Introduction to programming using Scala

– Experiences from 1st year of new course for CSE students

Björn Regnell

Dept. of Computer Science, LTH
Lund University, Sweden

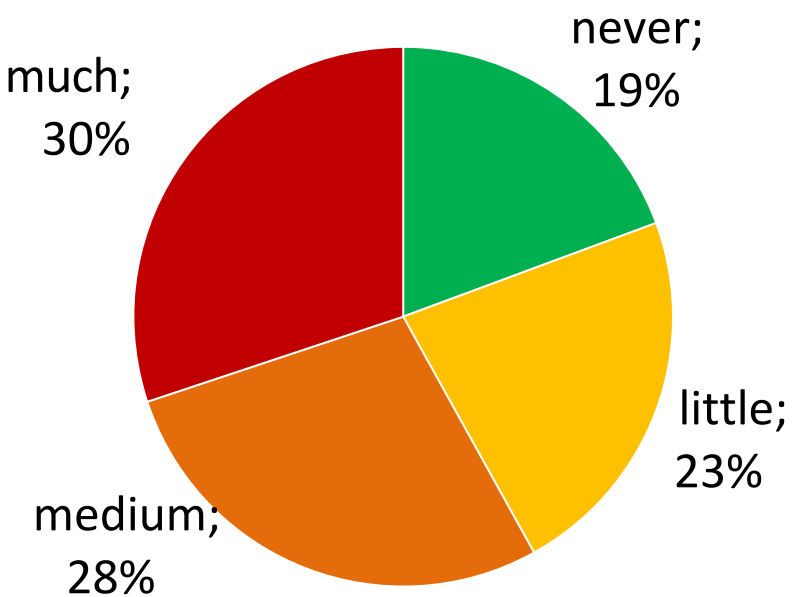
2016 April 20

Agenda

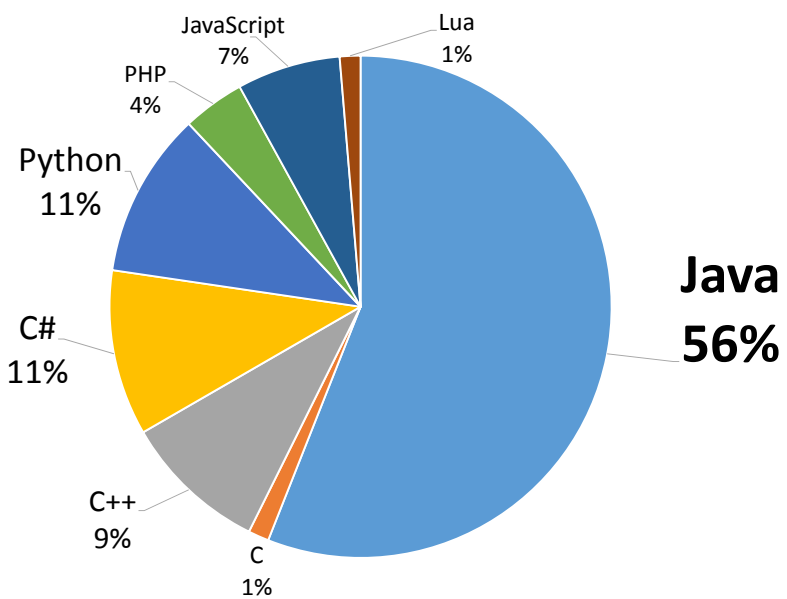
1. Background
   - Why a new course?
   - Why Scala?
2. Overview of new course at LTH: EDAA45
3. Experiences of new course at LTH: EDAA45
4. A brief intro to Scala

Course intro survey 2015, CSE (Datateknik) EDA016 Programming, first course (Java)

Have programmed before?

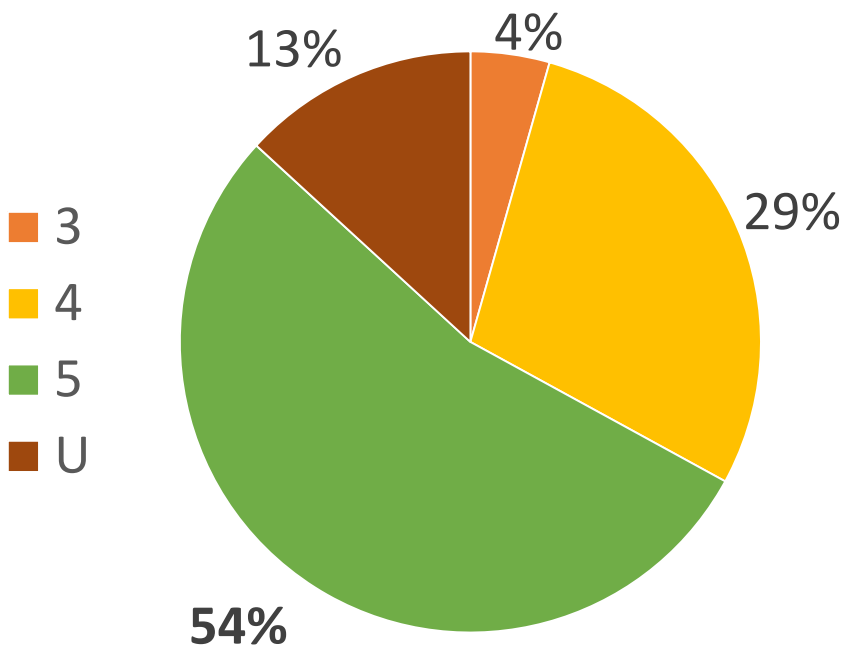


What language?



Course results 2015, CSE (Datateknik) EDA016 Programming, first course (Java)

Grades D-program 2015



Goal of 2016

To **increase** the **challenge level** & **learning outcome** significantly.

Why Scala?

Easy for beginners and interesting for non-beginners:

- Regular semantics; e.g. value types are real objects
- Concise, expressive syntax
- Interactive learning in the Scala REPL
- Multi-paradigm, pragmatic:
  imperative, object-oriented, functional
- Rich semantics: can demonstrate many cs concepts
- Modern, evolving language: exciting for new students
- Free, open source language and tools

What is Scala?

- Aims to be a scalable, pragmatic, real-world language
- Multi-paradigm:
  - object-oriented
  - functional
  - imperative
  - concurrent
- Typing: static, strong, inferred, structural
- Designed by: Martin Odersky
- Developer: EPFL, Lightbend, OSS Community
- First appeared: January 20, 2004
- Platform: JVM, JavaScript
- License: BSD 3-clause
- Official site: `https://www.scala-lang.org/`

Overview of new LTH Course EDAA45

Home: **http://cs.lth.se/pgk**

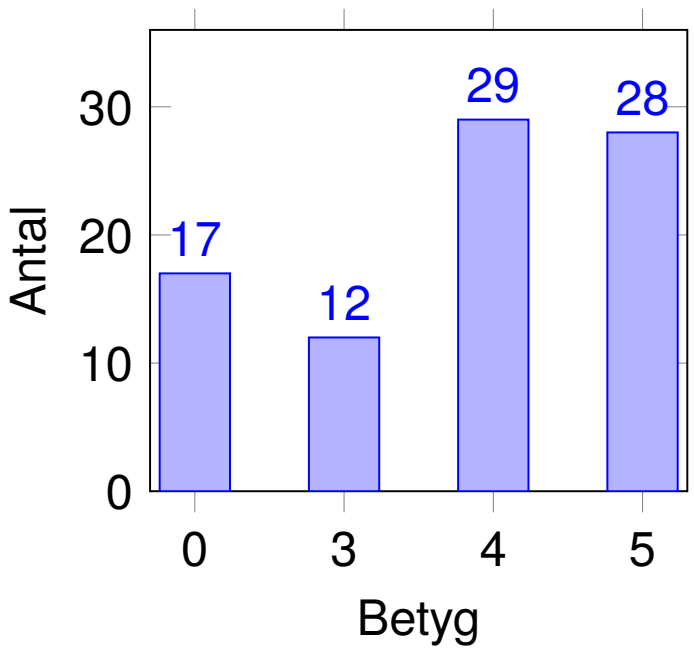
W	*Modul*	*Övn*	*Lab*
W01	Introduktion	expressions	kojo
W02	Kodstrukturer	programs	–
W03	Funktioner	functions	irritext
W04	Objekt	objects	blockmole
W05	Klasser	classes	turtlegraphics
W06	Sekvensalgoritmer	sequences	shuffle
W07	Datastrukturer	data	pirates
KS	KONTROLLSKRIVN.	–	–
W08	Matriser, typparametrar	matrices	maze
W09	Arv	inheritance	turtlerace-team
W10	Mönster, undantag, likhet	patterns	chords-team
W11	Scala och Java	scalajava	lthopoly-team
W12	Sökning, sortering, ordning	sorting	survey
W13	Repetition, tentaträning, projekt	–	Projekt
W14	Extra: jämlöpande exekvering	threads	–
T	TENTAMEN	–	–

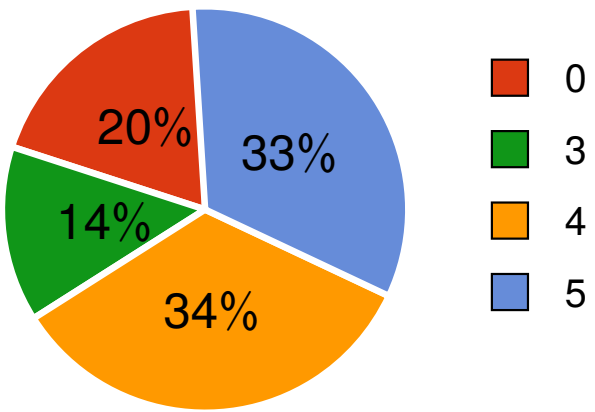
Students participation in teaching development through open sourcing of teaching material for Scala

https://github.com/lunduniversity/introprog

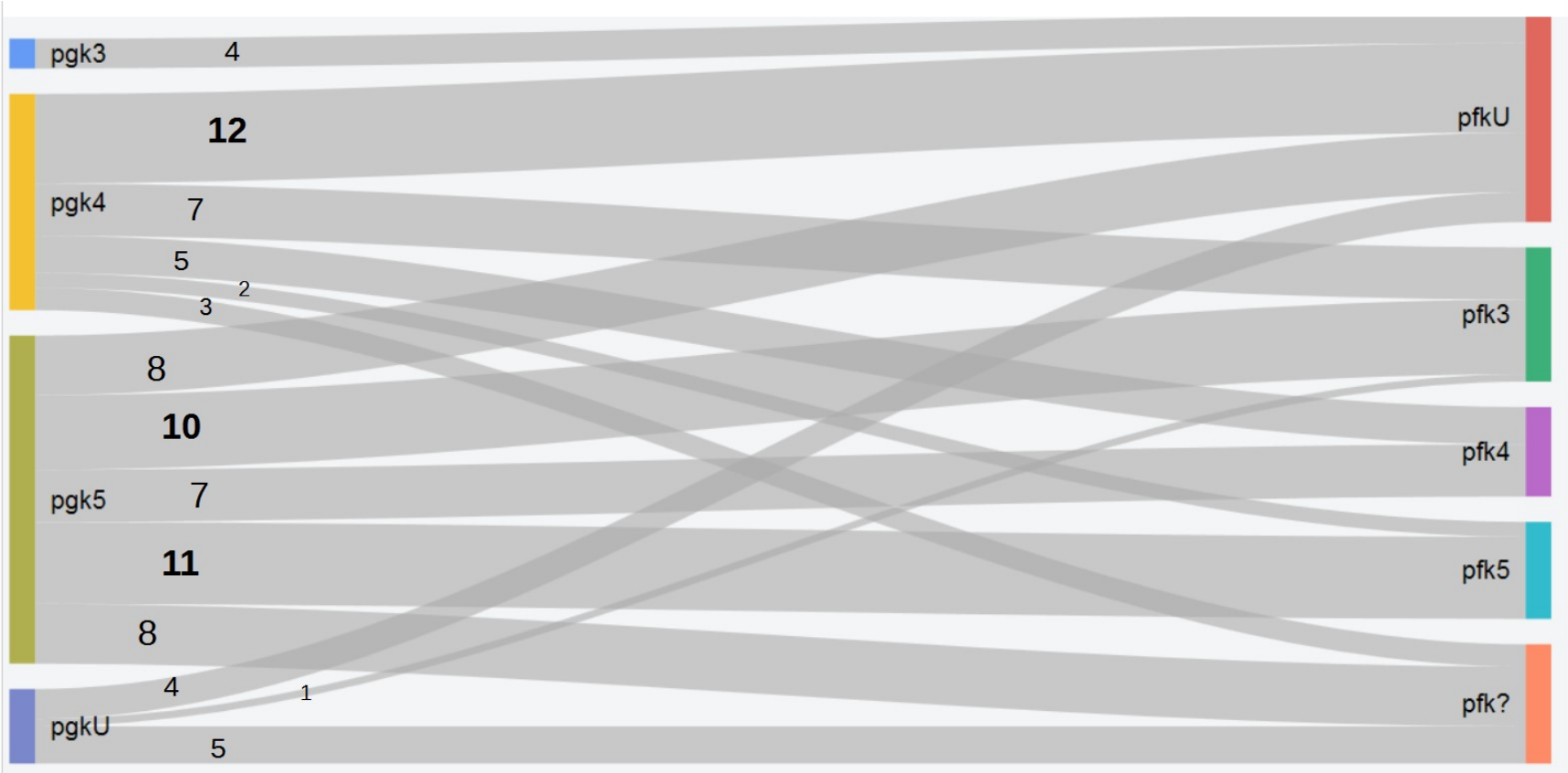
Anders Buhl,
Anna Palmqvist Sjövall,
Anton Andersson,
Casper Schreiter,
Cecilia Lindskog,
Christine Boghammar,
Emil Wihlander,
Emma Asklund,
Erik Bjäreholt,
Erik Grampp,
Erik Söderbarg,
Filip Stjernström,
Fredrik Danebjer,
Gustav Cedersjö,
Henrik Olsson,
Jacob Bohlin,
Jakob Hök,
Johan Ravnborg,
Jonas Danebjer,
Maj Stenmark,
Maximilian Rundgren,
Måns Magnusson,
Oscar Sigurdsson,
Oskar Berg,
Oskar Widmark,
Povel Larsson,
Sebastian Hegardt,
Stefan Jonsson,
Tom Postema,
Valthor Halldorsson,
Viktor Claesson

Betygsfördelning, D-are, totalt 86 st





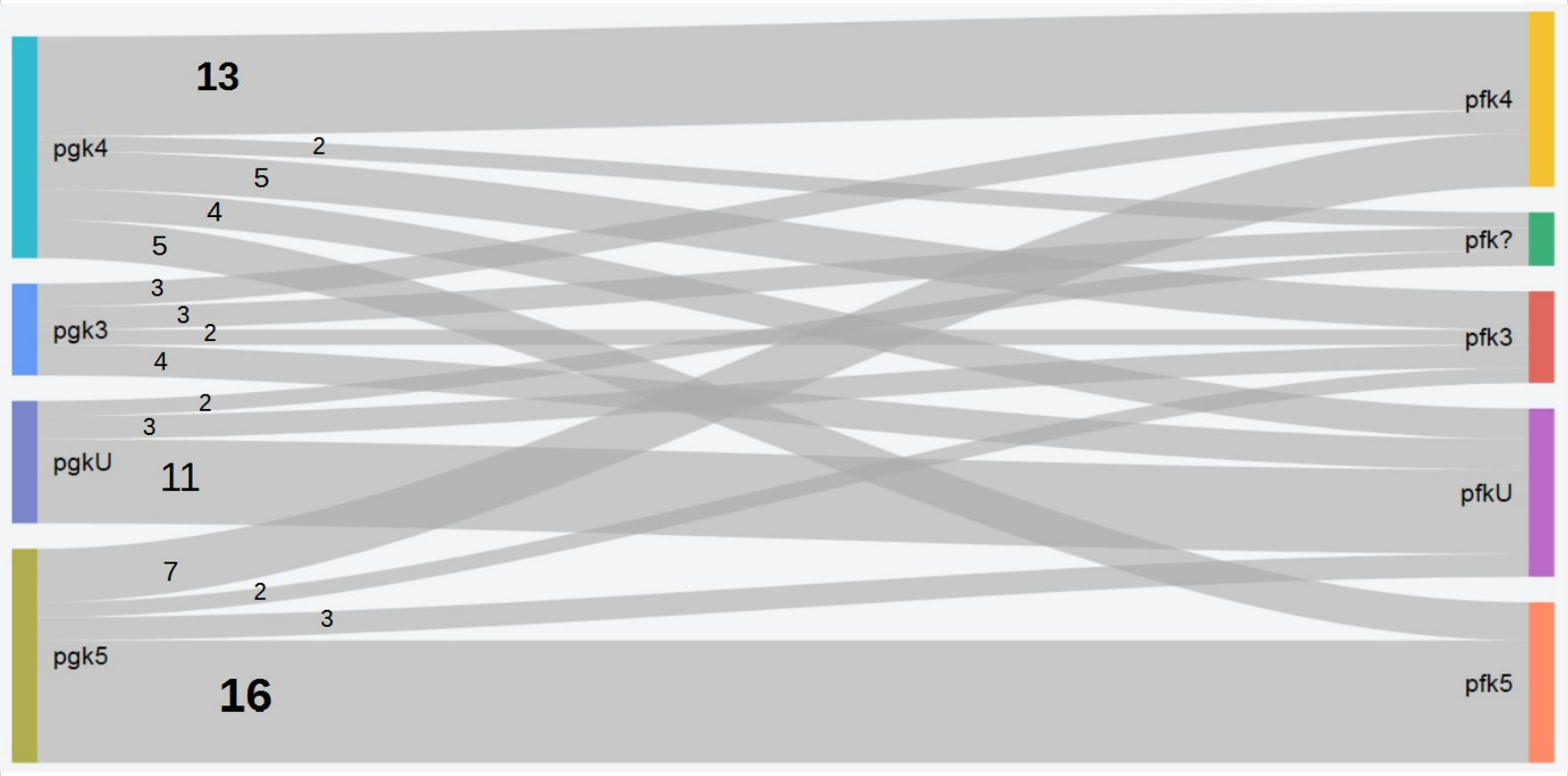
Grade progress 2015: EDA016 Java => EDAA01 Java



https://jsfiddle.net/dyb0kpvh/

pfk? resultat saknas på ordinarie tentamen

Grade progress 2016: EDAA45 Scala => EDAA01 Java



https://jsfiddle.net/wa7fr8po/

A brief intro to Scala

www.scala-lang.org

Scala – the simple parts

Lecture by **Martin Odersky**: www.youtube.com/watch?v=ecekSCX3B4Q

Scala for every-day dev actions:

1. **Compose:** everything is a composable **expression**
2. **Match:** decompose data with **pattern**-matching
3. **Group:** everything can be grouped and **nested**
4. **Recurse:** compose at any depth, loop with tail recursion
5. **Abstract:** functions are objects
6. **Aggregate:** collections aggregate & **transform** data
7. **Mutate:** local, private mutability to optimize perf.

SF Scala: Martin Odersky, Scala -- the Simple Parts

Some similarities between Scala and Java

- Both are object-oriented and imperative
- Both are statically typed (~ 100 times faster than Python)
- Both have C-like block syntax { }
- Both have lambdas (Java 8)
- Both run on the JVM
- Both can execute each other's byte code

Some differences between Scala and Java

- Scala is a more ”pure” OO language:
  instances of `Int, Double, Char`, etc. are **real objects**
- Scala is a more advanced functional language:
  easy to transform immutable data in functional collections
- Scala unifies OO and functional programming:
  functions are objects with an `apply`-method
- singelton `object` instead of Java’s `static`
- Some syntax differences:

Some differences between Scala and Java

- Scala is a more ”pure” OO language:
  instances of `Int, Double, Char`, etc. are **real objects**
- Scala is a more advanced functional language:
  easy to transform immutable data in functional collections
- Scala unifies OO and functional programming:
  functions are objects with an `apply`-method
- singelton `object` instead of Java's `static`
- Some syntax differences:
  - semicolons are inferred; newline btw statements is enough
  - no need for `return` as blocks are values
  - Type *after* names and colon: `val name: String = "Kim"`
  - generic types in `[T]` instead of <T>
  - Five types of members: `def, val, lazy val, var, type`
    Methods: `def isChild: Boolean = age < 18`
    Immutable fields: `val gender = "Female"`
    Delayed init: `lazy val r = List.fill(1000)(math.random)`
    Mutable fields: `var age: Int = 42`
    Type alias: `type Matrix = Map[Int, Map[Int, String]]`

Classes in Java and Scala

```java
// this is Java

public class JPerson {
    private String name;
    private int age;

    public JPerson(String n, int a) {
      name = n;
      age = a;
    }

    public JPerson(String n) {
      this(n, 42);
    }

    public String getName() {
      return name;
    }

    public int getAge() {
      return age;
    }

    public void setAge(int a) {
      age = a;
    }
}
```

Classes in Java and Scala

```java
// this is Java

public class JPerson {
    private String name;
    private int age;

    public JPerson(String n, int a) {
      name = n;
      age = a;
    }

    public JPerson(String n) {
      this(n, 42);
    }

    public String getName() {
      return name;
    }

    public int getAge() {
      return age;
    }

    public void setAge(int a) {
      age = a;
    }
}
```

```scala
// same in (non-idiomatic) Scala

class SPerson(n: String, a: Int) {
  private var name: String = n
  private var age: Int = a

  def this (n: String): Unit = {
    this(n, 42)
  }

  def getName = name

  def getAge = age

  def setAge(a: Int): Unit = {
    age = a
  }
}
```

Classes in Java and Scala

```java
// this is Java

public class JPerson {
    private String name;
    private int age;

    public JPerson(String n, int a) {
      name = n;
      age = a;
    }

    public JPerson(String n) {
      this(n, 42);
    }

    public String getName() {
      return name;
    }

    public int getAge() {
      return age;
    }

    public void setAge(int a) {
      age = a;
    }
}
```

```scala
// same in (non-idiomatic) Scala

class SPerson(n: String, a: Int) {
  private var name: String = n
  private var age: Int = a

  def this (n: String): Unit = {
    this(n, 42)
  }

  def getName = name

  def getAge = age

  def setAge(a: Int): Unit = {
    age = a
  }
}
```

```scala
// this is idiomatic Scala

case class Person(
  name: String,
  age: Int = 42
)
```