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

Download slides:
https://github.com/lunduniversity/intropro g/blob/master/about/course-experience-first-year

May 22, 2017

Agenda

1. Why a new course?
2. Why Scala?
3. Overview of new course
4. Experiences from first year
5. A very brief intro to Scala

Why a new course?

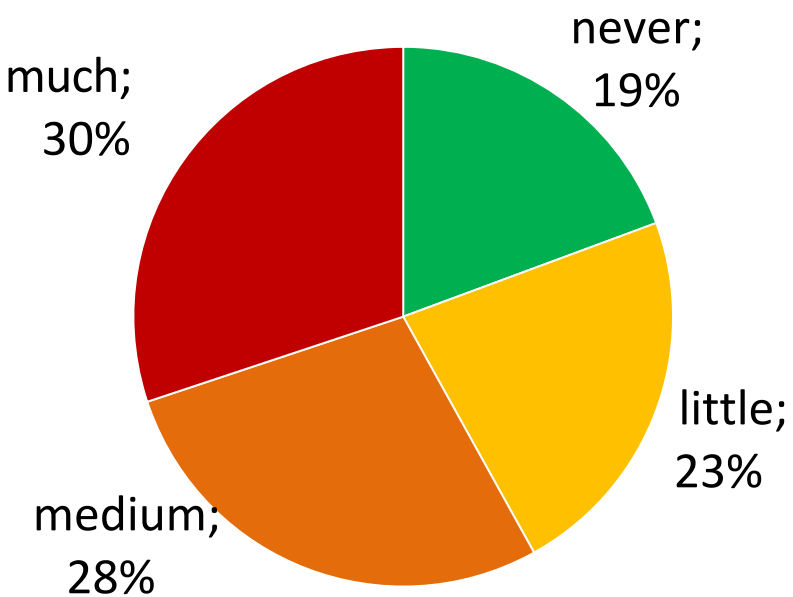
Why a new programming course for CSE students?

- Special teaching challenges when teaching introductory programming to CSE[1] program students at LTH:
  - Very large spread in pre-knowledge
  - Many students have very high ambitions
  - Some students have great difficulties
- The previous first course EDA016 needed an update after having the same architecture for many years

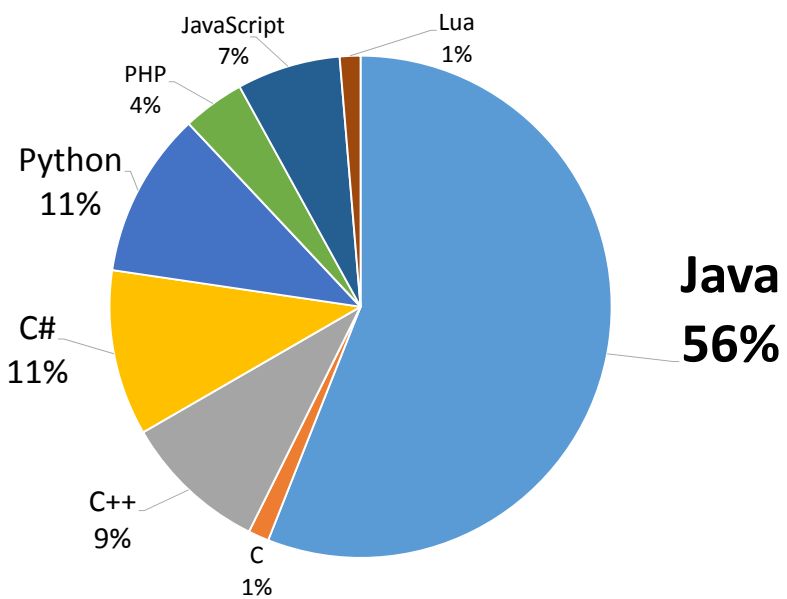
[1] CSE = Computer Sci. & Eng., In Swedish: Datateknikprogrammet (D)

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

Have programmed before?

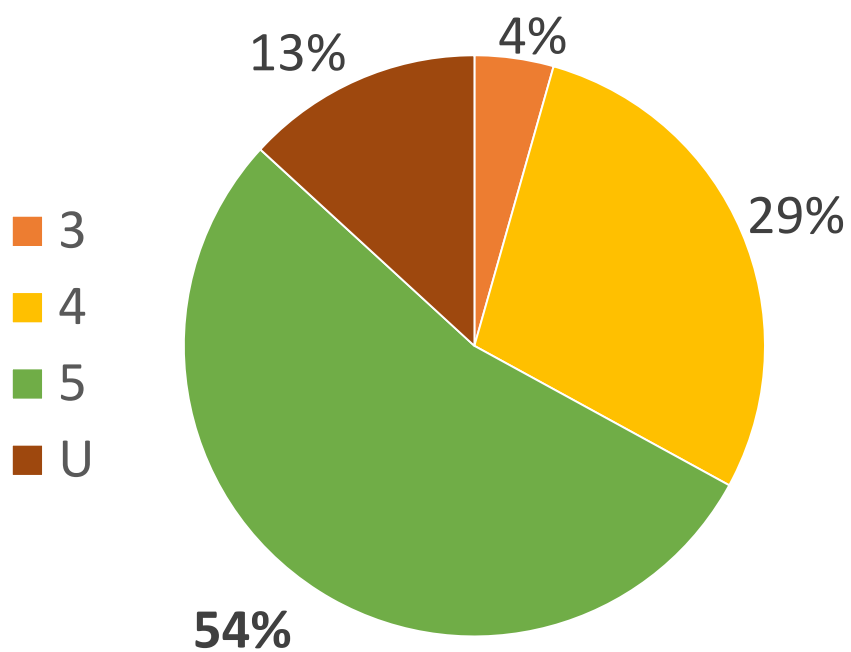


What language?



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

Grades 2015, old Java course, 91 students attending first exam



Main goals with new course

- Increase the **challenge**
- Increase the **learning outcome**
- Make an **interesting & engaging** start for CSE students
- Try new concepts in introductory programming teaching:
  - Involve old students in an open source project
  - Increase learning collaboration among students
  - Pioneering Scala as a first language
- Enable students to continue with an unchanged second course

First language at LTH

History of first languages at LTH for CSE students:

First language at LTH

History of first languages at LTH for CSE students:

Algol

First language at LTH

History of first languages at LTH for CSE students:

Algol 1982

First language at LTH

History of first languages at LTH for CSE students:

Algol	1982
Pascal	

First language at LTH

History of first languages at LTH for CSE students:

Algol	1982
Pascal	1985

First language at LTH

History of first languages at LTH for CSE students:

Algol	1982
Pascal	1985
Simula	

First language at LTH

History of first languages at LTH for CSE students:

Algol	1982
Pascal	1985
Simula	1990

First language at LTH

History of first languages at LTH for CSE students:

Algol	1982
Pascal	1985
Simula	1990
Java	

First language at LTH

History of first languages at LTH for CSE students:

Algol	1982
Pascal	1985
Simula	1990
Java	1997

First language at LTH

History of first languages at LTH for CSE students:

Algol	1982
Pascal	1985
Simula	1990
Java	1997
Scala	2016

Why Scala?

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 course

Summary of major changes

- New contents (selected):
  - Start imperative and functional, then gradually introduce OO
  - Start with objects as modules before instances
  - Immutable versus immutable datastructures
  - Use (but not implement) several structures: Vector, Set, Map, List
  - Use higher-order collection methods: map, foreach, filter, ...
  - Pattern matching to decompose data in case classes
  - Concept focus: compare paradigms, compare languages
  - ...
- Scala as a **first** language, with Java as a **second** language in the same course to make concept learning deeper
- Learning to code in collaboration teams
- New open source course material:
  - New lectures
  - New exercises
  - New labs
  - New type of written exam

Module structure (adapted based on experiences)

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

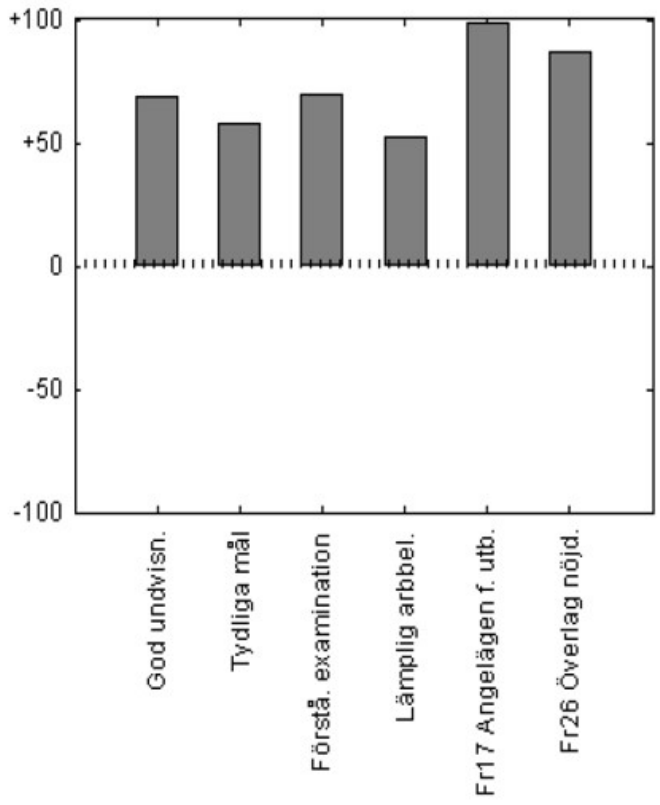
Experiences from first year

Summary of experiences

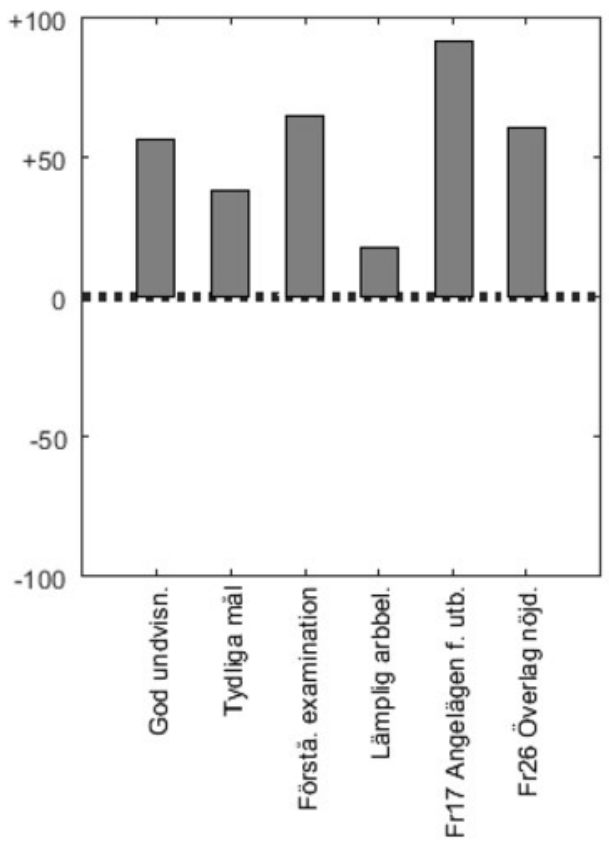
- Goals met:
  - Students with different pre-knowledge were all challenged
  - Higher learning outcome
  - Increased student engagement
- Scala is an excellent first language for CSE students and a pleasant to teach

Course Experience Questionnaire

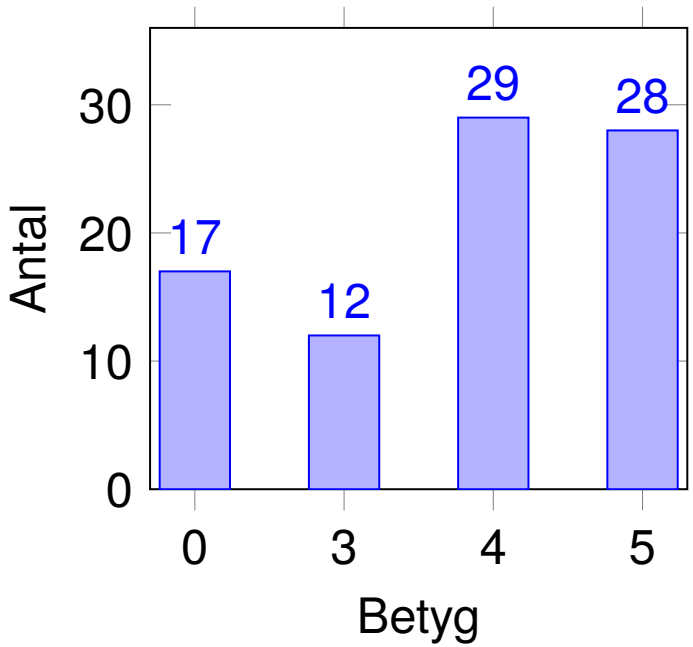
CEQ 2015 Old Java course

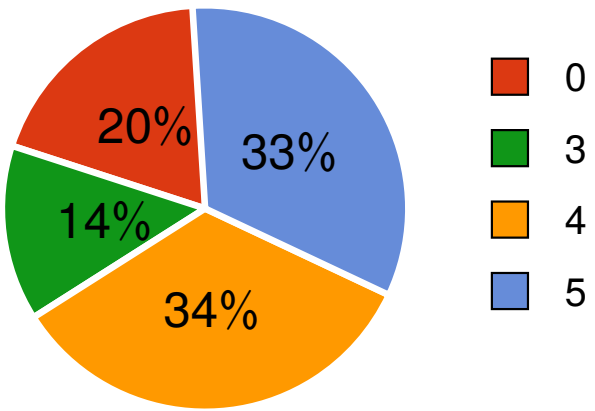


CEQ 2015 New Scala course

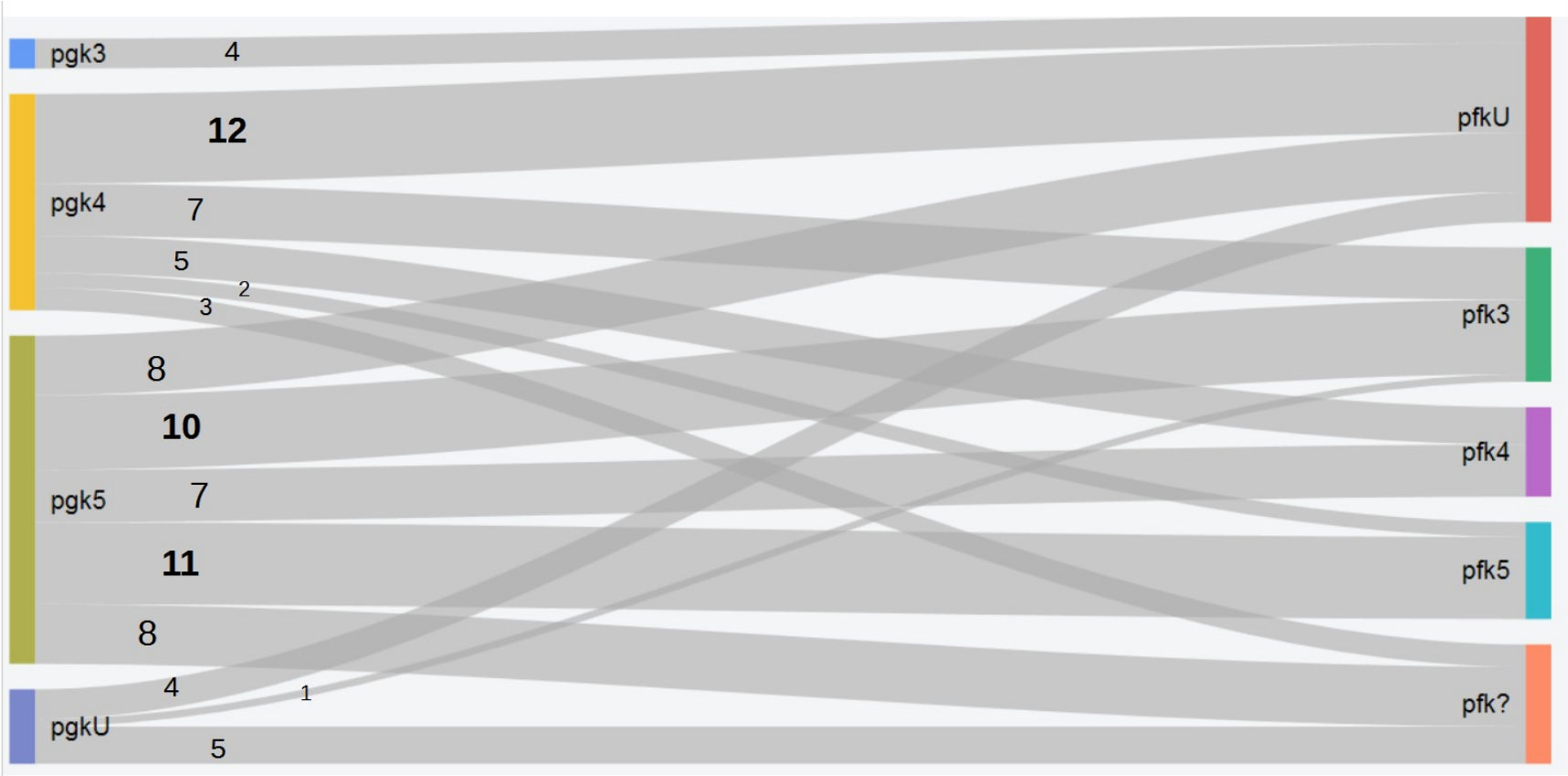


Betygsfördelning, D-are, totalt 86 st





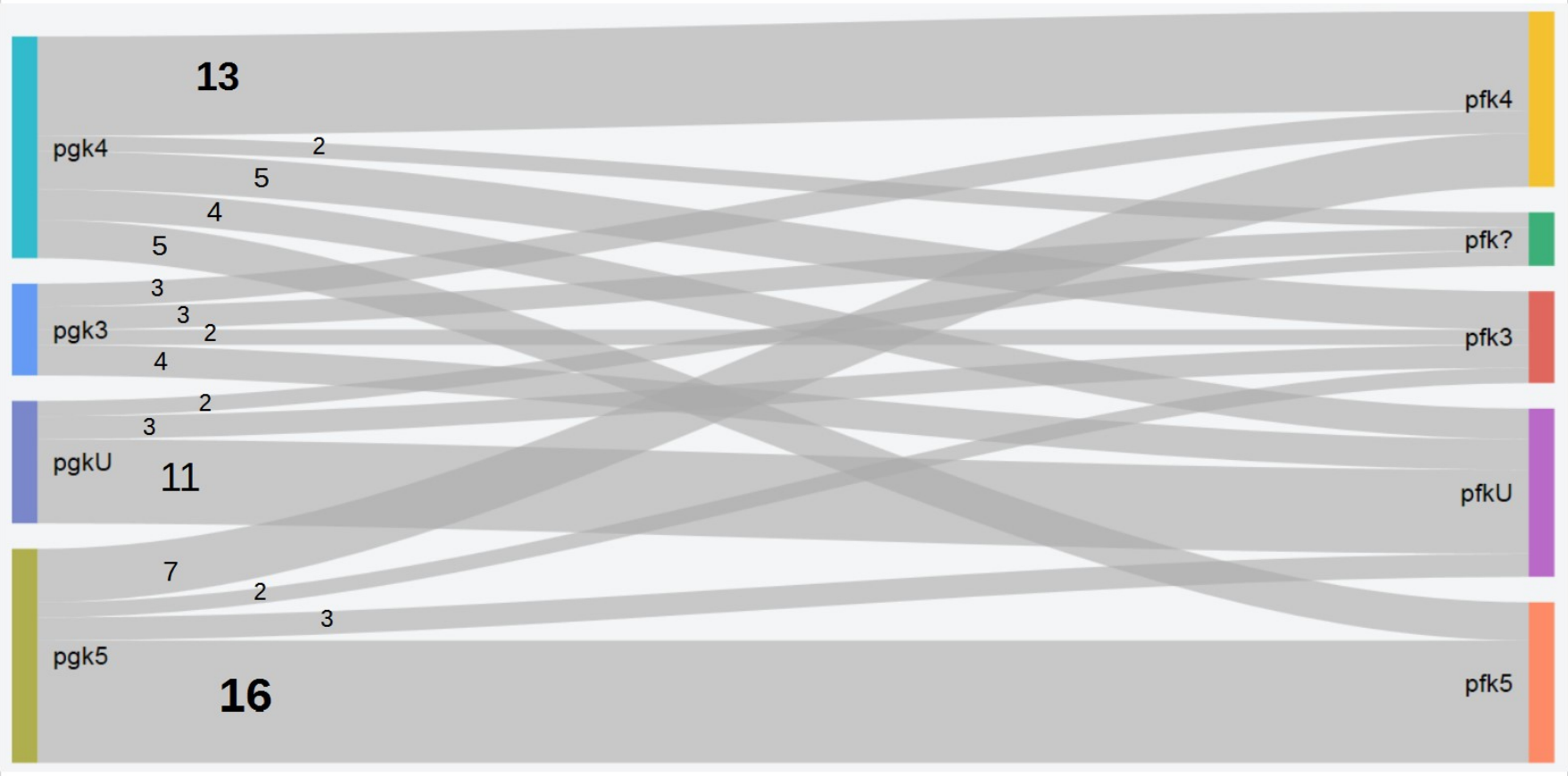
Grade progress 2015: EDA016 Java => EDAA01 Java



https://jsfiddle.net/dyb0kpvh/

pfk? resultat saknas på ordinarie tentamen

Grade progress 2016: EDAA45 Scala => EDAA01 Java



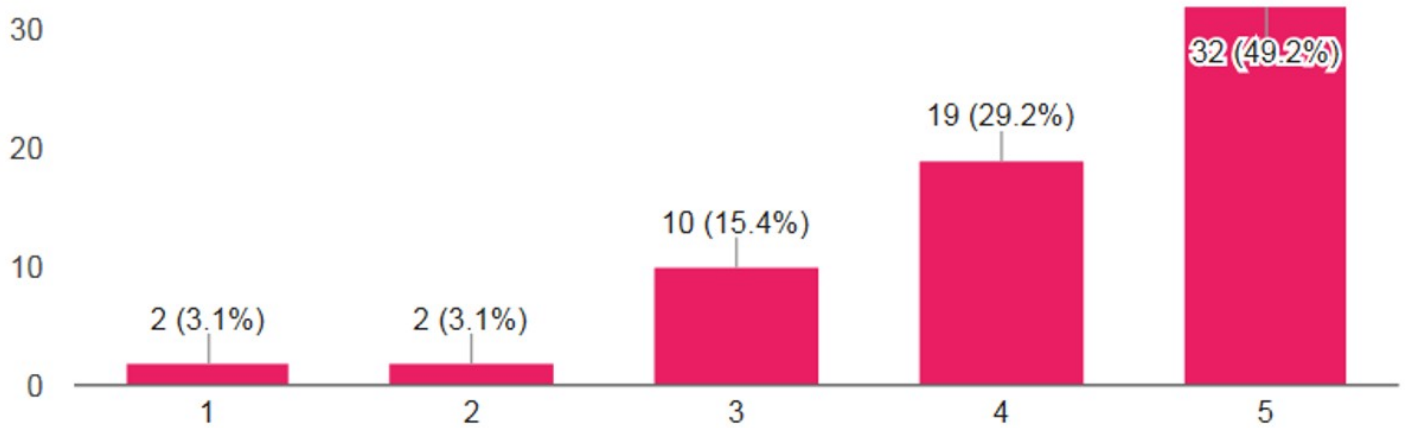
https://jsfiddle.net/wa7fr8po/

”I learn fast if I make an effort”

Mina programmeringskunskaper växer snabbt om jag anstränger mig. *

	1	2	3	4	5	
Håller inte alls med: det gick långamt för mig.	○	○	○	○	○	Håller helt med: det gick snabbt för mig.

(65 responses)

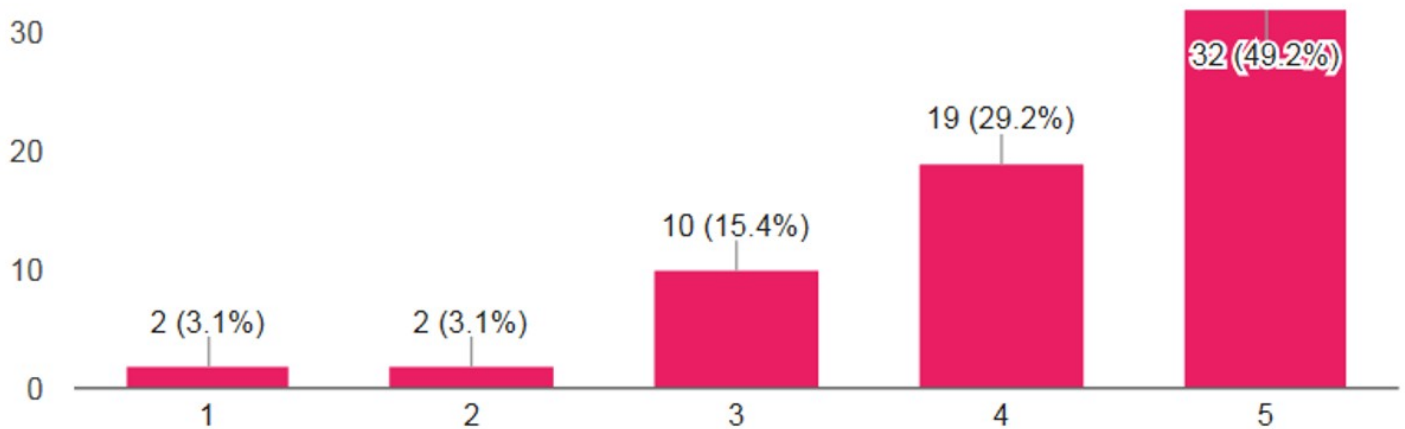


”I learn fast if I make an effort”

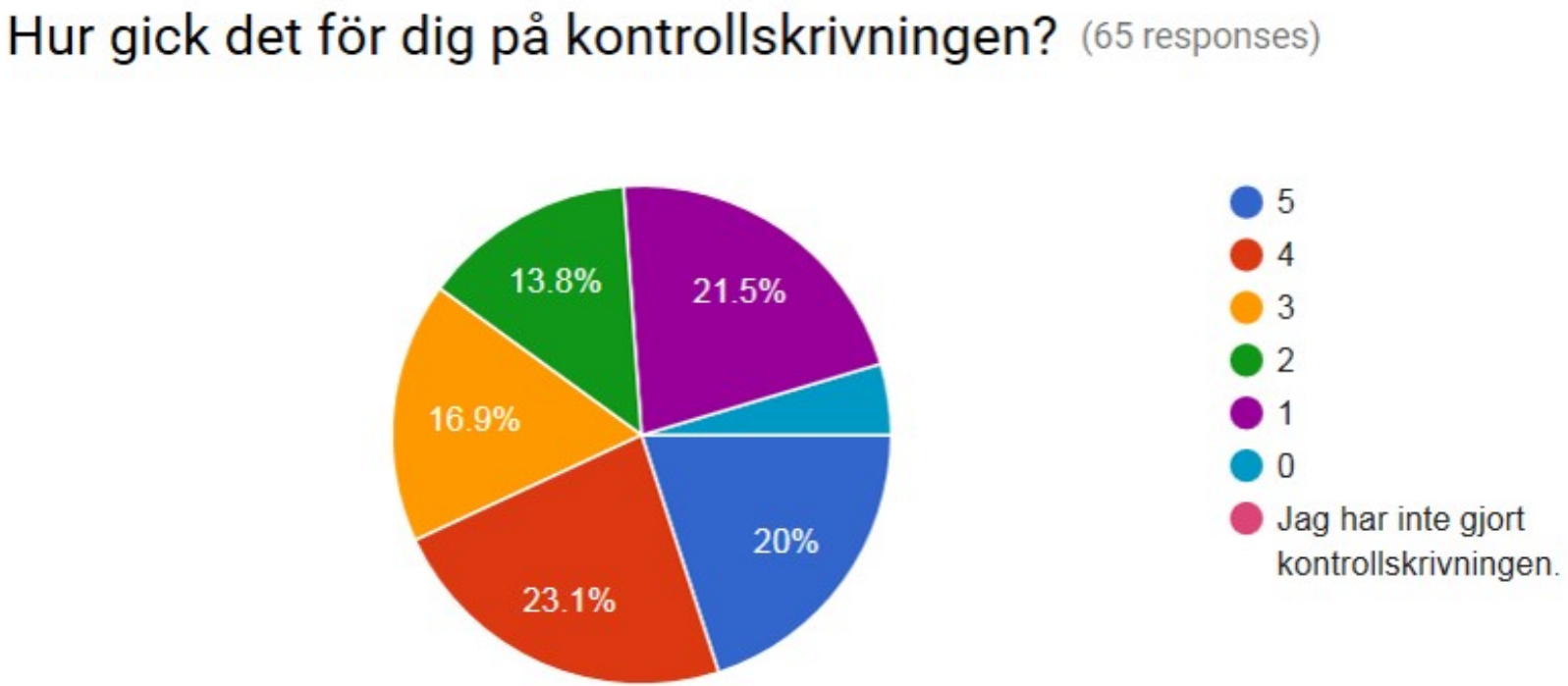
Mina programmeringskunskaper växer snabbt om jag anstränger mig. *

	1	2	3	4	5	
Håller inte alls med: det gick långamt för mig.	○	○	○	○	○	Håller helt med: det gick snabbt för mig.

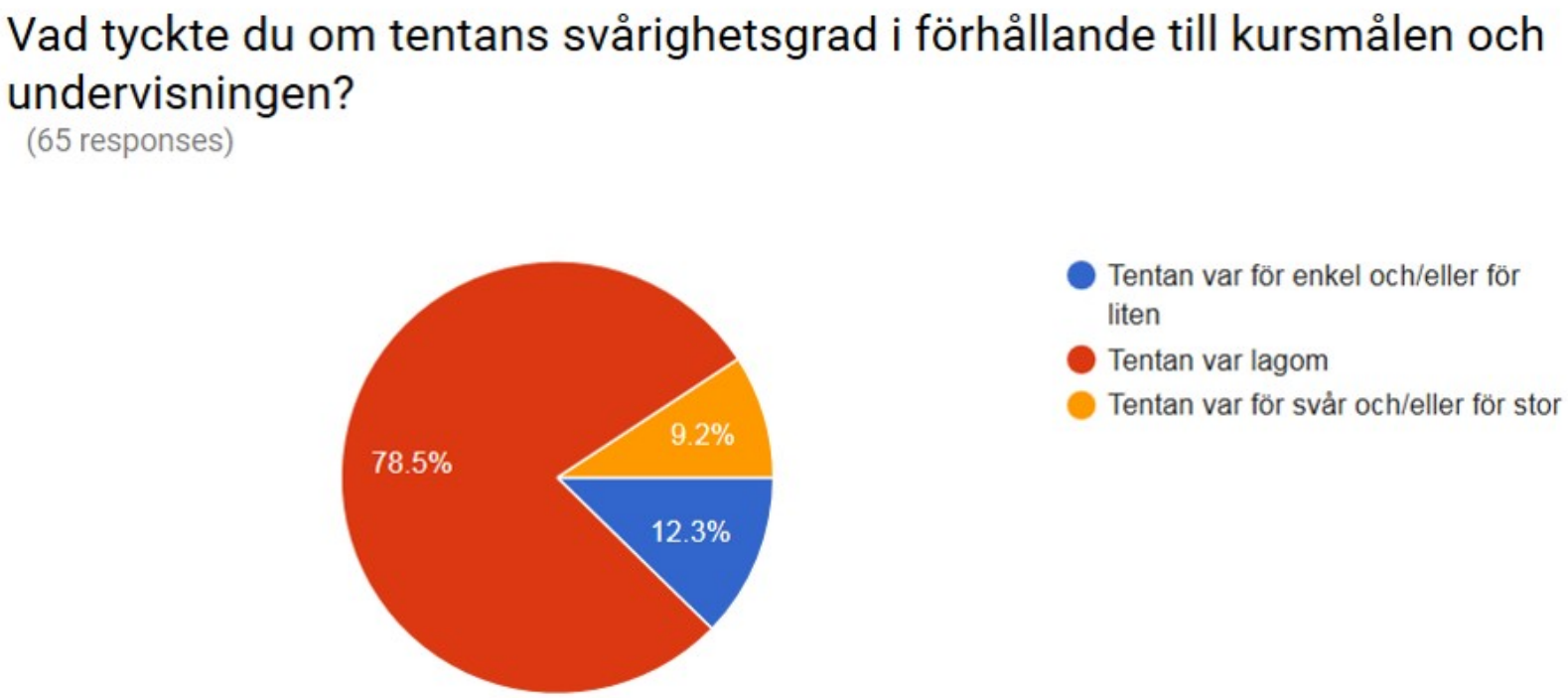
(65 responses)



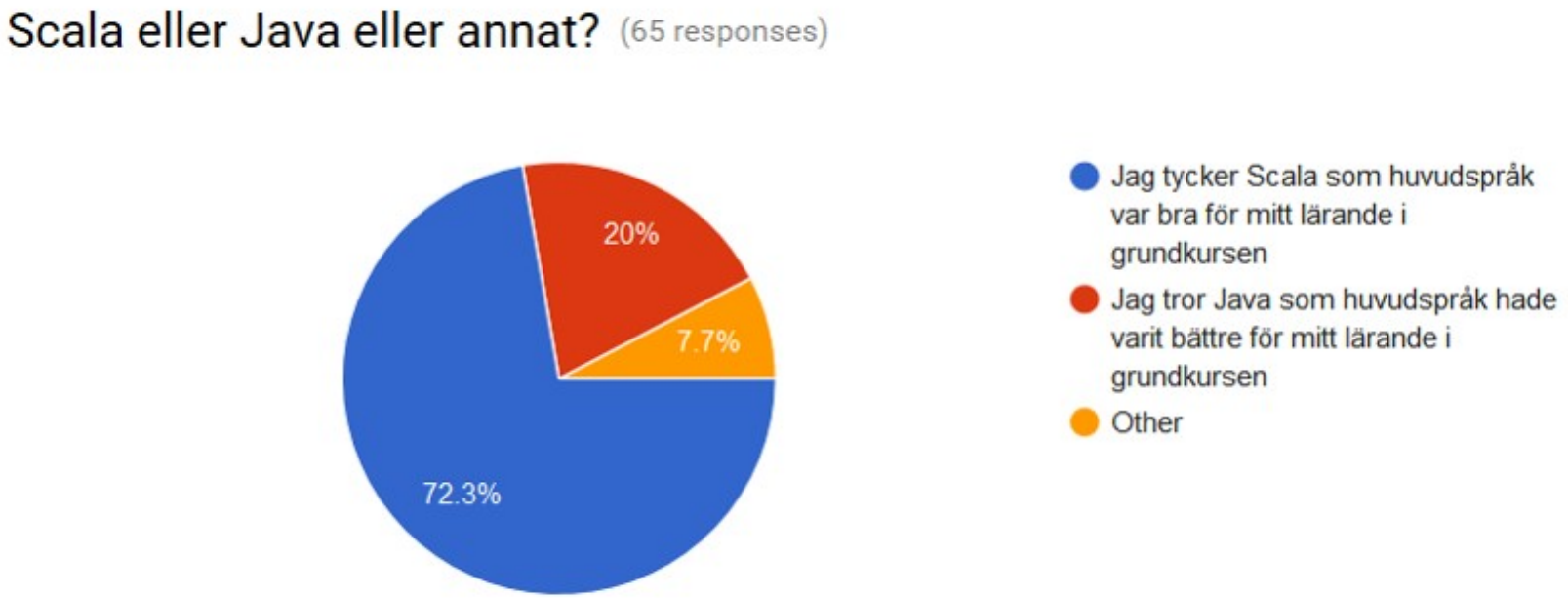
”How did you make it at the diagnostic mid-term test?”



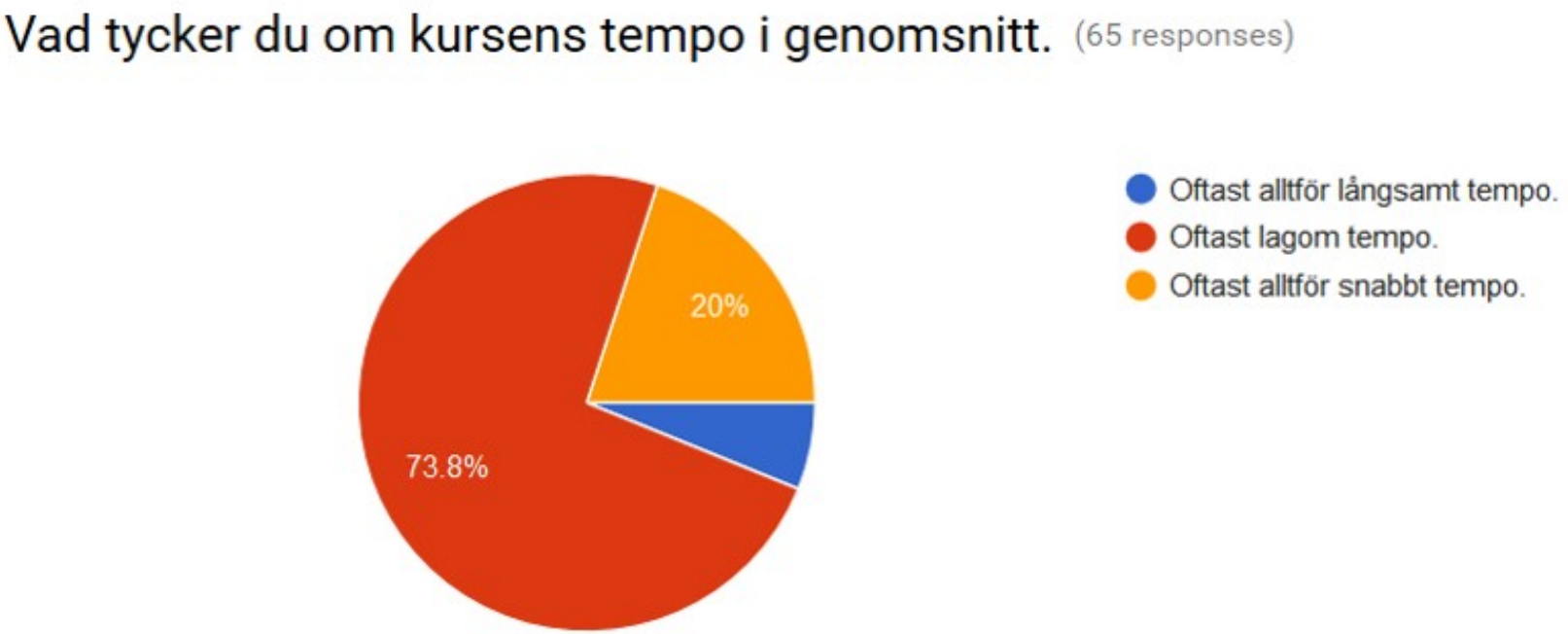
”What did you think about the level of the exam?”



”Scala or Java?”



”What do you think about the course tempo?”



”How do you value your learning outcome?”

A very brief intro to Scala

Official home of Scala

www.scala-lang.org
with tutorials, docs, etc.

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;
    }
}
```

```
// 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
  }
}
```

```
// this is idiomatic Scala

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

Live coding: code-along

Start the REPL:

```
$ scala
Welcome to Scala 2.11.8 (Java HotSpot(TM) VM, Java 1.8.
Type in expressions for evaluation. Or try :help.

scala> case class Person(name: String, age: Int = 42)
defined class Person

scala> Person("Björn", 48)
res0: Person = Person(Björn,48)

scala> Person("Kim")
res1: Person = Person(Kim,42)
```

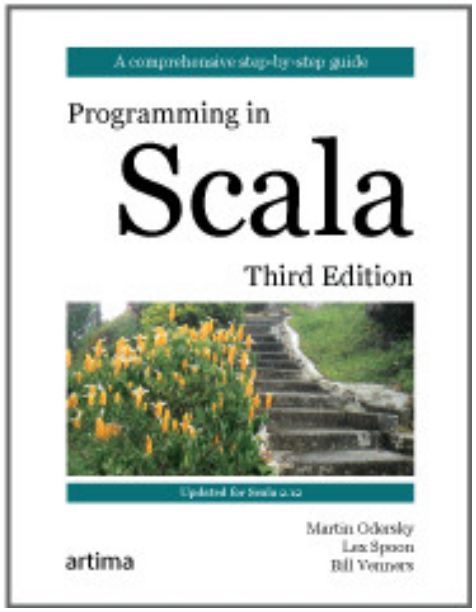
Functions are first-class values; Try this in REPL:

```
def öka(i: Int) = i + 1

val nums = Vector(1, 2, 3, 4, 42)

nums.map(öka)

nums.map(i => i + 1)

nums.map(_ + 1)

def mappa(xs: Vector[Int], f: Int => Int) = xs.map(f)

mappa(nums, öka)

def upprepa(n: Int)(block: => Unit) = for (i <- 1 to n) block
```

Recommended book for further reading

”Programmin in Scala”, 3rd Ed., by Odersky et al.

https://www.artima.com/shop/programming_in_scala_3ed



First edition available online:

http://www.artima.com/pins1ed/

Questions?

bjorn.regnell@cs.lth.se