Scala

Veel functionaliteit voor weinig code



Agenda

- eenvoudige statements
- list
- class / object
- trait
- optional and match
- scala en java

Agenda

- eenvoudige statements
- list
- class / object
- trait
- optional and match
- scala en java

- object-oriented
- functional programming
- statically typed language
- function value is an object

```
val favoriteNumber: Int = 31

val name: String = "Wiggert"
val nameWithoutType = "Wiggert"

println(name)
```

```
val name: String = "Wiggert"
// no reassignment
name = "Robin"
```

> reassignment to val

```
var name = "Wiggert"
name = "Robin"
```

```
// A function
def max(x: Int, y: Int): Int = {
  if (x > y) \times else y
// variables have types
// return type
// = sign: function defines an
expression that results in a value
```

```
def max(x: Int, y: Int): Int = {
  if (x > y) x else y
}

// return type may be omitted

def max2(x: Int, y: Int) = {
  if (x > y) x else y
}
```

Try It Yourself 1

Agenda

- eenvoudige statements
- list
- class / object
- trait
- optional and match
- scala en java

```
val numbers = List(5, 6, 7, 8, 9)
```

```
val numbers = List(5, 6, 7, 8, 9)
// Lists are always immutable
// adding an element creates a new
list which you can assign
val moreNumbers = 4 :: numbers
> moreNumbers = List(4, 5, 6, 7, 8, 9)
```

```
val numbers = List(5, 6, 7, 8, 9)
// Methods of list:
numbers (0)
> 5
numbers.filter(i => i > 6)
> List(7, 8, 9)
numbers.count(i => i < 6)</pre>
> 1
```

```
val numbers = List(5, 6, 7, 8, 9)
numbers. head
> 5
numbers.tail
> List(6, 7, 8, 9)
numbers init
> List(5, 6, 7, 8)
numbers.last
```

```
val numbers = List(5, 6, 7, 8, 9)

numbers.map(i => i + 1)
> List(6, 7, 8, 9, 10)

// empty list
Nil
```

```
val numbers = List(5, 6, 7, 8, 9)

// underscore notation
numbers.filter(i => i > 6)
> List(7, 8, 9)

numbers.filter(_ > 6)
> List(7, 8, 9)
```

Try It Yourself 2

Agenda

- eenvoudige statements
- list
- class / object
- trait
- optional and match
- scala en java

```
// method parameters always val
// fields assign:
// val or var or <nothing> (= private)
class Person(name: String, val age: Int) {
  def introduceYourself() = {
    "Hello. My name is "
      + name + " and I am " + age
      + " years old."
```

```
class Person(name: String, val age: Int) {
  def introduceYourself() = {
    "Hello. My name is "
      + name + " and I am " + age
      + " years old."
val wiggert = new Person("Wiggert", 29)
wiggert.age
> 29
```

```
class Person(name: String, val age: Int) {
  def introduceYourself() = {
    "Hello. My name is "
      + name + " and I am " + age
      + " years old."
val wiggert = new Person("Wiggert", 29)
wiggert name
> Cannot resolve symbol name
```

```
class Person(name: String, val age: Int) {
  def introduceYourself() = {
    "Hello. My name is "
      + name + " and I am " + age
      + " years old."
val wiggert = new Person("Wiggert", 29)
wiggert.introduceYourself()
> "Hello. My name is Wiggert and I am 29
years old."
```

```
// No statics in class. Use 'Object'
object Calculator {
   def max(x: Int, y: Int) = {
      if (x > y) x else y
   }
}
Calculator.max(3, 5)
> 5
```

```
// Special class
case class Person(name: String, age: Int)
// Like a pojo
// By default: all fields are val
// By default: equals, hashcode, toString
```

Try It Yourself 3

Agenda

- eenvoudige statements
- list
- class / object
- trait
- optional and match
- scala en java

```
trait Happy {
  def sing() = {
   println("Ik ben vandaag zo vrolijk...")
class Developer extends Happy
val d = new Developer
d.sing
> Ik ben vandaag zo vrolijk...
```

```
trait Happy {
 def sing(): Unit = {
   println("Ik ben vandaag zo vrolijk...")
class Employee
class JavaDeveloper extends Employee with Happy
val d = new JavaDeveloper
d.sing
> Ik ben vandaag zo vrolijk...
```

- lijkt op abstract class
- lijkt op een interface
- methode en field definitions 'mix' je in een class
- 'thin' interface verrijken

```
class Rational(val numer: Int, val denom: Int)
val twoThird = new Rational(2, 3)

// 2
// ---
// 3
```

```
class Rational(val numer: Int, val denom: Int) {
    def < (that: Rational) =
        this.numer * that.denom > that.numer * this.denom
    def > (that: Rational) = that < this
    def <= (that: Rational) = (this < that) || (this == that)
    def >= (that: Rational) = (this > that) || (this == that)
}
```

• maar > en < logica staat los van Rational op zich

```
package scala.math
trait Ordered[A] extends scala.Any with java.lang.Comparable[A] {
    def compare(that : A) : scala.Int
    def <(that : A) : scala.Boolean = { /* compiled code */ }
    def >(that : A) : scala.Boolean = { /* compiled code */ }
    def <=(that : A) : scala.Boolean = { /* compiled code */ }
    def >=(that : A) : scala.Boolean = { /* compiled code */ }
    def compareTo(that : A) : scala.Int = { /* compiled code */ }
}

object Ordered extends scala.AnyRef {
    implicit def orderingToOrdered[T](x : T)(implicit ord :
        scala.math.Ordering[T]) : scala.math.Ordered[T] = { /* compiled code */ }
}
```

Try It Yourself 4

Agenda

- eenvoudige statements
- list
- class / object
- trait
- optional and match
- scala en java

```
val capitals: Map[String, String] = Map(
   "France" -> "Paris",
   "Japan" -> "Tokyo"
)
```

```
// Option[Type]
// None

val name1: Option[String] = Some("Wiggert")

val name2: Option[String] = None
```

```
def show1(x: Option[String]) = {
  if (x.isDefined) {
    x.get
  } else {
    "?"
  }
}
```

```
val capitals = Map(
  "France" -> "Paris",
  "Japan" -> "Tokyo"
def show1(x: Option[String]) = {
  if (x.isDefined) x.get else "?"
show1(capitals.get("France"))
> "Paris"
show1(capitals.get("Japan"))
> "Tokyo"
show1(capitals.get("Holland"))
```

```
val capitals = Map(
  "France" -> "Paris",
  "Japan" -> "Tokyo"
def show2(x: Option[String]) = x match {
  case Some(s) => s
  case None => "?"
show2(capitals.get("France"))
> "Paris"
show2(capitals.get("Japan"))
> "Tokyo"
show2(capitals.get("Holland"))
> "?"
```

```
// Why?

// Option[String] more clear it can be None,
than String can be null

// Using a value before checking null: is
now a type error
```

```
// List, functions, pattern matching and
tail recursion

def sum(list: List[Int]): Int = list match {
   case Nil => 0
   case head :: tail => head + sum(tail)
}

sum(List(1, 2, 3, 4))
> 10
```

Try It Yourself 5

Agenda

- eenvoudige statements
- list
- class / object
- trait
- optional and match
- scala en java

```
// All code on jvm, so scala and java class can call
each other!
// Maven to include scala:
<dependency>
   <groupId>org.scala-lang
   <artifactId>scala-library</artifactId>
   <version>2.11.7
</dependency>
<dependency>
   <groupId>org.scalatest
   <artifactId>scalatest 2.11</artifactId>
   <version>2.2.6</version>
   <scope>test</scope>
</dependency>
```

```
// All code on jvm, so scala and java class can call
each other!
def something(names: java.util.List[String]) = {
 // do something with names
def useScalaInJavaEnvironmentWrong() = {
  val scalaEnthusiasts: List[String] =
    List("Robin", "Wiggert")
  something(scalaEnthusiasts)
> Type mismatch, expected: util.List[String],
actual: List[String]
```

```
def something(names: java.util.List[String]) = {
   // do something with names
def useScalaInJavaEnvironmentCorrect() = {
  import scala.collection.JavaConverters._
  val scalaEnthusiasts: List[String] =
   List("Robin", "Wiggert")
  something(scalaEnthusiasts.asJava)
```

Try It Yourself 6

```
def javaFunctionWithInts(
  javaInts: java.util.List[java.lang.Integer]) = {
 // do something with the ints
def useScalaInJavaEnvironmentExtended() = {
 // watch out for type of element in list
  import scala.collection.JavaConverters.__
 val scalaNumbers: List[Int] = List(1, 2, 3, 4, 5, 6)
  javaFunctionWithInts(scalaNumbers.asJava)
> Type mismatch, expected: util.List[Integer], actual:
util.List[Int]
```

```
def javaFunctionWithInts(
  javaInts: java.util.List[java.lang.Integer]) = {
 // do something with the ints
def useScalaInJavaEnvironmentExtended() = {
 // watch out for type of element in list
  import scala.collection.JavaConverters._
  val scalaNumbers: List[Int] = List(1, 2, 3, 4, 5, 6)
 // now you have to do some magic:
  javaFunctionWithInts(
    scalaNumbers.map(i => i: java.lang.Integer).asJava)
```

Try it Yourself 7

Learn You a Haskell for Great Good!

ragmatic rogrammer

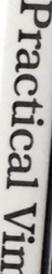


Hunt • Thomas

Addison Wesley







Neil



Seven Languages in Seven Weeks Tate



GROWING OBJECT-ORIENTED SOFTWARE

FREEMAN | PRYCE









Веск





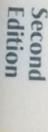
Rios & Hardin Dhanjani



HALL



Martin



Bloch

Effective Java







#

HALL



Development Software

Pilone





Head First

떊 JSP Servlets

2nd Edition

Covers J2EE 1.5

Basham.

Sierra

& Bates

& Miles





EXAM 310-065

Programmer for Java 6 Study Guide

Sierra

Bates

Sun Certified

Programming in

Venners Odersky Spoon

artima

Second Edition

O REILLY'

O'REILLY"