Moduldokumentation

Modul Functional Programming (fprog)

Simon Wächter

2016

Inhalt

[1 Einleitung 2](#_Toc462236318)

[1.1 Einleitung 2](#_Toc462236319)

[1.2 Lernziele 2](#_Toc462236320)

[1.3 Prüfungen 2](#_Toc462236321)

[2 Woche 1 3](#_Toc462236322)

[2.1 What is a Functional Language 3](#_Toc462236323)

[2.2 Example 3](#_Toc462236324)

[2.3 History 3](#_Toc462236325)

[2.4 Glasgow Haskell Compiler 4](#_Toc462236326)

[2.5 Prelude 4](#_Toc462236327)

[2.6 Examples 4](#_Toc462236328)

[2.7 Function Application 4](#_Toc462236329)

[2.8 Examples 5](#_Toc462236330)

[2.9 Haskell Scripts 5](#_Toc462236331)

[2.10 First Example 5](#_Toc462236332)

[2.11 GHCI Commands 6](#_Toc462236333)

[2.12 Name Requirements 6](#_Toc462236334)

[2.13 Layout Rule 6](#_Toc462236335)

[3 Woche 2 8](#_Toc462236336)

# 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

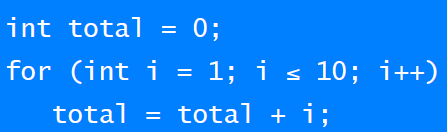
## What is a Functional Language

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

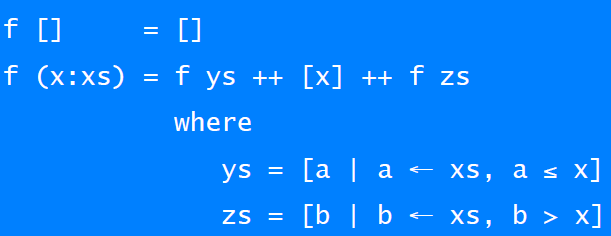
Summing the integers 1 to 10 in Java (The computation method is **variable assignment**):



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



Using the quicksort algorithm:



## 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

In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space (Apply the function to a and b, and add the result of the product of c and d):



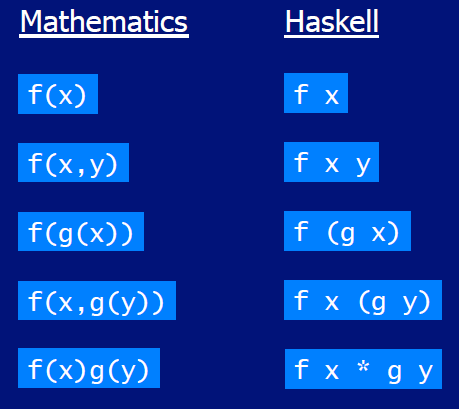
In Haskell, function application is denoted using space, and multiplication is denoted using \* (As previously, but in Haskell syntax):



Moreover, function application is assumed to have higher priority than all other operators (Means (f a) + b, rather than f(a + b)):



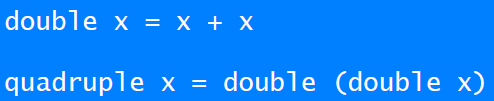
## Examples



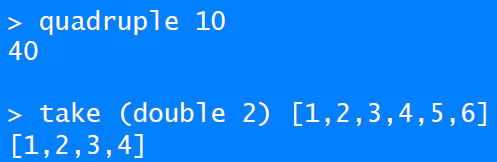
## Haskell Scripts

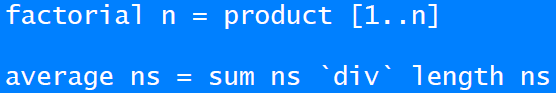
* As well as the functions in the standard library, you can also define your own functions;
* New functions are defined within a script, a text file comprising a sequence of definitions;
* By convention, Haskell scripts usually have a .hs suffix on their filename. This is not mandatory, but is useful for identification purposes

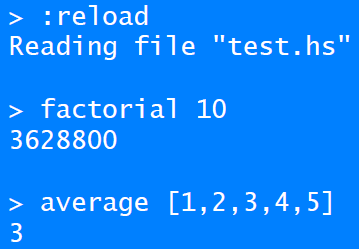
## First Example



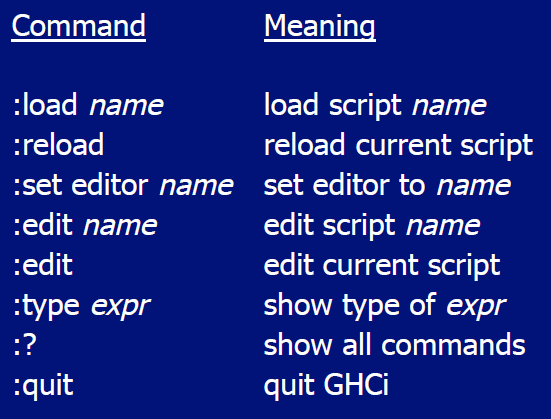








## GHCI Commands

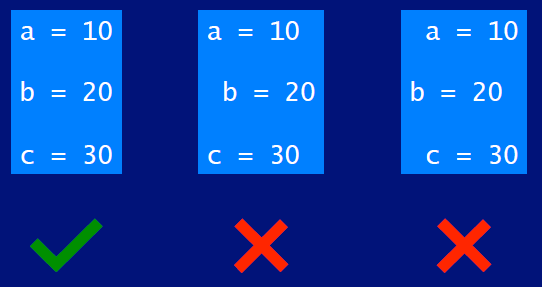


## Name Requirements

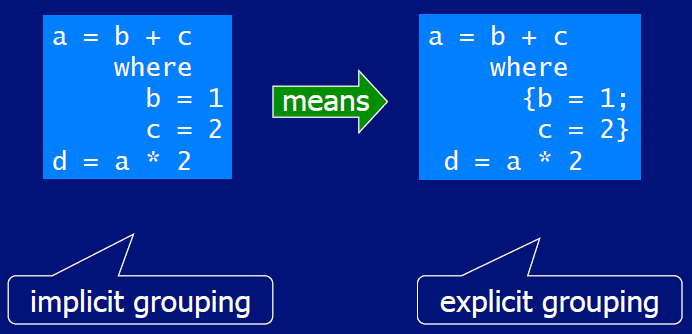
* Function and argument names must begin with a lower-case letter. For example
  + myFun
  + fun1
  + arg\_2
  + x’
* By convention, list arguments usually have an s suffix on their name. For example:
  + xs
  + ns
  + nss

## Layout Rule

In a sequence of definition, each definition must begin in precisely the same column:



The layout rule avoids the need for explicit syntax to indicate the grouping of definitions:



# Woche 2