A scalable language

INTRODUCTION TO SCALA

David Venturi
Curriculum Manager, DataCamp

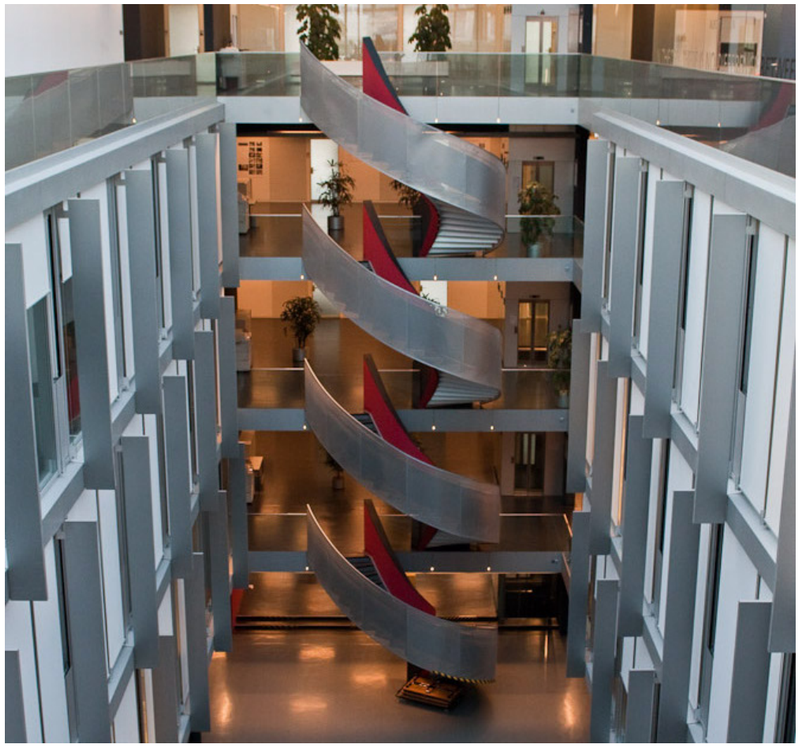
Learn by doing

- What is Scala?
- Why use Scala?
- Who uses Scala?

[1] http://bit.ly/twenty_one_wikipedia

Staircase to mastery

The Scala logo



EPFL, Computer Science Building by Miles Sabin

[1] https://www.flickr.com/photos/montpelier/3957416434/

What is Scala?

> Scala is a general-purpose programming language providing support for functional programming and a strong static type system. Designed to be concise, many of Scala's design decisions aimed to address criticisms of Java.

[1] http://bit.ly/scala_wikipedia

What is Scala?

> Scala is a general-purpose programming language providing support for functional programming and a strong static type system. Designed to be concise, many of Scala's design decisions aimed to address criticisms of Java.

> **Scala source code** is intended to be compiled to Java bytecode, so that the resulting executable code **runs on a Java virtual machine**.

[1] http://bit.ly/jvm_wikipedia

Why use Scala?

SCAlable **LA**nguage

NETFLIX

Morgan Stanley

airbnb

Deutsche Bank

The cathedral vs. the bazaar

Cathedral

Bazaar

http://bit.ly/cathedral-bazaar-book

[1] Canterbury Cathedral by WyrdLight.com

Flexible and convenient

Flexible

- Scala lets you add new types, collections, and control constructs that feel like they are built-in to the language

Convenient

- The Scala standard library has a set of convenient predefined types, collections, and control constructs

Who uses Scala?

Roles

Software Engineer

Data Engineer

Data Scientist

Machine Learning Engineer

Industries

Finance

Tech

Healthcare

So many more...

Who uses Scala?

Roles

Software Engineer

Data Engineer

Data Scientist

Machine Learning Engineer

Industries

Finance

Tech

Healthcare

So many more...

Who uses Scala?

[1] https://insights.stackoverflow.com/survey/2019

Let's practice!

INTRODUCTION TO SCALA

Scala code and the Scala interpreter

INTRODUCTION TO SCALA

David Venturi
Curriculum Manager, DataCamp

datacamp

What is Scala?

> Scala is a general-purpose programming language providing support for functional programming and a strong static type system. Designed to be concise, many of Scala's design decisions aimed to address criticisms of Java.

> **Scala source code** is intended to be compiled to Java bytecode, so that the resulting executable code **runs on a Java virtual machine**.

What is Scala?

> Scala combines object-oriented and functional programming in one concise, high-level language. Scala's static types help avoid bugs in complex applications, and its JVM and JavaScript runtimes let you build high-performance systems with easy access to huge ecosystems of libraries.

https://www.scala-lang.org/

Scala fuses OOP and FP

> Scala combines object-oriented and functional programming

Scala fuses OOP and FP -> Scala is scalable

> Scala combines object-oriented and functional programming

Scala fuses OOP and FP -> Scala is scalable

> Scala combines object-oriented and functional programming

Scala is object-oriented

- Every value is an object
- Every operation is a method call

```
val sumA = 2 + 4
```

```
sumA: Int = 6
```

Scala fuses OOP and FP -> Scala is scalable

> Scala combines object-oriented and functional programming

Scala is object-oriented

- Every value is an object
- Every operation is a method call

```
val sumA = 2.+(4)
```

```
sumA: Int = 6
```

Scala fuses OOP and FP -> Scala is scalable

> Scala combines object-oriented and functional programming

Scala is functional

1. Functions are first-class values

Scala fuses OOP and FP -> Scala is scalable

> Scala combines object-oriented and functional programming

Scala is functional

1. Functions are first-class values
2. Operations of a program should map input values to output values rather than change data in place

More answers to "Why use Scala?"

More answers to "Why use Scala?"

> Scala combines object-oriented and functional programming in one **concise**

More answers to "Why use Scala?"

> Scala combines object-oriented and functional programming in one concise, **high-level language.**

More answers to "Why use Scala?"

> Scala combines object-oriented and functional programming in one concise, high-level language. **Scala's static types help avoid bugs in complex applications**

More answers to "Why use Scala?"

> Scala combines object-oriented and functional programming in one concise, high-level language. Scala's static types help avoid bugs in complex applications, and **its JVM and JavaScript runtimes let you build high-performance systems with easy access to huge ecosystems of libraries.**

The Scala interpreter

```
$ scala
```

```
Welcome to Scala 2.12.7.
Type in expressions for evaluation. Or try :help.
```

```
scala>
```

[1] http://bit.ly/scala_repl

The Scala interpreter

```
$ scala
```

```
Welcome to Scala 2.12.7.
Type in expressions for evaluation. Or try :help.
```

```
scala> 2 + 3
```

```
res0: Int = 5
```

```
res0 * 2
```

```
res1: Int = 10
```

[1] http://bit.ly/scala_repl

The Scala interpreter

```
scala> println("Let's play Twenty-One!")
```

```
Let's play Twenty-One!
```

[1] http://bit.ly/scala_repl

Let's practice!

INTRODUCTION TO SCALA

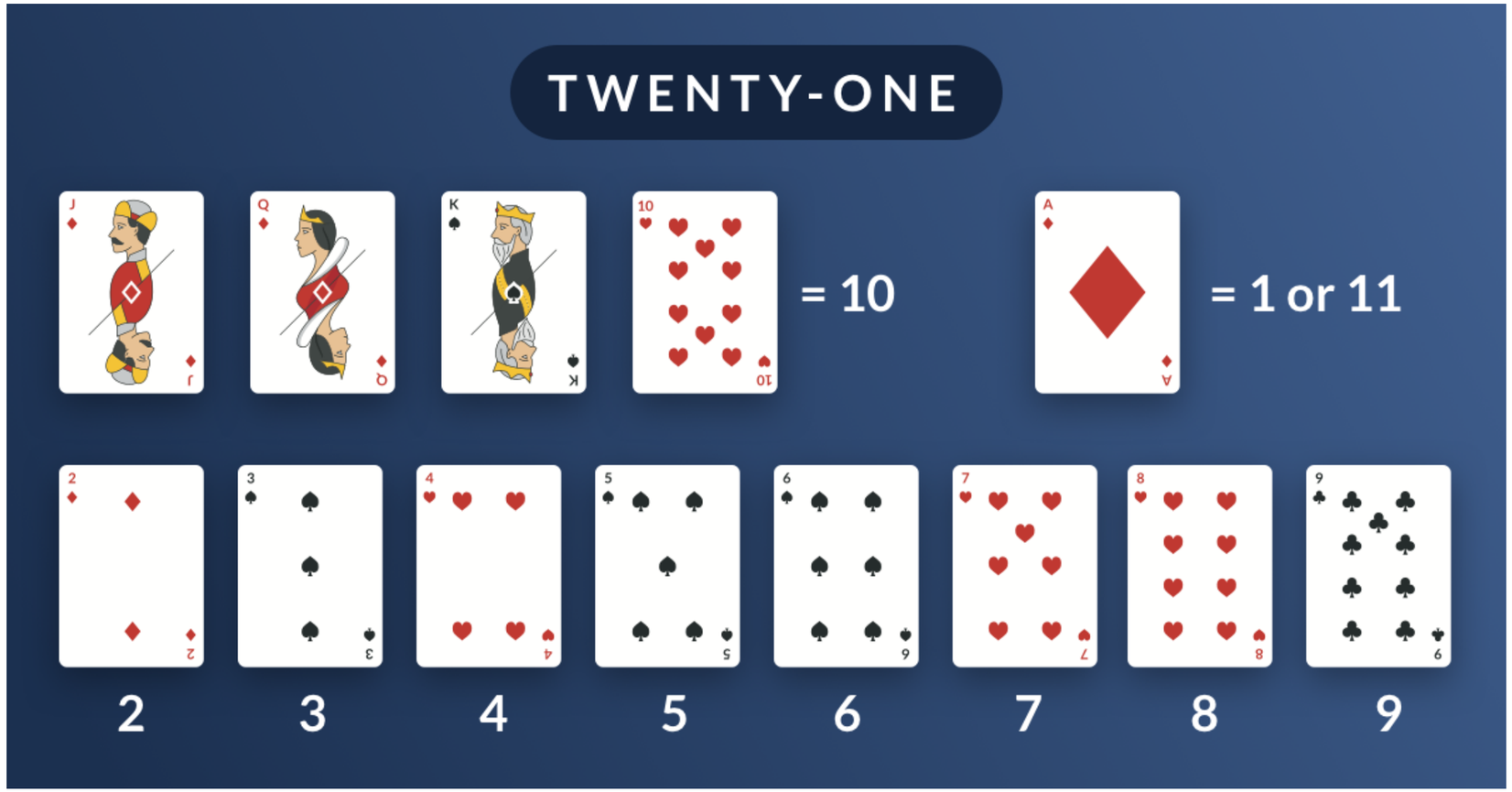
Immutable variables (val) and value types

INTRODUCTION TO SCALA

David Venturi
Curriculum Manager, DataCamp

datacamp

The game of Twenty-One



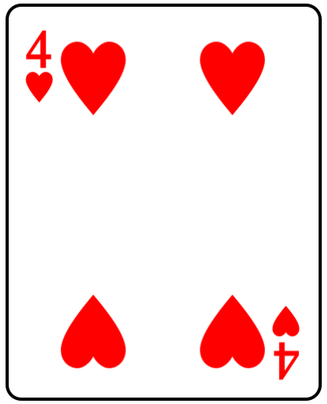
[1] http://bit.ly/twenty_one_wikipedia

Scala has two kinds of variables

`val` (immutable)

`var` (mutable)

- can't be reassigned



```
scala> val fourHearts: Int = 4
```

```
fourHearts: Int = 4
```

Reassigning a val produces an error

```
scala> val fourHearts: Int = 4
```

```
fourHearts: Int = 4
```

```
scala> fourHearts = 5
```

```
<console>:12: error: reassignment to val
       fourHearts = 5
                  ^
```

Scala value types

- `Double`
- `Float`
- `Long`
- `Int`
- `Short`
- `Byte`
- `Char`
- `Boolean`
- `Unit`

Scala value types

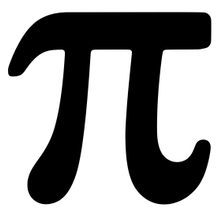
Most common types for data-related tasks:

- `Double`
- `Int`
- `Boolean`
- `String`

Double

- 64-bit IEEE-754 double-precision floating point number
- 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative)

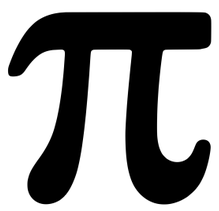
Double



```
scala> val piDouble: Double = 3.14
```

```
piDouble: Double = 3.14
```

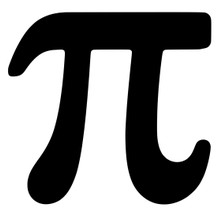
Double



```
scala> val piFloat: Float = 3.14
```

```
<console>:11: error: type mismatch;
 found   : Double(3.14)
 required: Float
       val piFloat: Float = 3.14
```

Double is more precise than Float



```
scala> val piDouble: Double = 3.14159265358979323846264338
```

```
piDouble: Double = 3.141592653589793
```

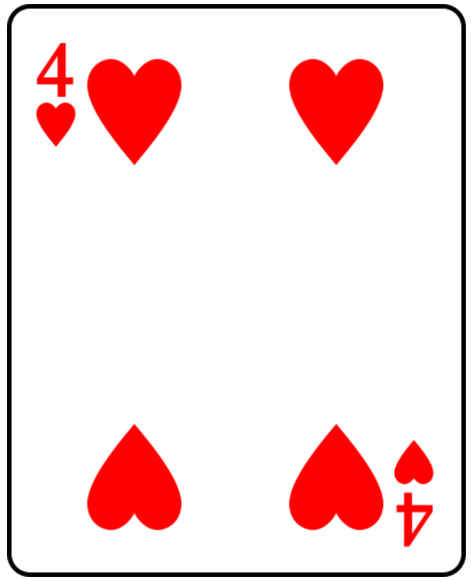
```
scala> val piFloat: Float = 3.14159265358979323846264338327
```

```
piFloat: Float = 3.1415927
```

Int

- 32-bit signed integer
- -2^31 to 2^31-1, inclusive
- -2,147,483,648 to 2,147,483,647

Int

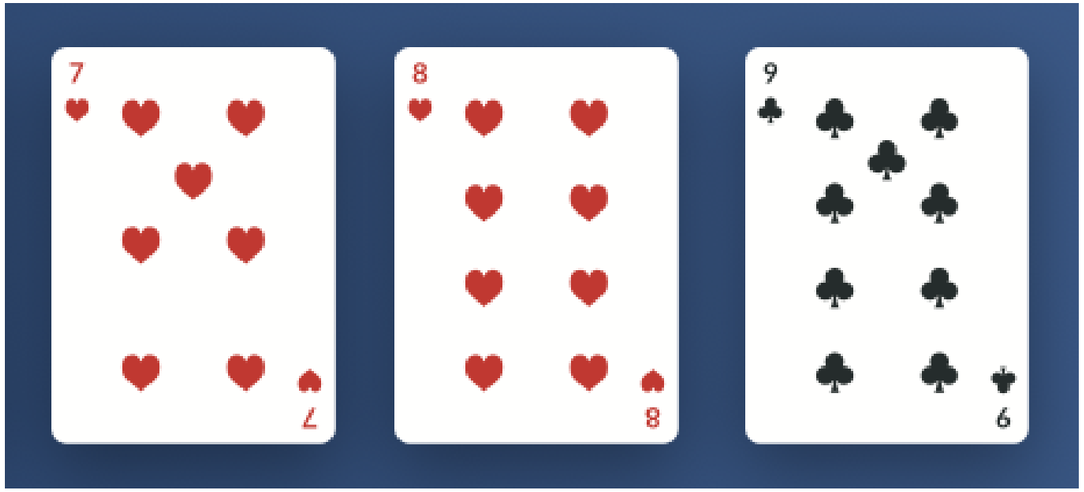


```
scala> val fourHearts: Int = 4
```

```
fourHearts: Int = 4
```

Boolean

- true or false



$$= 24$$

$$(\text{bust!})$$

```
scala> val handBusts: Boolean = true
```

```
handBusts: Boolean = true
```

Char and String

Char

- 16-bit unsigned Unicode integer
- 0 to 2^16-1, inclusive
- 0 to 65,535

String

- `String` : a sequence of `Char`

```
scala> val symbolAceSpades: String = "A♠"
```

```
symbolAceSpades: String = A♠
```

Scala value types

- `Double`
- `Float`
- `Long`
- **`Int`**
- `Short`
- `Byte`
- `Char`
- `Boolean`
- `Unit`

Scala value types

- `scala.Double`
- `scala.Float`
- `scala.Long`
- **`scala.Int`**
- `scala.Short`
- `scala.Byte`
- `scala.Char`
- `scala.Boolean`
- `scala.Unit`

Scala value types

```
scala> val fourHearts: Int = 4
```

```
fourHearts: Int = 4
```

```
scala> val fiveHearts: scala.Int = 5
```

```
fiveHearts: Int = 5
```

Scala value types have equivalent Java types

Scala types

- `scala.Double`
- `scala.Float`
- `scala.Long`
- `scala.Int`
- `scala.Short`
- `scala.Byte`
- `scala.Char`
- `scala.Boolean`
- `scala.Unit`

Java types

- `java.lang.Double`
- `java.lang.Float`
- `java.lang.Long`
- `java.lang.Integer`
- `java.lang.Short`
- `java.lang.Byte`
- `java.lang.Character`
- `java.lang.Boolean`

Choosing the right type

- `Double`
- `Float`
- `Long`
- `Int`
- `Short`
- `Byte`
- `Char`
- `Boolean`
- `Unit`

Let's practice!

INTRODUCTION TO SCALA

Mutable variables (var) and type inference

INTRODUCTION TO SCALA

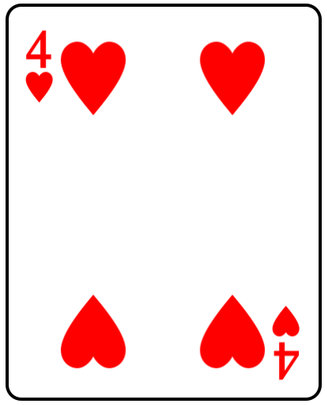
David Venturi
Curriculum Manager, DataCamp

datacamp

Scala has two kinds of variables

`val` (immutable)

- can't be reassigned

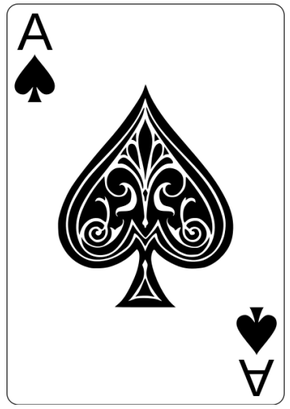


```
scala> val fourHearts: Int = 4
```

```
fourHearts: Int = 4
```

`var` (mutable)

- can be reassigned



```
scala> var aceSpades: Int = 1
```

```
aceSpades: Int = 1
```

Reassigning a val produces an error

```
scala> val fourHearts: Int = 4
```

```
fourHearts: Int = 4
```

```
scala> val fourHearts = 5
```

```
<console>:12: error: reassignment to val
       fourHearts = 5
                    ^
```

Reassigning a var works

```
scala> var aceSpades: Int = 1
```

```
aceSpades: Int = 1
```

```
scala> aceSpades = 11
```

```
aceSpades: Int = 11
```

Pros and cons of immutability

Pros

- Your data won't be changed inadvertently
- Your code is easier to reason about
- You have to write fewer tests

Cons

- More memory required due to data copying

Scala nudges us towards immutability

immutability

Type inference is powerful

```
scala> val fourHearts: Int = 4
```

```
fourHearts: Int = 4
```

```
scala> var aceSpades: Int = 1
```

```
aceSpades: Int = 1
```

```
scala> val fourHearts = 4
```

```
fourHearts: Int = 4
```

```
scala> var aceSpades = 1
```

```
aceSpades: Int = 1
```

Semicolons

```
scala> val fourHearts = 4
```

```
fourHearts: Int = 4
```

```
scala> var aceSpades = 1
```

```
aceSpades: Int = 1
```

Semicolons

```
scala> val fourHearts = 4;
```

```
fourHearts: Int = 4
```

```
scala> var aceSpades = 1;
```

```
aceSpades: Int = 1
```

Semicolons

```
scala> val fourHearts = 4
```

```
fourHearts: Int = 4
```

```
scala> var aceSpades = 1
```

```
aceSpades: Int = 1
```

Let's practice!

INTRODUCTION TO SCALA