

# Scala Introduction

---

Jens Grassel

May 16, 2019

Wegtam GmbH

# Environment and Tooling

---

Scala runs on the JVM, but not only there!

- Scala.js for Javascript endpoints
- Scala Native for LLVM endpoints

- multiple build tools are available
  - SBT
  - Mill
  - CBT
  - Fury
- SBT has the biggest ecosystem and is widely used
- stick to SBT but maybe checkout Mill ;-)

# SBT Setup

SBT is usually setup on a per project basis via the `build.sbt` file. However here are some nice settings for the global configuration, put it into `~/.sbt/1.0/global.sbt`

```
// Prevent Strg+C from killing sbt.
cancelable in Global := true
// Coloured console.
initialize ~ = ( _ =>
  if (ConsoleLogger.formatEnabled)
    sys.props("scala.color") = "true"
)
```

Some useful global plugins can be enabled by adding them to the file  
`~/.sbt/1.0/plugins/plugins.sbt`

```
addSbtPlugin("net.virtual-void" % "sbt-dependency-graph" % "0.9.2")  
addSbtPlugin("com.timushev.sbt" % "sbt-updates" % "0.4.0")
```

There are several options available but beginners should maybe stick to a full blown IDE which in the case of Scala means IntelliJ IDEA with the Scala plugin.

For the confident:

- VS-Code with scala-metals
- Vim or Neovim with vim-scala plugin and optionally scala-metals
- Emacs
- Atom
- possibly others. . .

# Basics

---



**Table 1:** Values types in Scala

Type	Notes
Immutable	Data structures which cannot be changed.
Mutable	Data structures which can be changed.
Val	A variable that cannot be re-assigned.
Var	A variable that can be re-assigned (changed).

**Table 2:** When to use which type?

Definition	Notes
Immutable Val	The recommended way to go.
Immutable Var	Okay if used in a local scope.
Mutable Val	Try not to use this but there may be applications. <sup>1</sup>
Mutable Var	<b>Never ever do this!</b>

---

<sup>1</sup>However **if** you do this then **never** pass the value around!

As Scala provides tail recursion you should try to use it if possible.

**But...**

Please not simply for the sake of using it!

Evaluate if it is necessary.<sup>2</sup>

## **Remember**

Readability and maintainability trump obscure performance gains every time!

---

<sup>2</sup>Some algorithm are hard or impossible to do with tail recursion!

# Functions

Functions are there to make your life easier!

- Use HOF<sup>3</sup>
- Use Currying
- Use polymorphism

For additional benefit try to stick with **pure** functions. A pure function:

- is total (an output for every input)
- is free of side effects
- its output does only depend on its input

---

<sup>3</sup>Higher Order Functions

# Implicit

---

# What are implicits?

An **implicit** value can be used by the compiler to pass it to any function which depends on an implicit parameter of the same type.

- reduce boiler plate (implicit parameters)
- extend existing types with custom functions (wrapper classes)
- convert types implicitly (**Do not do this!**)

# Best practices

- always specify the type of implicit `val` or `def`
- do not to use implicits for simple datatypes (primitives)
- stick to the naming conventions e.g. `FooOps` when extending `Foo`
- put extension wrappers into `syntax` objects
- Do not use implicit conversions!



# Objects, Classes and Traits

---

