Moduldokumentation

Modul Functional Programming (fprog)

Simon Wächter

2016

Inhalt

[1 Einleitung 2](#_Toc462689669)

[1.1 Einleitung 2](#_Toc462689670)

[1.2 Lernziele 2](#_Toc462689671)

[1.3 Prüfungen 2](#_Toc462689672)

[2 Woche 1 3](#_Toc462689673)

[2.1 What is a Functional Language 3](#_Toc462689674)

[2.2 Example 3](#_Toc462689675)

[2.3 History 3](#_Toc462689676)

[2.4 Glasgow Haskell Compiler 3](#_Toc462689677)

[2.5 Prelude 3](#_Toc462689678)

[2.6 Examples 4](#_Toc462689679)

[2.7 Function Application 4](#_Toc462689680)

[2.8 Examples 5](#_Toc462689681)

[2.9 Haskell Scripts 5](#_Toc462689682)

[2.10 GHCI Commands 5](#_Toc462689683)

[2.11 Name Requirements 6](#_Toc462689684)

[2.12 Layout Rule 6](#_Toc462689685)

[3 Woche 2 7](#_Toc462689686)

[3.1 What is a Type 7](#_Toc462689687)

[3.2 Type Errors 7](#_Toc462689688)

[3.3 Types in Haskell 7](#_Toc462689689)

[3.4 Basic Types 8](#_Toc462689690)

[3.5 List Types 8](#_Toc462689691)

[3.6 Tuple Types 9](#_Toc462689692)

[3.7 Function Types 10](#_Toc462689693)

[3.8 Curried Functions 10](#_Toc462689694)

[3.9 Why is Currying useful 11](#_Toc462689695)

[3.10 Currying Conventions 12](#_Toc462689696)

[3.11 Polymorphic Functions 12](#_Toc462689697)

[3.12 Overloaded Functions 13](#_Toc462689698)

[3.13 Hints and Tips 14](#_Toc462689699)

[4 Woche 3 15](#_Toc462689700)

# Einleitung

## Einleitung

Dieses Dokument stellt die Moduldokumentation für das Modul fprog dar. Allfällige Unterlagen sind im Modulordner zu finden.

## Lernziele

Das Modul beinhaltet folgende Lernziele:

* Sinn der funktionalen Programmierung
* Anwenden der funktionalen Programmierung

## Prüfungen

Die Modulnote setzt sich zu 100% aus zwei Semesterprüfungen zu 50 % zusammen.

# Woche 1

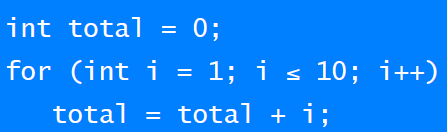
## What is a Functional Language

Opinions differ, and it is difficult to give a precise definition, but generally speaking:

* Functional programming is **style** of programming in which the basic method of computation is the application of functions to arguments;
* A functional language is one that **supports** and **encourages** the functional style.

## Example

Summing the integers 1 to 10 in Java (The computation method is **variable assignment**):



Summing the integer 1 to 10 in Haskell (The computation method is **function application**):



## History

* 1930s: Alonzo Church develops the lambda calculus, a simple but powerful theory of functions.
* 1950s: John McCarthy develops Lisp, the first functional language, with some influences from the lambda calculus, but retaining variable assignments.
* 1960s: Peter Landin develops ISWIM, the first pure functional language, based strongly on the lambda calculus, with no assignments.
* 1970s: John Backus develops FP, a functional language that emphasizes higher-order functions and reasoning about programs.
* 1970s: Robin Milner and others develop ML, the first modern functional language, which introduced type inference and polymorphic types.
* 1970s – 1980s: David Turner develops a number of lazy functional languages, culminating in the Miranda system.
* 1987: An international committee starts the development of Haskell, a standard lazy functional language.
* 1990s: Phil Wadler and others develop type classes and monads, two of the main innovations of Haskell.
* 2003: The committee publishes the Haskell Report, defining a stable version of the language; an updated version was published in 2010.
* 2010 – today: Haskell Platform - Standard distribution, library support, new language features, development tools, use in industry, influence on other languages, etc.

## Glasgow Haskell Compiler

* GHC is the leading implementation of Haskell, and comprises a compiler and interpreter;
* The interactive nature of the interpreter makes it well suited for teaching and prototyping;
* GHC is freely available from the Haskell website;

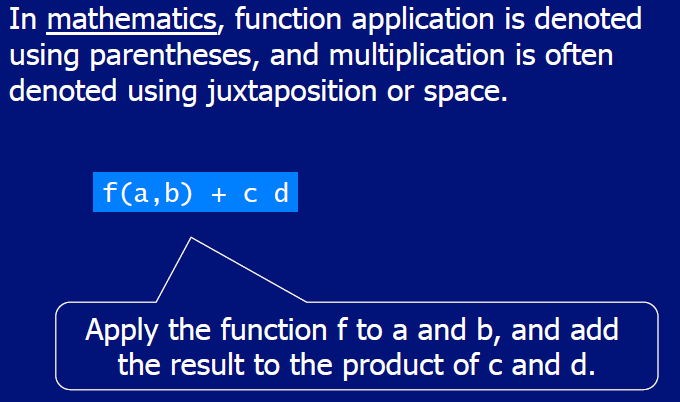
## Prelude

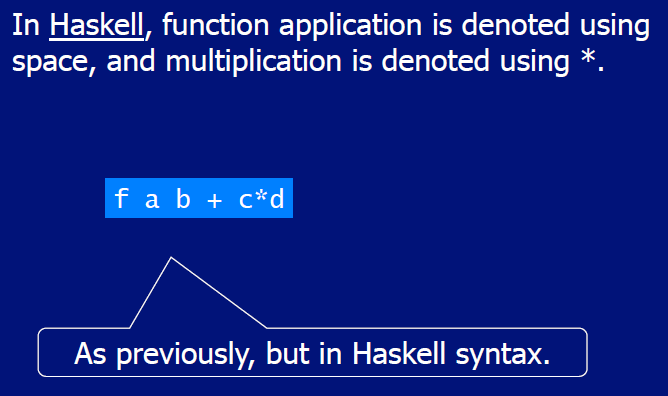
* Haskell comes with a large number of standard library functions. In addition to the familiar numeric functions such as + and \*, the library also provides many useful functions on lists.

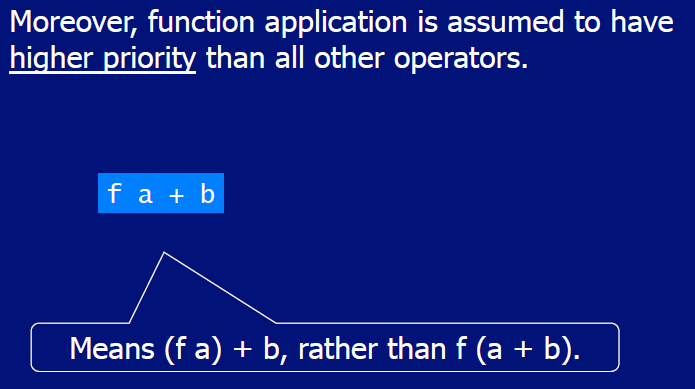
## Examples

* Select the first element of a list: head [1,2,3,4,5] 🡪 1
* Remove the first element of a list: tail [1,2,3,4,5] 🡪 [2,3,4,5]
* Select the nth element of a list: [1,2,3,4,5] !! 2 🡪 3
* Select the first n elements of a list: take 3 [1,2,3,4,5] 🡪 [1,2,3]
* Remove the first n elements from a list: drop 3 [1,2,3,4,5] 🡪 [4,5]
* Calculate the length of a list: length [1,2,3,4,5] 🡪 5
* Calculate the sum of a list of numbers: sum [1,2,3,4,5] 🡪 15
* Calculate the product of a list of numbers: product [1,2,3,4,5] 🡪 120
* Append two lists: [1,2,3] ++ [4,5] 🡪 [1,2,3,4,5]
* Reverse a list: reverse [1,2,3,4,5] 🡪 [5,4,3,2,1]

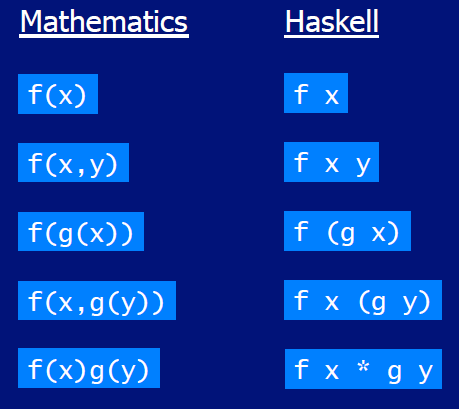
## Function Application



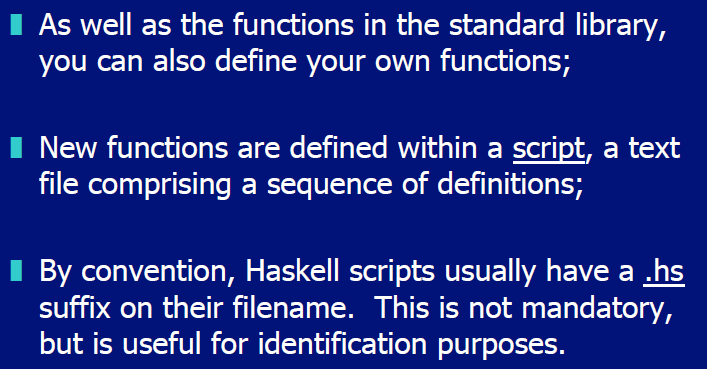




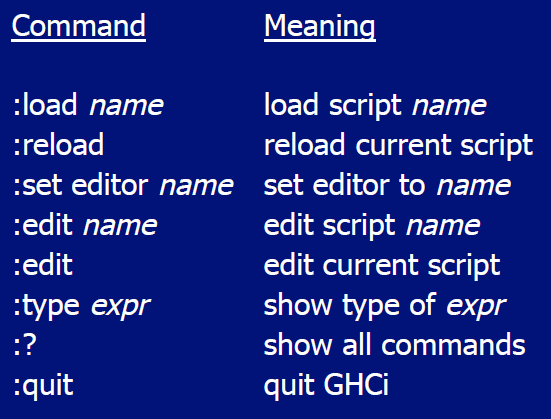
## Examples



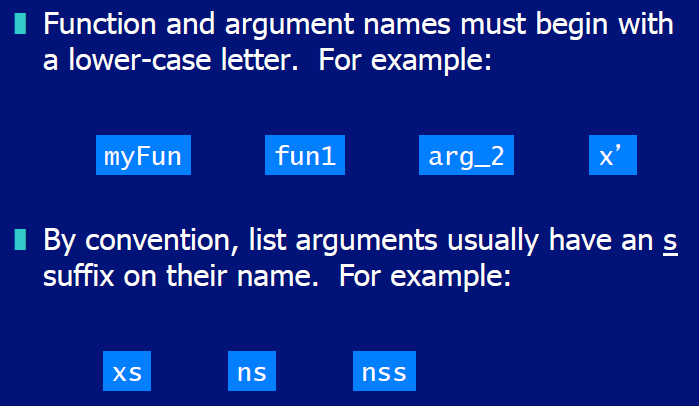
## Haskell Scripts



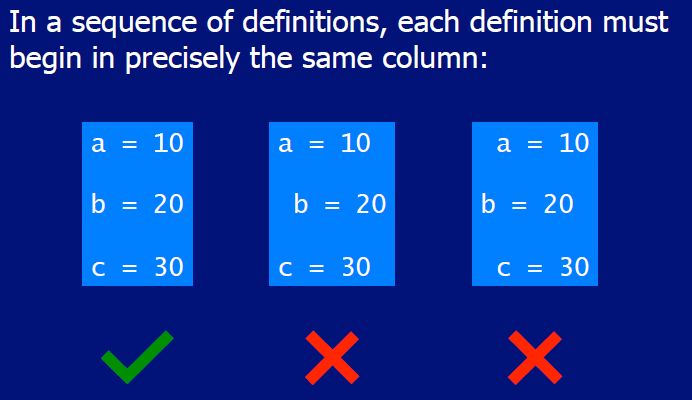
## GHCI Commands

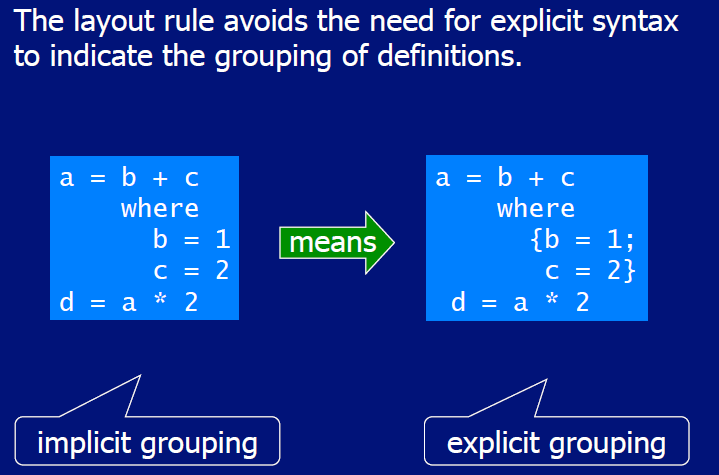


## Name Requirements



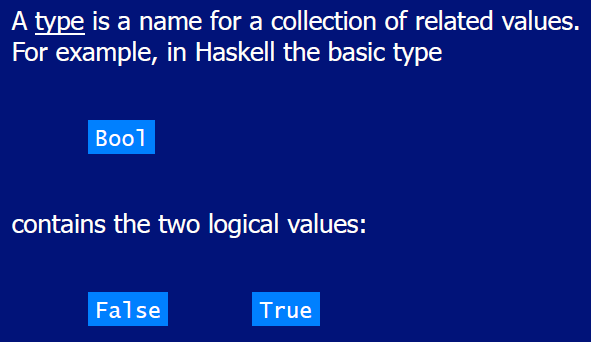
## Layout Rule



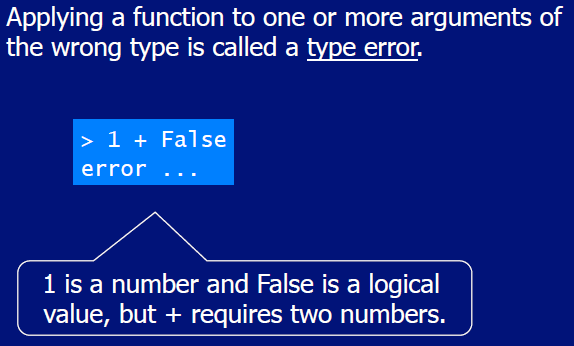


# Woche 2

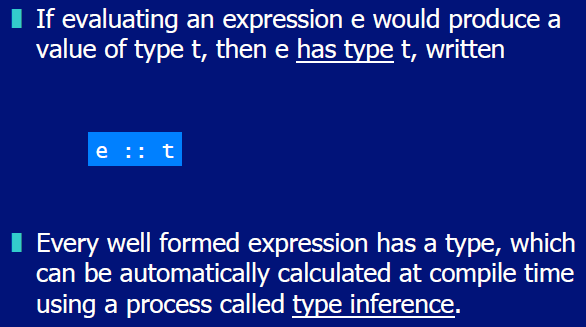
## What is a Type

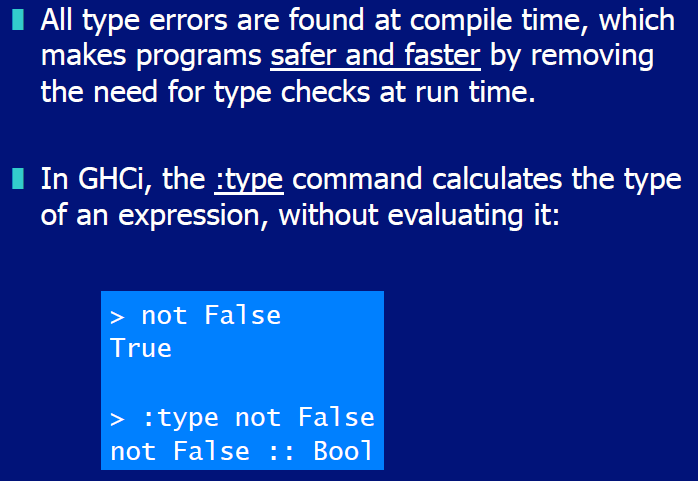


## Type Errors

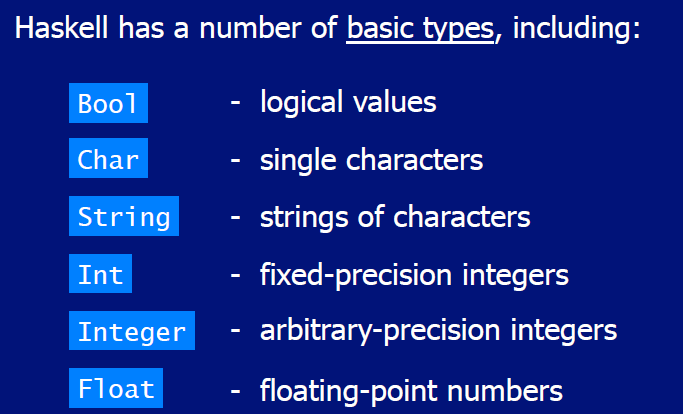


## Types in Haskell

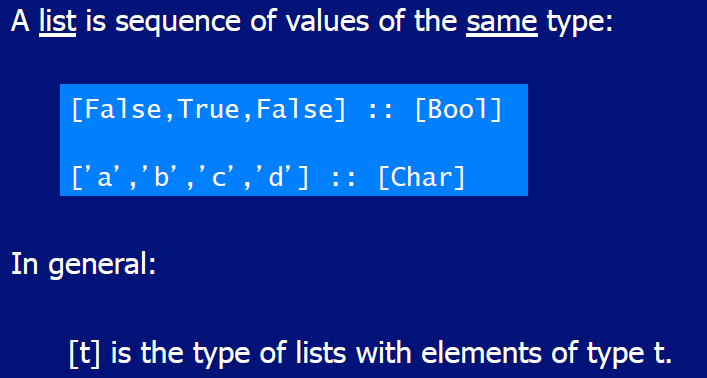


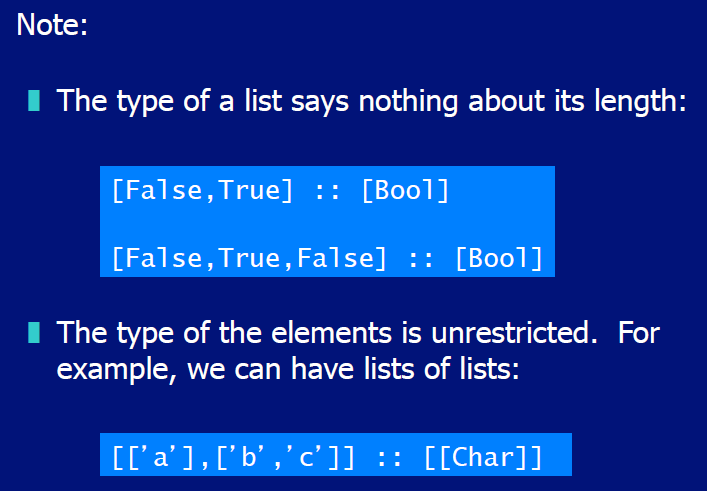


## Basic Types

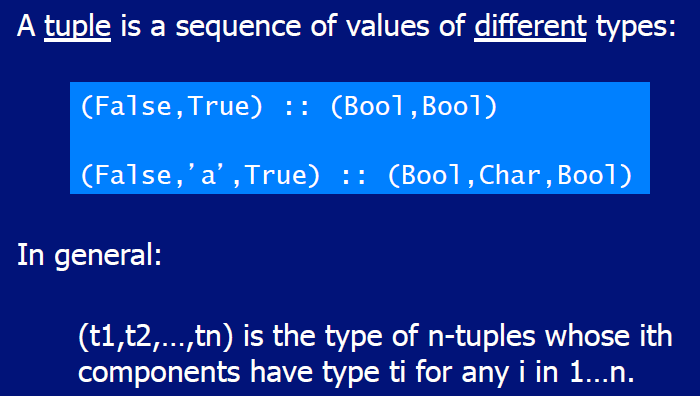


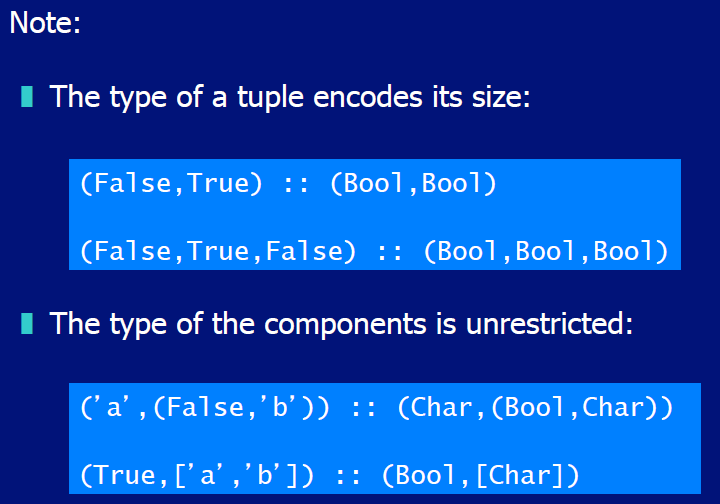
## List Types



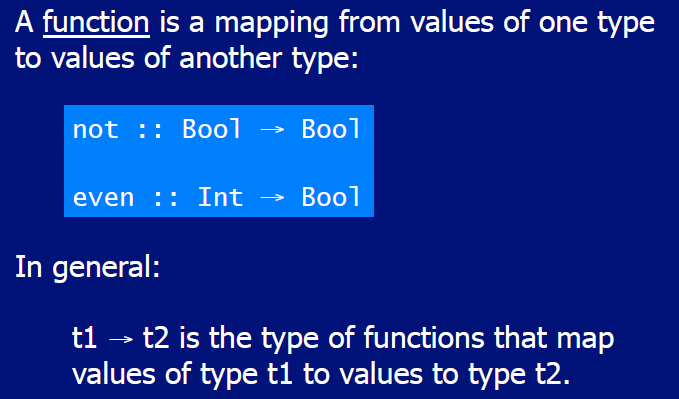


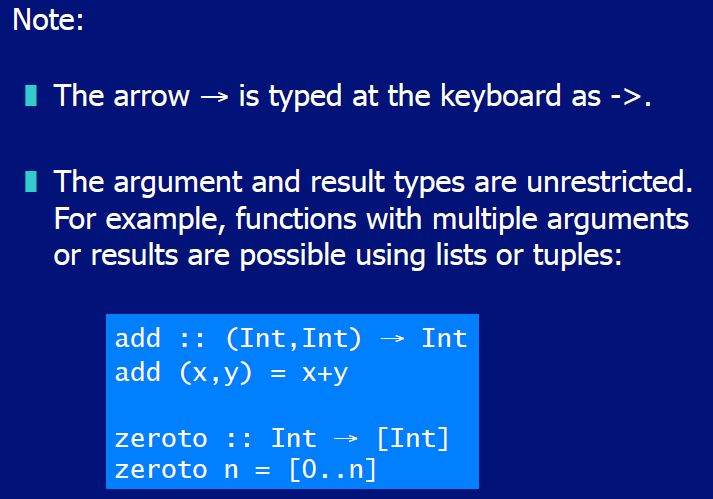
## Tuple Types



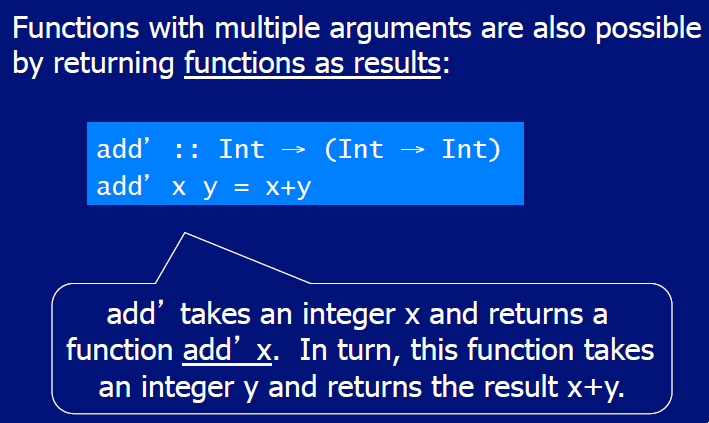


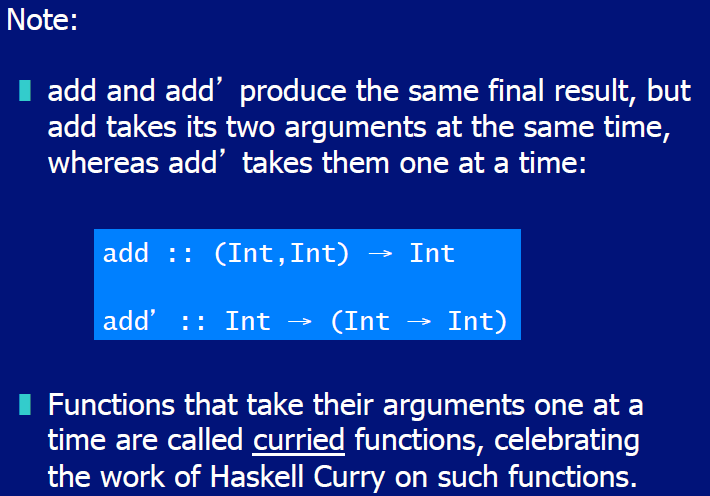
## Function Types

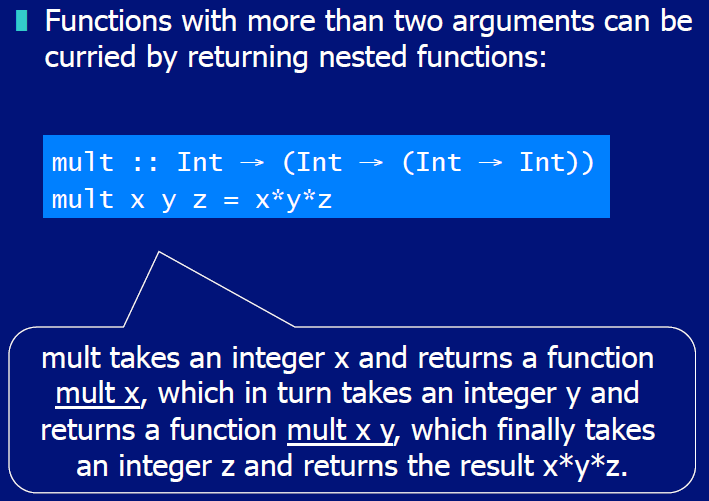




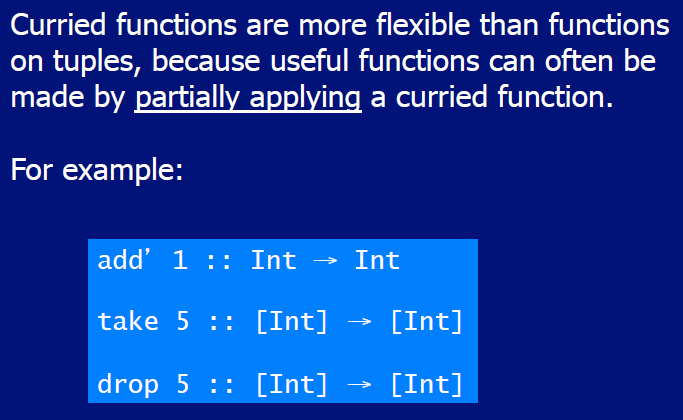
## Curried Functions



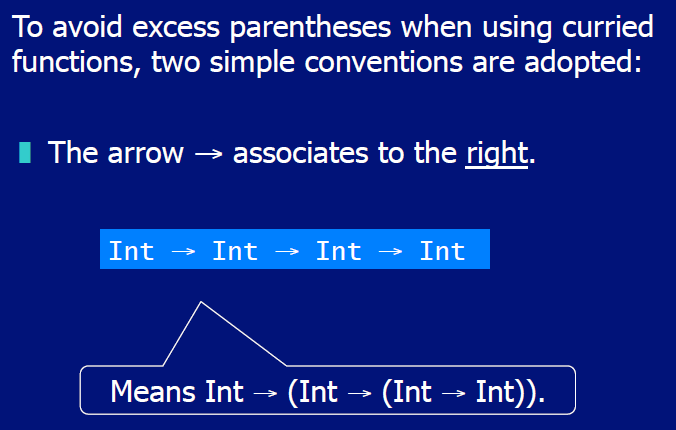


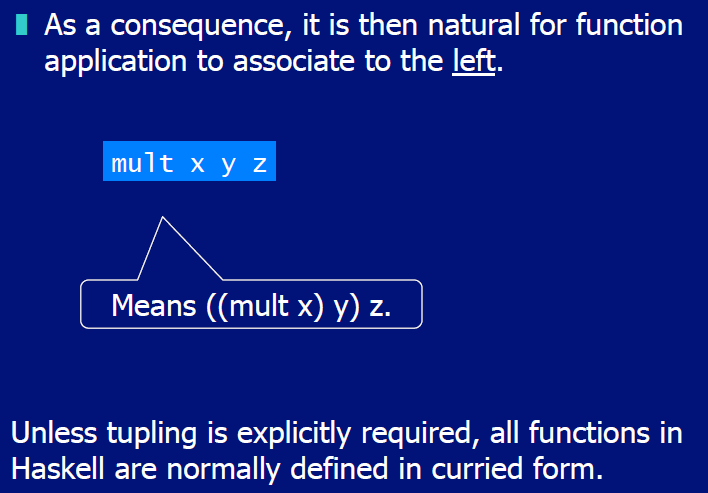


## Why is Currying useful

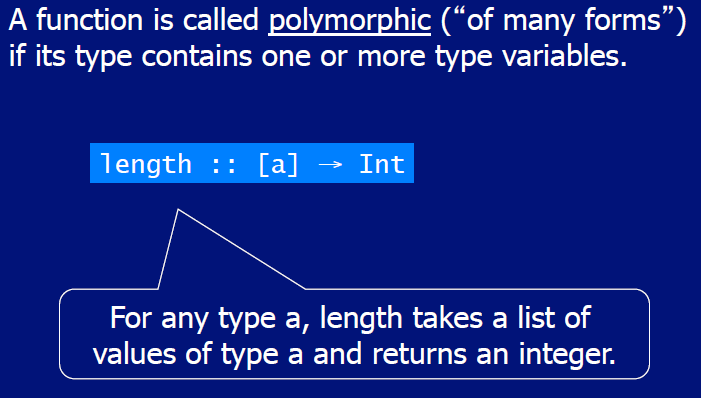


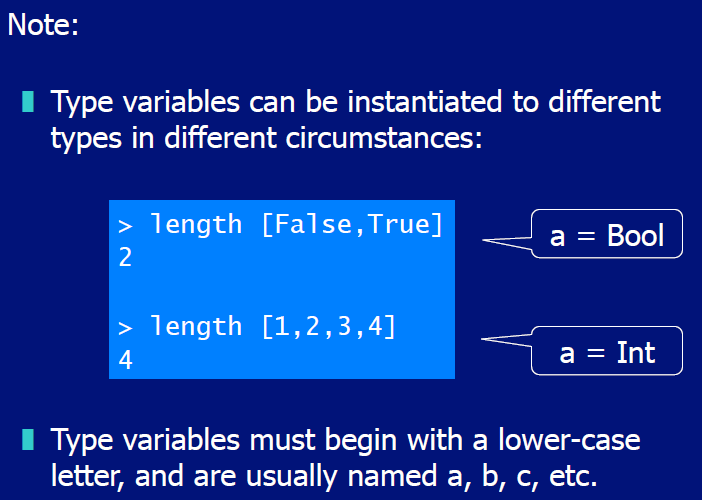
## Currying Conventions

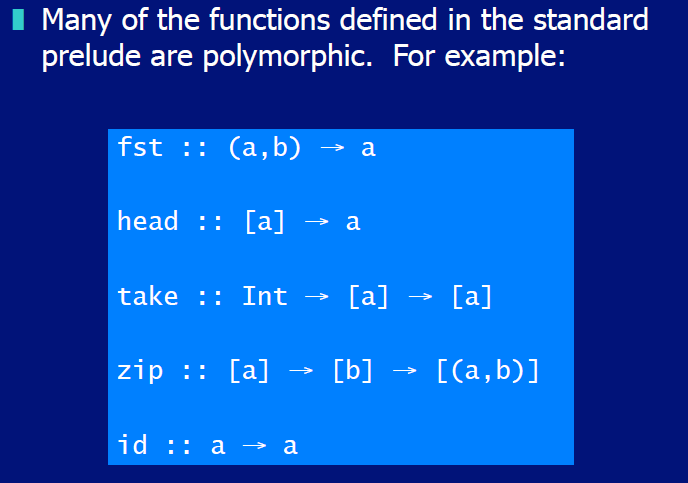




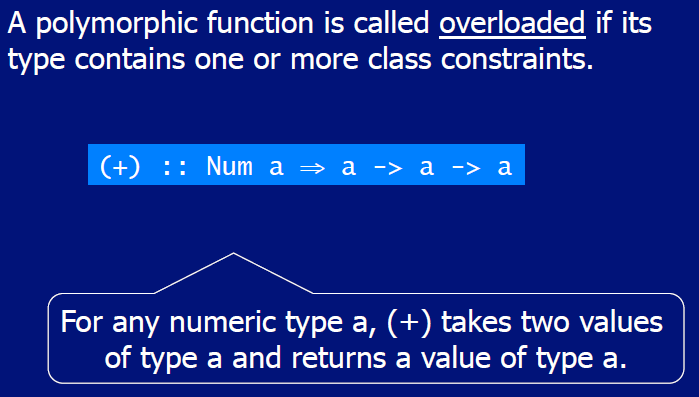
## Polymorphic Functions

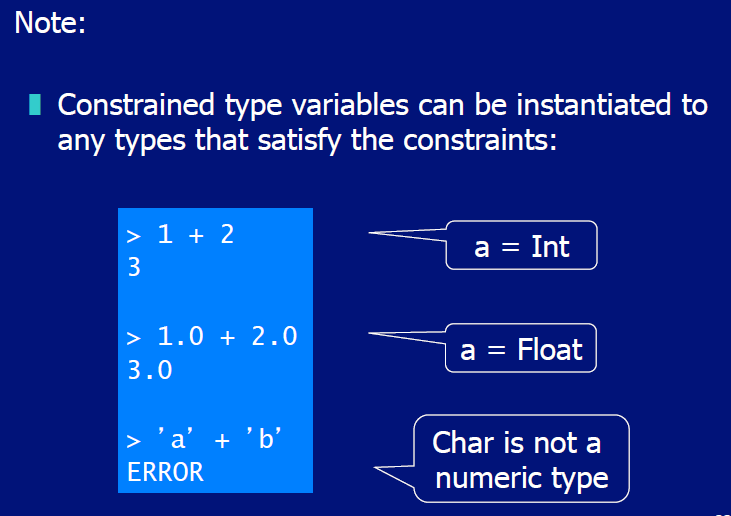


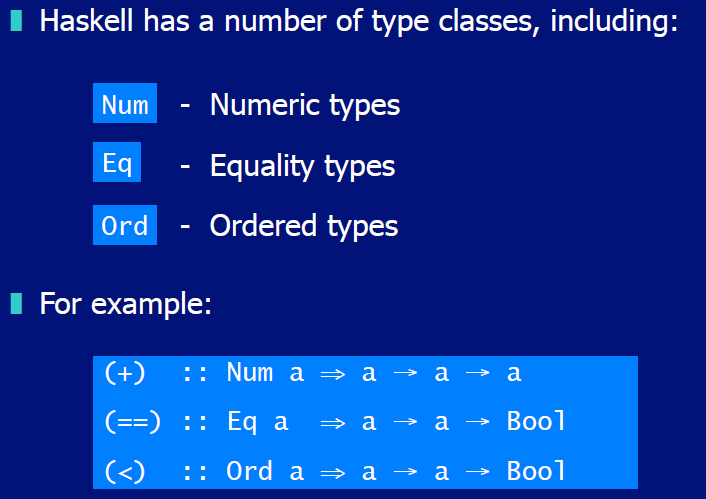




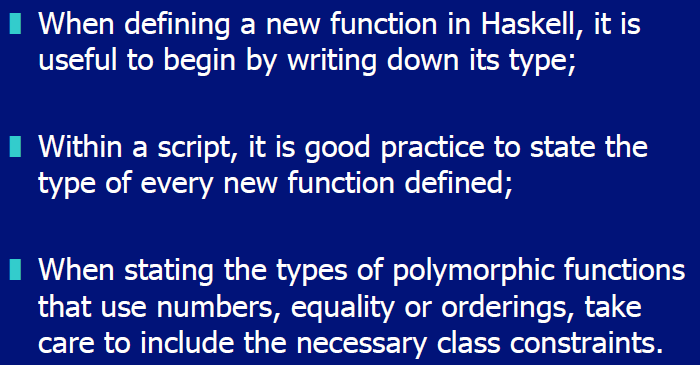
## Overloaded Functions







## Hints and Tips



# Woche 3