

# Introduction to Functional Programming

## Course Summary and Future

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
             // guaranteed to be random.  
}
```

# The End of the Course

- Next week: Exam
  - Example exams + answers on the web
  - No computers
  - In English: Bring an English dictionary
    - answers may be in swedish
  - A list of standard Haskell functions

# What If ...

- You are not done with the labs in time?
  - Next year: This course runs again
    - Reuse labs
    - Possibly other/changed labs

# What If ...

- You do not pass the exam?
  - January: Re-exam
  - August: Re-exam
  - Next year: This course runs again

# What Have We Learned?

- Programming
  - For some of you: first time
  - Make the computer do some useful tasks
- Programming *Language*
  - Haskell
  - Different from what most of you had seen before
- Programming *Principles*
  - ...

# Programming Principles (I)

- Modelling
  - Create a new **type** that models what you are dealing with
  - Design and define **typed functions** around your types
  - Sometimes your type has an extra **invariant**
  - Invariants should be **documented** (for example as a property)

# Programming Principles (II)

- Properties
  - When you define **functions** around your types...
  - Think about and define **properties** of these functions
  - Properties can be **tested** automatically to find mistakes
  - Mistakes can be in your functions (program) or in your properties (understanding)

# Programming Principles (III)

- Recursion
  - When you need to solve a **large, complicated problem...**
  - Break the problem up into a smaller piece, or a number of **smaller pieces**
  - These can be solved **recursively**
  - Solve the whole problem by **combining** all recursive solutions



# Programming Principles (IV)

- Abstraction and Generalization
  - When you find yourself **repeating** a programming task
  - Take a step back and see if you can **generalize**
  - You can often define an **abstraction** (higher-order function) performing the old task *and* the new one
  - Avoid **copy-and-paste programming**

# Programming Principles (V)

- Pure functions
  - Use **pure functions** as much as possible
  - These are easier to **understand, specify** and **test**
  - Concentrate **IO instructions** in a small part of your program
  - Concentrate **GUI instructions** in a small part of your program

# Programming Principles (VI)

- Separation
  - Divide up your program into **small units** (functions)
  - These should be grouped together into **larger units** (modules)
  - **Minimize** dependencies between these parts
  - So that it is easy to make **internal changes**, without affecting your whole program

# Programming Principles

- Important!
- Independent of *programming language*

# Report from the front

“Läste kursen 2010 när jag började på D och lärde mig mycket, fast jag tyckte att jag kunde programmera innan. Fick 2012 jobb på Ericsson och programmerade då i Python, och använde då dagligen tekniker som jag lärde mig i kursen, framförallt då rekursion, operationer på listor och delar av det funktionella programmerings sättet som var nytt för mig 2010.”

# Report from the front

“En vanlig fråga/missuppfattning som jag minns från början av Chalmers är just 'varför Haskell? Ingen använder det på riktigt i industrin', och det kan vara värt att påminna en extra gång om att man lär sig metoder och tankesätt som är användbara oavsett vilket språk man sedan kodar i.”

# Why Haskell?

- What is easy in Haskell:
  - Defining types
  - Properties and testing
  - Recursion
  - Abstraction, higher-order functions
  - Pure functions
  - Separation (laziness)

# Why Haskell (II)?

- What is harder in Haskell:
  - Ignoring types
    - Static strong typing
    - Expressive type system
      - Most advanced type system in a real-world language
  - Impure functions
    - All functions are pure
      - The only general existing programming language
    - Instructions are created and composed explicitly
      - Makes it clear where the "impure stuff" happens



# Functional Programming

- “Drives” development of new programming languages
  - Type systems
  - Garbage collection
  - Higher-order functions / Lambdas
  - List comprehensions
  - ...
- Haskell is the most advanced functional programming language today

# Functional Programming

- Hot topic in PL community and industry
  - Compilers/compiler-like
  - Domain-specific languages (Haskell)
    - build *your own* programming language with little effort
  - Telecom industry (Erlang)
    - Dealing with complex protocols/data-flow
    - Need to get *right*
  - Financial industry (Haskell)
    - Dealing with complex calculations
    - Need to get *right*

# Functional Programming

- Writing programs = defining (pure) functions and composing functions
- Running programs = evaluating expressions
- Functions are “first-class”, they can be created (lambda expressions) and passed around as arguments (higher order functions)

**Functional programming language =**  
a language in which this style is *easy* and *encouraged*

# Imperative Programming

- Writing programs = writing instructions and composing instructions that do things and change things
- Running programs = executing instructions

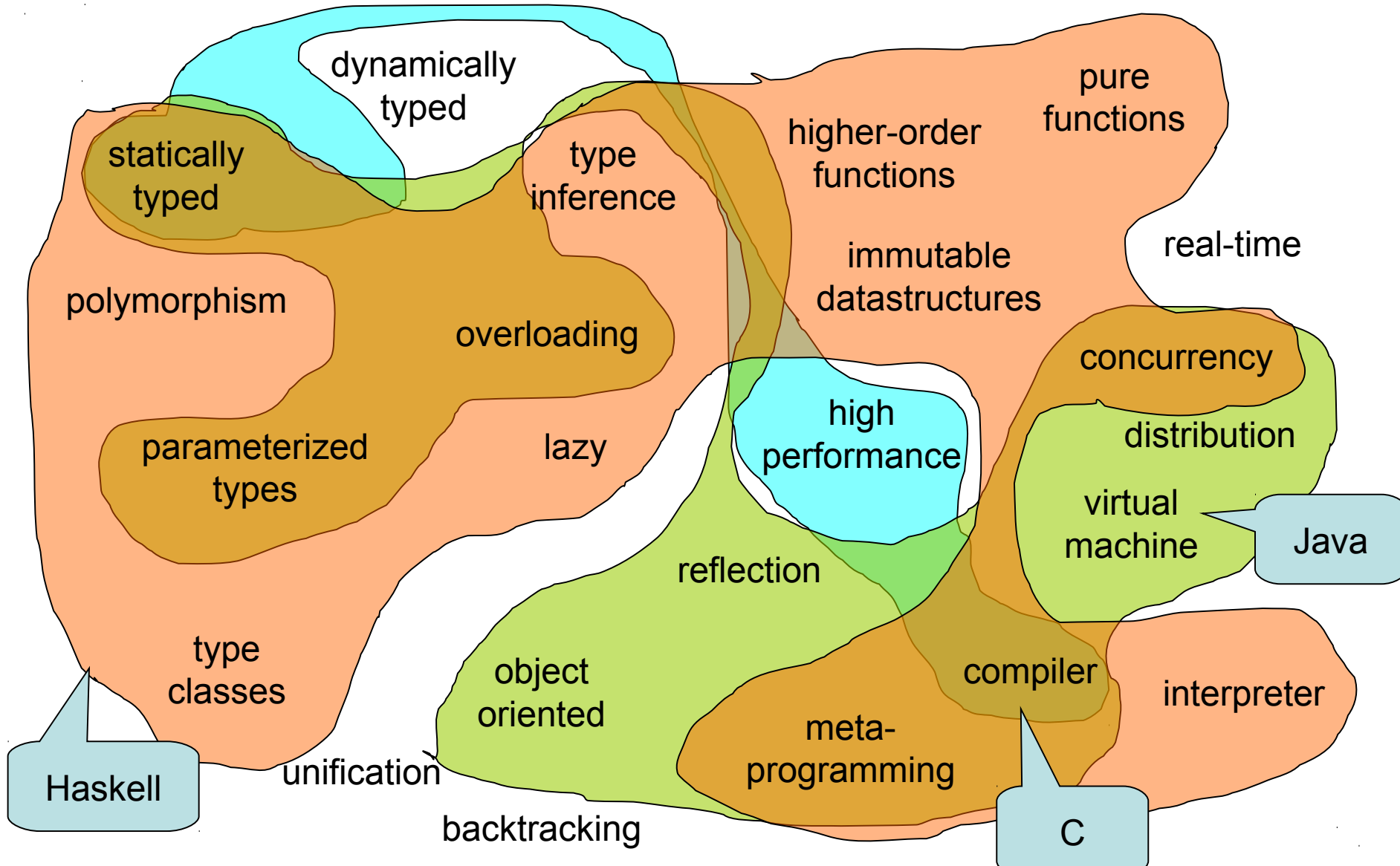
A Wise Man ..

A Good Functional  
Programmer is a  
Good Programmer

# Programming Languages

Lisp Scheme C BASIC  
Haskell Java C++  
ML Python C# JavaScript  
O'CaML Curry csh Perl  
Erlang bash Prolog Ruby  
Lustre Mercury PostScript  
VHDL Esterel SQL PDF  
Verilog

# Programming Language Features



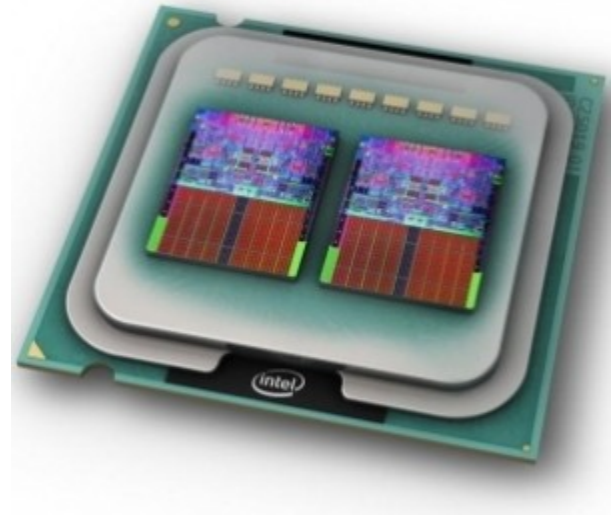
# Learning a Programming Language

- Learn the new features, principles, associated with the language
- Reuse things you know from other languages
- Learn *different* languages
  - what is popular now might not be popular in 5 years from now
- Use the right language for the right job
  - Systems consist of several languages



# Multi-core Revolution

- Traditional ways of programming *do not work* – a **challenge** for the programming language community
- Right now, industry is looking for alternatives
  - Intel
  - Microsoft
  - IBM
  - ...



# Alternatives?

- Expression-level parallelism

- Haskell

- Other functional languages

**restriction:**  
no side effects

- Software Transactional Memory

- Haskell

**restriction:**  
control of  
side effects

- Message passing between processes

- Erlang

**restriction:**  
no shared  
memory

# This Course

- Introduction to programming
- Introduction to Haskell
  
- There is lots, lots more...

# Coming Programming Courses

D-line

- Objektorienterad programming
  - Java
- Embedded systems programming
  - Assembly
  - C

---
- Data structures
  - Java
  - Haskell

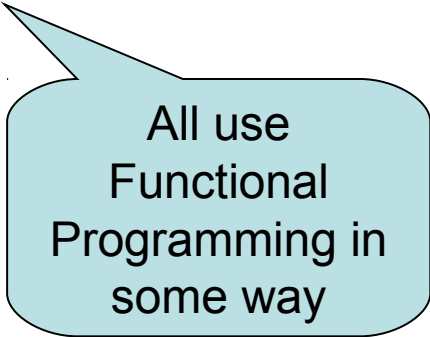
GU

- Two programming courses
  - Both in Java
- Datastructures
  - Java
  - Haskell

---

# Future Programming Courses

- Concurrent Programming
- Compiler Construction
- Advanced Functional Programming
- Parallel Functional Programming
- Software Engineering using Formal Methods
- Language Technology
- Programming Paradigms
- ...



All use  
Functional  
Programming in  
some way

# Course evaluation

- Please don't forget to fill in the course evaluation!
- This will help us improve the course in coming years