Programming in Haskell
Introduction

Prof. Dr. Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

Sommercampus 2017

Coordinates

- **Course hours:** Th, 9-18 Uhr, HS 101-00-026
- **Staff:** Alexander Thiemann, Prof. Dr. Peter Thiemann, Dr. Stefan Wehr

  `Gebäude 079, Raum 00-015`
  `Telefon: 0761 203 -8051/-8247`
  `E-mail: thiemann@cs.uni-freiburg.de`
  `Web: http://www.informatik.uni-freiburg.de/~thiemann`
- **Homepage:**
  https://github.com/proglang/HaskellKurs2017

Contents

- Basics of functional programming using Haskell
- Haskell development tools
- Writing Haskell programs
- Using Haskell libraries
- Your first Haskell project

What is Haskell?

In September of 1987 a meeting was held at the conference on Functional Programming Languages and Computer Architecture in Portland, Oregon, to discuss an unfortunate situation in the functional programming community: there had come into being more than a dozen non-strict, purely functional programming languages, all similar in expressive power and semantic underpinnings. There was a strong consensus at this meeting that more widespread use of this class of functional languages was being hampered by the lack of a common language. It was decided that a committee should be formed to design such a language, providing faster communication of new ideas, a stable foundation for real applications development, and a vehicle through which others would be encouraged to use functional languages.

From "History of Haskell"

What is Functional Programming?

Functions and values

rather than

Assignments and pointers

Functional Programming: Variables

Functional (Haskell)

```
x :: Int
x = 5nn
```

Variable `x` has value 5 forever

Functional Programming: Variables

Functional (Haskell)

```
x :: Int
x = 5nn
```

Variable `x` has value 5 forever

Imperative (Java)

```
int x = 5;
...
x = x+1;
```

Variable `x` can change its content over time

Functional Programming: Functions

Functional (Haskell)

```
f :: Int -> Int -> Int
f x y = 2*x + y

f 42 16 // always 100
```

Value of a function **only** depends on its inputs

Functional Programming: Functions

Functional (Haskell)

```
f :: Int -> Int -> Int
f x y = 2*x + y

f 42 16 // always 100
```

Value of a function **only** depends on its inputs

Imperative (Java)

```
boolean flag;
static int f (int x, int y) {
  return flag ? 2*x + y , 2*x - y;
}

f (42, 16); // who knows?
```

Return value depends on non-local variable

Functional Programming: Laziness

Haskell

```
x = expensiveComputation
g anotherExpensiveComputation
```

- The expensive computation will only happen if `x` is ever used.
- Another expensive computation will only happen if `g` uses its argument.

Functional Programming: Laziness

Haskell

```
x = expensiveComputation
g anotherExpensiveComputation
```

- The expensive computation will only happen if `x` is ever used.
- Another expensive computation will only happen if `g` uses its argument.

Java

```
int x = expensiveComputation;
g (anotherExpensiveComputation)
```

Both expensive computations will happen anyway.

Many more features that make programs more concise

- Algebraic datatypes
- Polymorphic types
- Parametric overloading
- Type inference
- Monads & friends (for IO, concurrency, ...)
- Comprehensions
- Metaprogramming
- Domain specific languages
- ...

References

- Paper by the original developers of Haskell in the conference on History of Programming Languages (HOPL III): http://dl.acm.org/citation.cfm?id=1238856
- The Haskell home page: `http://www.haskell.org`
- Haskell libraries repository: `https://hackage.haskell.org/`
- Haskell Tool Stack: `https://docs.haskellstack.org/en/stable/README/`

?

Let’s get started!