Haskell?



Do you want to look
at a language that
allows you to
implement things
nicely?

Do you want
to look at a
language that
has a similar
implementation

Well, aren't you in luck?!

Do you
(not) like
askell?

Do you want to look
at a language that
companies actually
use?



```
val training_df = sqlContext.createDataFrame(training_pca)
val test_df = sqlContext.createDataFrame(test_pca)
```

```
val tokenizer = new Tokenizer()
val hashingTF = new HashingTF()
val lr = new LogisticRegression()
val pipeline = new Pipeline()
```

```
val mod = pipeline.fit(training_df)
mod.transform(test_df)
  .select("id", "text", "probability", "prediction")
  .collect()
  .foreach { case Row(id: Long, text: String, prob: Vector, prediction: Double) =>
    println(s"(id, $text) --> prob=$prob, prediction=$prediction")
}
```

solves concurrency
in a "safer way"

static typing

"makes me a better
programmer"

pattern
matching

```
val training_df = sqlContext.createDataFrame(training_pca)
val test_df = sqlContext.createDataFrame(test_pca)|
```

```
val tokenizer = new Tokenizer()
val hashingTF = new HashingTF()
val lr1 = new LogisticRegression()
val pipeline = new Pipeline()
```

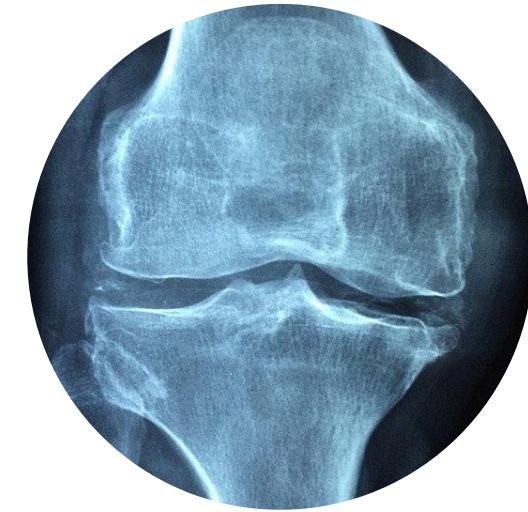
```
val mod = pipeline.fit(training_df)
```

Keoni D'Souza, 921231
mod.transform(test_df)
.select("id", "text", "probability", "prediction")
.collect()
.foreach { case Row(id: Long, text: String, prob: Vector, prediction: Double) =>
 println(s"(\$id, \$text, \${prob.map(_.toString).mkString(", ")}, \$prediction)")
}

Friday, 14 February 2020

Scala

*putting the fun into
functional programming*



Benvenuto/a!

Добро пожаловать!

Välkommen!

Witaj!

Bem-vindo/a!

Willkommen!

Welcome!

Bienvenue !

Croeso!

¡Bienvenido/a!

Selamat datang!

Hoş geldin!

Benvenuto/a!

Добро пожаловать!

Välkommen

[APPLAUSE]

venue !

Willkommen,

/a!

Bem-vindo/a!

Hoş geldin!

Selamat datang!

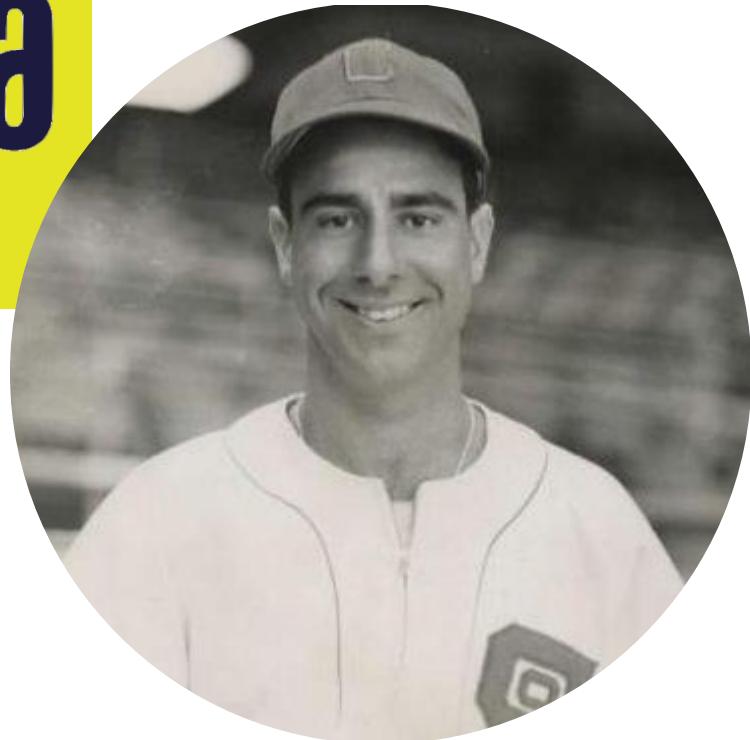
Willkommen!

WHAT ARE YOU
HERE TO TALK
ABOUT TODAY?

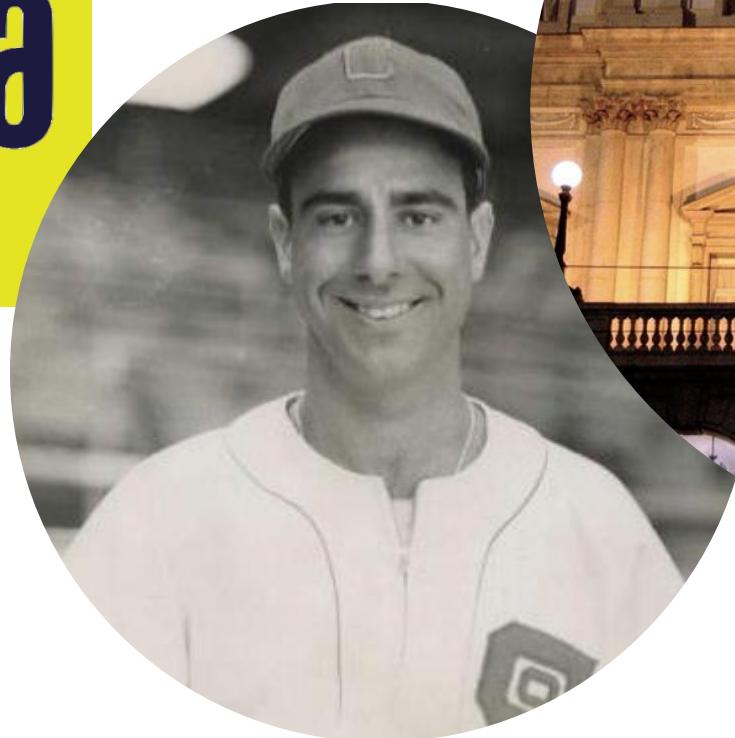
What are you here to talk about
today?



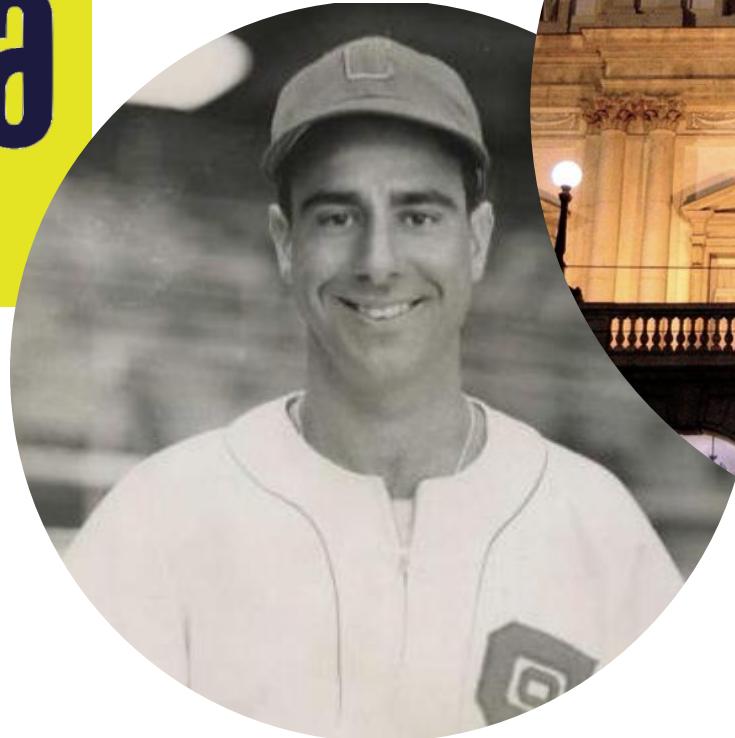
What are you here to talk about today?



What are you here to talk about today?

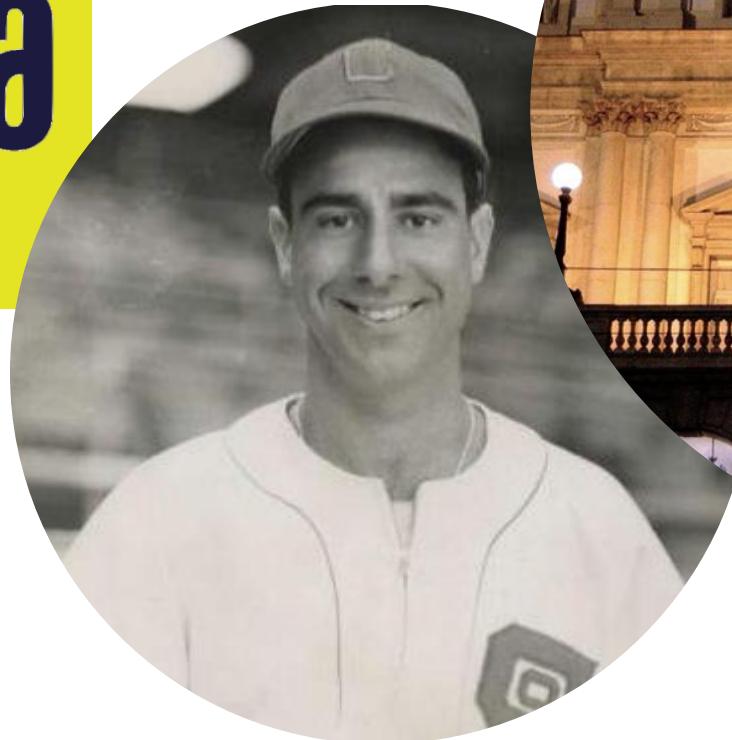


What are you here to talk about today?



Scala

What are you here to talk about today?



HOW DID YOU
GET HERE?

HOW DID YOU GET HERE?



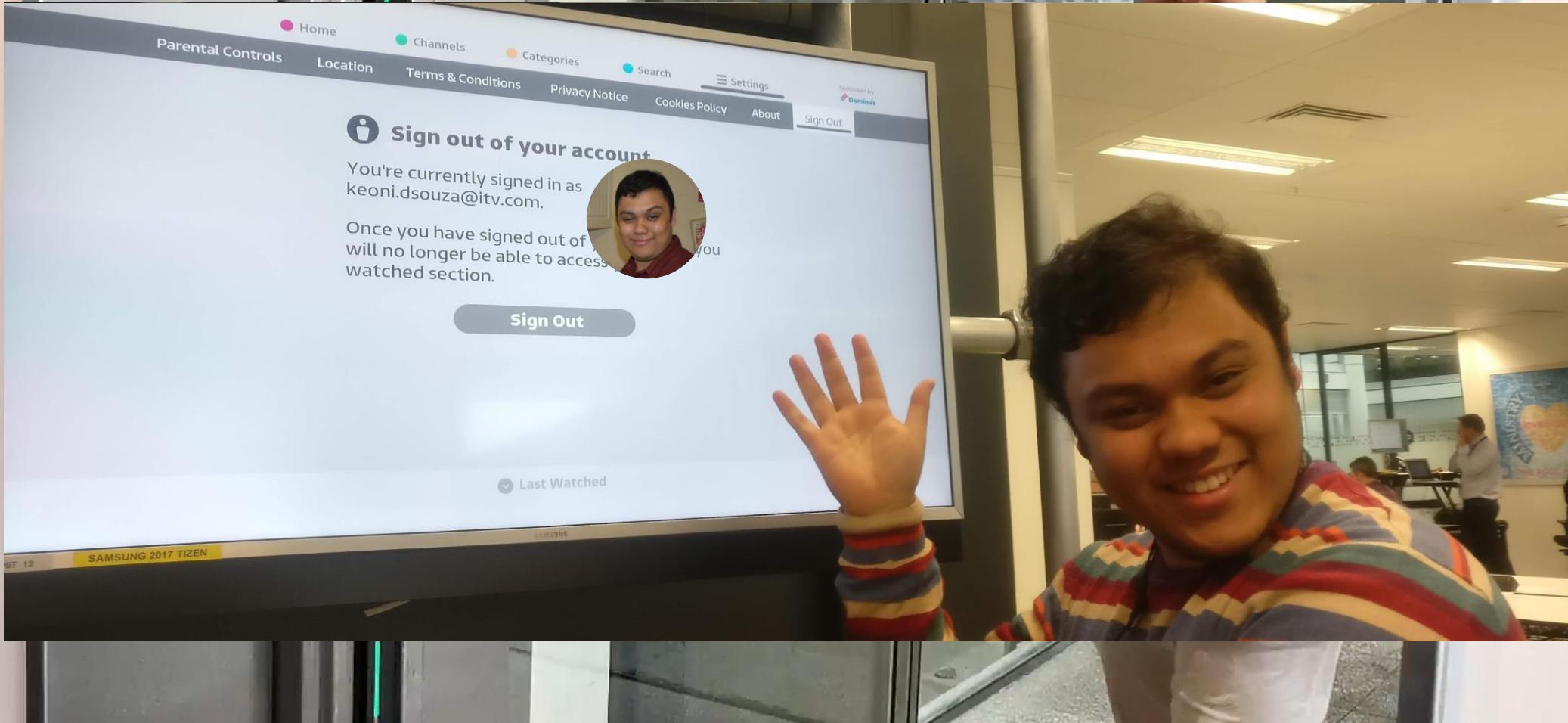
HOW DID YOU GET HERE?



HOW DID YOU GET HERE?

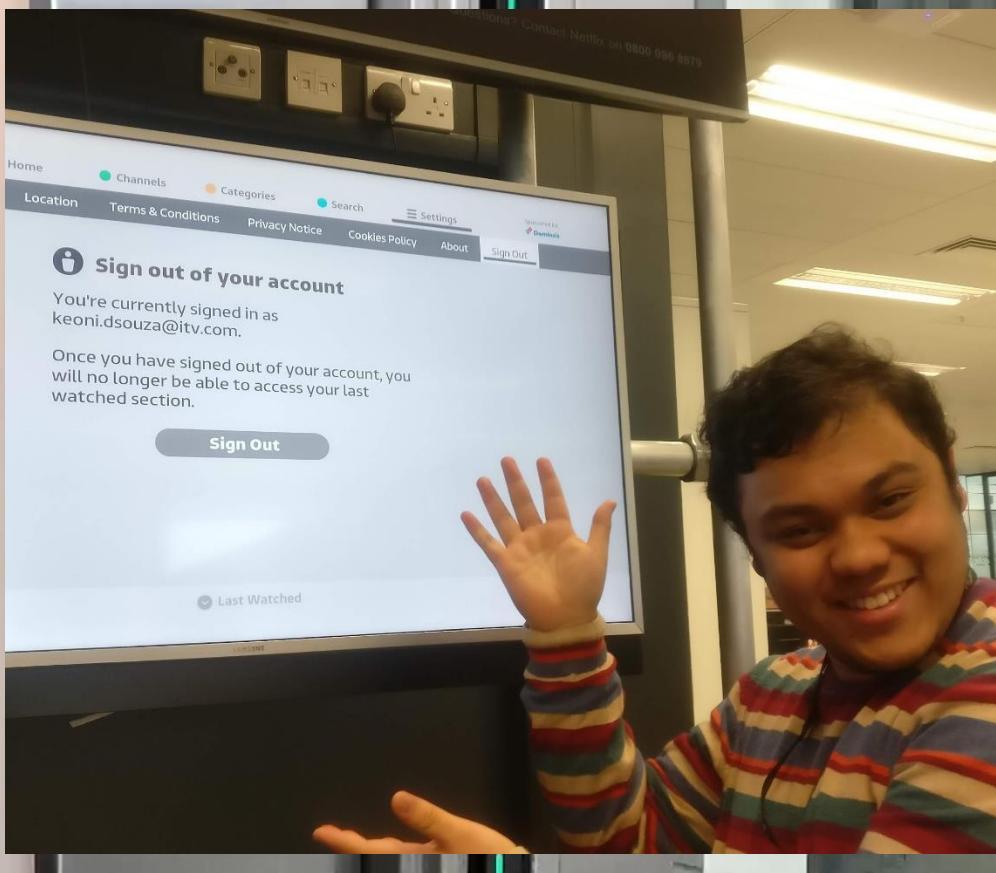
Project
motivations

Why are you here? Project motivations

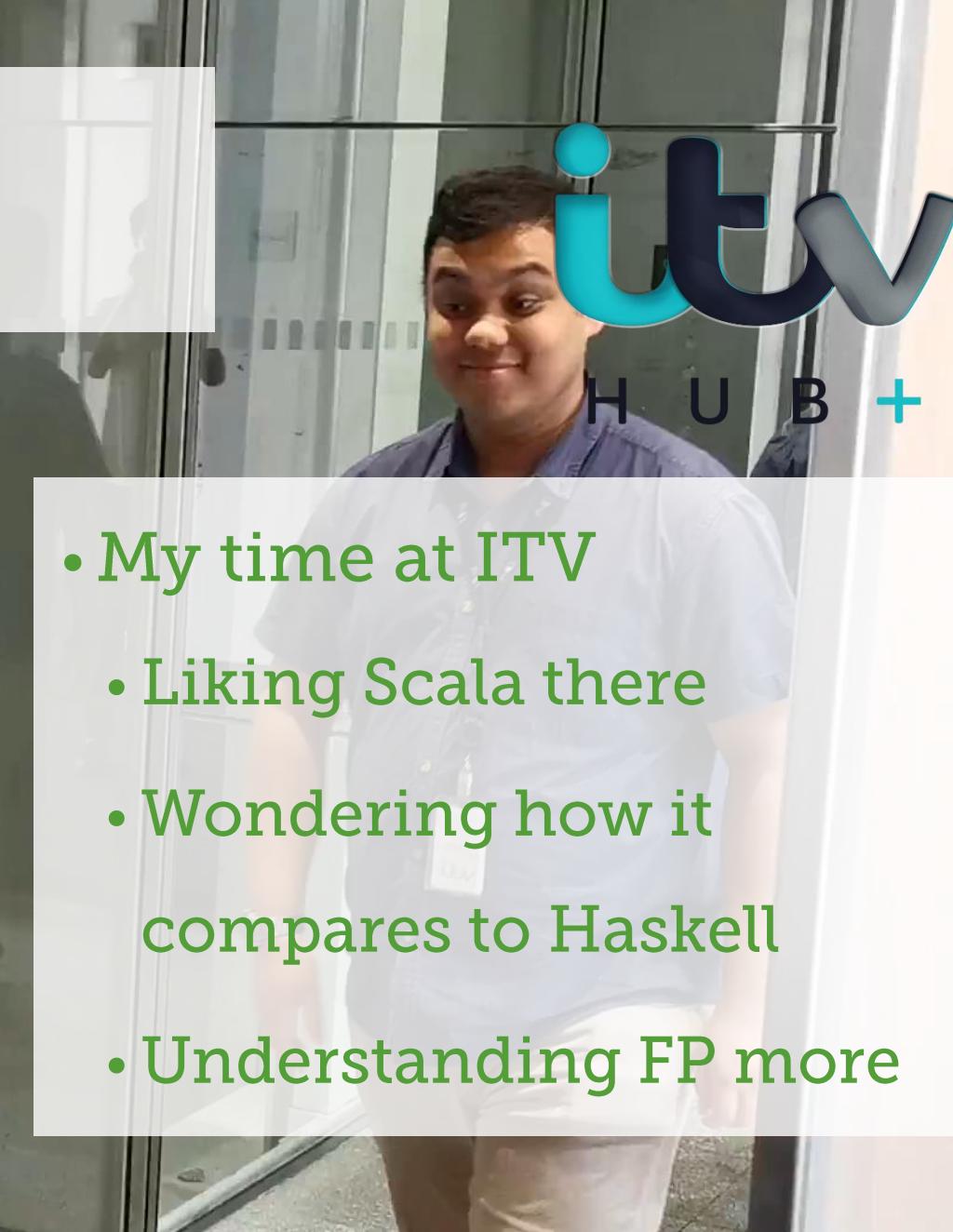


Why are you here?

Project motivations



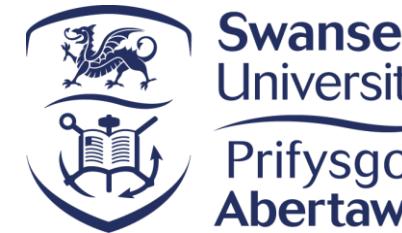
- My time at ITV
 - Liking Scala there
 - Wondering how it compares to Haskell
 - Understanding FP more

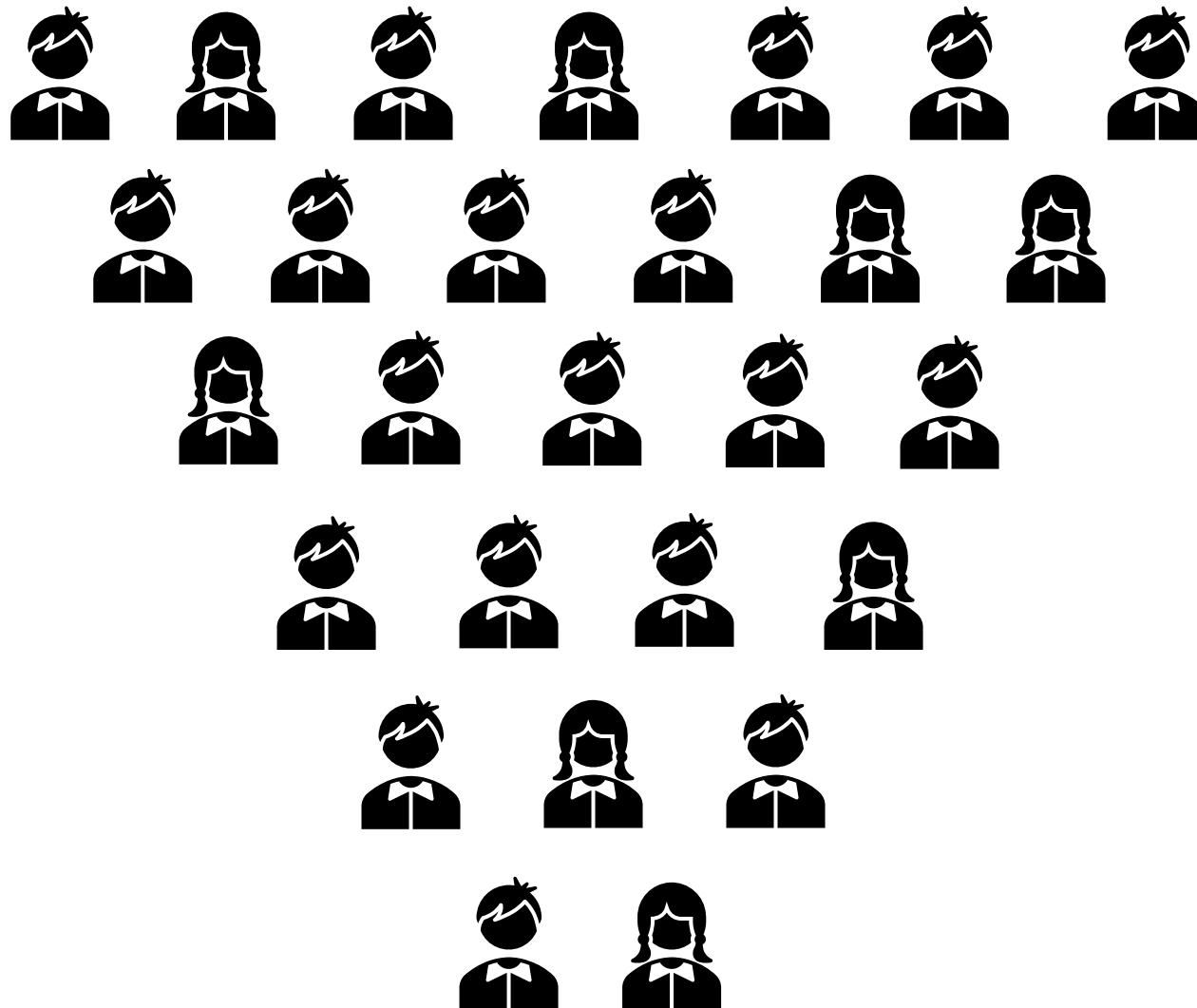


Why are you here?

Project motivations







itv
H U B

WHAT EVEN IS SCALA?

WHAT EVEN IS SCALA?

Background
on the
language

What even is Scala?

Background on the language

- Scala (scalable language)
 - small to the big
- Not a Java extension
 - Interoperability
 - Basic operators, data types and control structures shared

What even is Scala?

Background on the language

- Design started in 2001 at EPFL
- Better ways than Java
- Providing an alternative
- Precursors:
 - Pizza – moderately successful
 - Funnel – too academic
- First public release: 2003
- Scala v2.0: 2016

SHOW ME THE
GOOD STUFF...

SHOW ME THE
GOOD STUFF...

Programming
functionally
in Scala

Functions

- Function: a group of statements performing a task
- Scala has both functions and methods:
 - Method: part of a class with a name, signature, optionally some annotations, and some bytecode
 - Function: a complete object that can be assigned to a variable
 - *In other words, a function that's defined as a member of some object is called a method.*

Functions

- Functions are declared in the following form:

```
def functionName ([list of parameters]): [return type]
```

Functions

- Functions are defined in the following form:

```
def functionName ([list of parameters]): [return type] = {  
    function body  
    return [expr]  
}
```

- The return type could be any valid Scala data type and the parameter list will be a collection of variables separated by commas (both are optional).

Functions

```
object add {  
    def addInt(a: Int, b: Int): Int = {  
        var sum: Int = 0  
        sum = a + b  
        return sum  
    }  
}
```

Functions

```
object add {  
    def addInt(a: Int, b: Int): Int = {  
        var sum: Int = a + b  
        return sum  
    }  
}
```

Functions

```
object add {  
    def addInt(a: Int, b: Int): Int = {  
        return a + b  
    }  
}
```

Functions

```
object add {  
    def addInt(a: Int, b: Int): Int = return a + b  
}
```

Functions

```
object add {  
    def addInt(a: Int, b: Int): Int = a + b  
}
```

- Even cleaner!
- You don't even need the return keyword – the last value is automatically returned!
- Semicolons and braces are not always required.

Functions

- How would we write a function to calculate the square of a number?

Functions

- How would we write a function to calculate the square of a number?
- The math library has a built-in power function:

```
def pow(x: Double, y: Double): Double
```

where:

x is the base

y is the exponent

x^y is returned

that allows you to calculate the square easily:

```
scala.math.pow(n, 2).
```

Functions

- But, how would we describe one manually?

Functions

- But, how would we describe one manually?

```
// square: Int -> Int
def square(n: Int): Int = n * n
```

- The type annotations (the parts after the colons) don't necessarily have to be included because of Scala's built-in type inference.

Functions

- Functions that don't return anything are called procedures.
 - It doesn't actually return nothing – it returns a Unit, which is equivalent to Java's void.

```
object Hello {  
    def printMe(): Unit = {  
        println("Hello, Scala!")  
    }  
}
```

- What does this function do?

```
def iAmAFunction() = {  
    val name = scala.io.StdIn.readLine("Please enter  
your name: ")  
    println("Congratulations, " + name + " - you have  
been called to learn Scala!")  
}
```

Reading user input

- The `io.StdIn` library stands for standard input, allowing users to communicate with the keyboard and interact with Scala functions.
- What does this function do?

```
def iAmAFunction() = {  
    val name=scala.io.StdIn.readLine("Please enter your  
name: ")  
    println("Congratulations, " + name + " – you have  
been called to learn Scala!")  
}
```

Lists

mutable

immutable

Lists

mutability

you can change stuff

immutable

Lists

mutability

you can change stuff

immutability

stuff stays the same

Lists

mutability

you can change stuff

ListBuffer

immutability

stuff stays the same

Lists

mutability

you can change stuff

ListBuffer

immutability

stuff stays the same

List

Lists

- Creating a new ListBuffer instance:

```
var breads = new ListBuffer[String]()
```

```
import scala.collection.mutable.ListBuffer
```

Lists

- Adding to a ListBuffer instance:

```
breads += "Bagel"
```

```
breads ++= List("Baguette",  
                 "Boule",  
                 "Brioche")
```

```
import scala.collection.mutable.ListBuffer
```

Lists

- Removing from a ListBuffer instance:

```
breads -= "Brioche"
```

```
import scala.collection.mutable.ListBuffer
```

Lists

- Also:

```
breads remove 0 // the same as  
                breads.remove(0)
```

- Here, we're using postfix notation for `remove()` instead of brackets.
- The number refers to the position in the `ListBuffer`.

List operations

- `anyList.head` **returns the first element**
- `anyList.tail` **returns the list minus the first element**
- `anyList.isEmpty` **returns a Boolean asking if the list is empty**

Collection operations: map

```
val ints = anyList.map(s => s.toInt)
```

- The `map` operation takes a predicate and applies it to every element contained within the collection.
- It's part of the `TraversableLike` trait, so will work on all different types of collections.
- In this example, for each `s` it applies the `toInt` function to convert it into an integer.

Collection operations: filter

```
val ints = anyList.map(s => s.toInt).filter(_ % 2 == 0)
```

- The filter operation takes a predicate that returns a Boolean.
- If an element evaluates to true, it is returned. Falsely evaluated items are filtered out of the result.
- In this example, even numbers are returned. The underscore (_) symbol (wildcard) represents, in each case, the evaluated element.

Collection operations: flatten

```
val couples = List(  
    List("Keoni", "Louis"),  
    List("Richard", "Per"))  
val people = couples.flatten
```

Collection operations: flatten

```
val pairs = List(List("Keoni", "Louis"), List("Richard", "Per"))
val people = couples.flatten
```

- The flatten operation takes a collection of n dimensions and squashes it into $n - 1$ dimensions.
- It works, from the lowest degree, with two-dimension collections or higher.
- In this example, the pairs in pairs are flattened into an array containing all the peoples' names. Compare the 2-dimensional couples with the 1-dimensional people.

Collection operations: flatmap

- flatmap combines the flatten and map operations.

```
val pairs = List(List("Keoni", "Louis"),  
                List("Richard", "Per"))  
val people = couples.flatmap(_ + " Herrey")
```

- It is syntactic sugar for:

```
val people = couples.flatten.map(_ + " Herrey")
```

- In this example, for each person it adds the same surname.





WILL THIS WORK
FOR PEOPLE LIKE
US?

WILL THIS WORK FOR PEOPLE LIKE US?

Teaching
Scala to
students

Will this work for people like us?

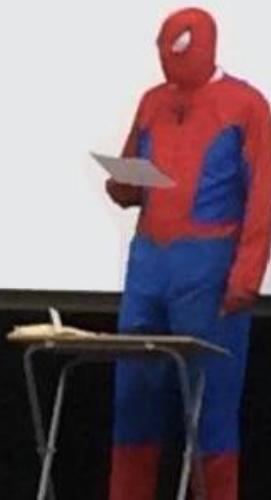
Teaching Scala to students

- Multi-paradigm – there's leeway
- Elegant – you can write beautiful code
- You could earn, on average, £85,000
- Big companies use the language
- Second most in-demand last year (8.5 interviews offered over a 2-6-week period)

IS THAT IT?

Rounding
off the
presentation

Gonna tell my kids
this was a great
Scala presentation



That's it
from me –
for now...!

1

Part I/ Writing
and reading in
Scala

2

Part II/ Lists and
higher order
functions in
Scala

3

Part III/
Interacting Scala
with Java

Lab time! Log in and go to
keonidsouza.com/scala