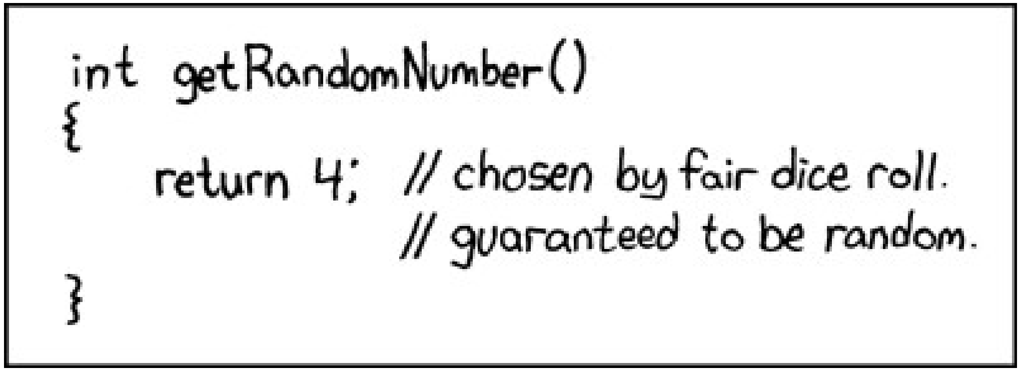
Introduction to Functional Programming

Course Summary and Future



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

Based on material by Koen Lindström Claessen

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
    - Possibly changed labs

- 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)

- Breaking up problems into simpler parts, recursion
  - When you need to solve a **large, complicated problem**...
  - Continue breaking up until the parts are simple, or until you can use an existing solution
  - The parts can be solved **recursively**
  - Solve the whole problem by **combining** the solutions of all parts

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

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 programmeringssättet 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
      - Unique among real-world languages
    - Instructions are created and composed explicitly
      - Makes it clear where the ”impure stuff” happens

Two major paradigms

Imperative programming:

- Instructions are used to change the computer's state:
  - x := x+1
  - deleteFile(”slides.pdf”)
- Run the program by following the instructions top-down

Functional programming:

- Functions are used to declare dependencies between data values:
  - y = f(x)
- Dependencies drive evaluation

Two major paradigms

Imperative programming:

- **Instructions** are used to change the computer's **state**:
  - x := x+1
  - deleteFile(”slides.pdf”)
- Run the program by following the instructions top-down

Functional programming:

- **Functions** are used to declare dependencies between **data values**:
  - y = f(x)
- Dependencies drive evaluation

Functional Programming

- **Functions** are used to declare dependencies between data values:
  - y = f(x)
- **Functions** are the basic building blocks of programs
- **Functions** are used to compose **functions** into larger **functions**
- In a (pure) **function**, the result depends *only* on the argument (no external communication)

Functional Programming

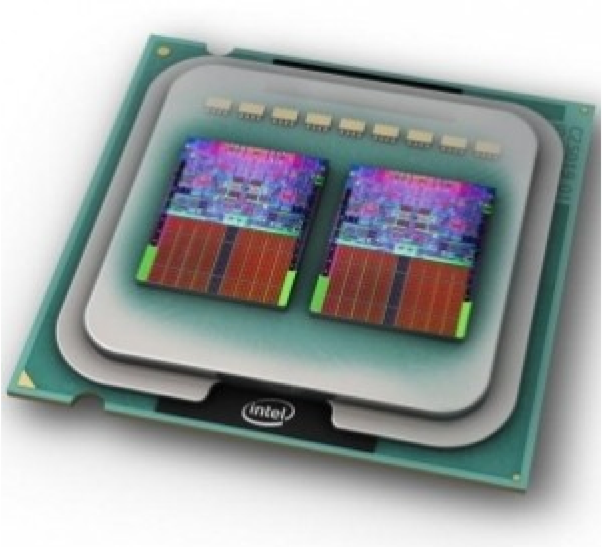
- “Drives” development of new programming languages
  - Type systems
  - Garbage collection
  - Higher-order functions / Lambdas
  - List comprehensions
  - ...
- Haskell is the most advanced functional programming language today

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
- Software Transactional Memory
  - Haskell
- Message passing between processes
  - Erlang

restriction:
no
side effects

restriction:
control of
side effects

restriction:
no shared
memory

This Course

- Introduction to programming
- Introduction to Haskell

- There is lots, lots more...

Coming Programming Courses

D-line

- Grundläggande datorteknik
  - Assembler
- Objektorienterad programming
  - Java
- Inbyggda system
  - 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