Moduldokumentation

Modul Functional Programming (fprog)

Simon Wächter

2016

Inhalt

[1 Einleitung 3](#_Toc467341448)

[1.1 Einleitung 3](#_Toc467341449)

[1.2 Lernziele 3](#_Toc467341450)

[1.3 Prüfungen 3](#_Toc467341451)

[2 Woche 1 4](#_Toc467341452)

[2.1 What is a Functional Language 4](#_Toc467341453)

[2.2 Example 4](#_Toc467341454)

[2.3 History 4](#_Toc467341455)

[2.4 Glasgow Haskell Compiler 4](#_Toc467341456)

[2.5 Prelude 4](#_Toc467341457)

[2.6 Examples 5](#_Toc467341458)

[2.7 Function Application 5](#_Toc467341459)

[2.8 Examples 6](#_Toc467341460)

[2.9 Haskell Scripts 6](#_Toc467341461)

[2.10 GHCI Commands 6](#_Toc467341462)

[2.11 Name Requirements 7](#_Toc467341463)

[2.12 Layout Rule 7](#_Toc467341464)

[3 Woche 2 8](#_Toc467341465)

[3.1 What is a Function? 8](#_Toc467341466)

[3.2 What is a Partial Function? 10](#_Toc467341467)

[3.3 Domain of Definition 10](#_Toc467341468)

[3.4 Confusion 10](#_Toc467341469)

[3.5 Additional types 11](#_Toc467341470)

[3.6 Summary 11](#_Toc467341471)

[4 Woche 3 12](#_Toc467341472)

[4.1 What is a Type 12](#_Toc467341473)

[4.2 Type Errors 12](#_Toc467341474)

[4.3 Types in Haskell 12](#_Toc467341475)

[4.4 Basic Types 13](#_Toc467341476)

[4.5 List Types 13](#_Toc467341477)

[4.6 Tuple Types 14](#_Toc467341478)

[4.7 Function Types 15](#_Toc467341479)

[4.8 Curried Functions 15](#_Toc467341480)

[4.9 Why is Currying useful 16](#_Toc467341481)

[4.10 Currying Conventions 17](#_Toc467341482)

[4.11 Polymorphic Functions 17](#_Toc467341483)

[4.12 Overloaded Functions 18](#_Toc467341484)

[4.13 Hints and Tips 19](#_Toc467341485)

[5 Woche 4 20](#_Toc467341486)

[5.1 Typeclasses 20](#_Toc467341487)

[6 Woche 5 21](#_Toc467341488)

[6.1 Conditional Expressions 21](#_Toc467341489)

[6.2 Guarded Equations 21](#_Toc467341490)

[6.3 Pattern Matching 22](#_Toc467341491)

[6.4 List Patterns 23](#_Toc467341492)

[7 Woche 6 25](#_Toc467341493)

[7.1 Lambda Expressions 25](#_Toc467341494)

[7.2 Why are Lambdas useful 25](#_Toc467341495)

[7.3 Operator Sections 27](#_Toc467341496)

[7.4 Why are Sections useful 27](#_Toc467341497)

[7.5 Set Comprehensions 28](#_Toc467341498)

[7.6 List Comprehensions 28](#_Toc467341499)

[7.7 Dependent Generators 29](#_Toc467341500)

[7.8 Guards 30](#_Toc467341501)

[8 Woche 7 32](#_Toc467341502)

[8.1 Zip Function 32](#_Toc467341503)

[8.2 String Comparison 33](#_Toc467341504)

[9 Woche 8 35](#_Toc467341505)

[10 Woche 9 36](#_Toc467341506)

[10.1 Introduction to Recursion 36](#_Toc467341507)

[10.2 Recursive Functions 36](#_Toc467341508)

[10.3 Why is Recursion useful 37](#_Toc467341509)

[10.4 Recursion on Lists 38](#_Toc467341510)

[10.5 Multiple Arguments 40](#_Toc467341511)

[10.6 Quicksort 40](#_Toc467341512)

[11 Woche 10 42](#_Toc467341513)

# 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

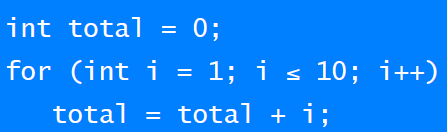
## What is a Functional Language

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;

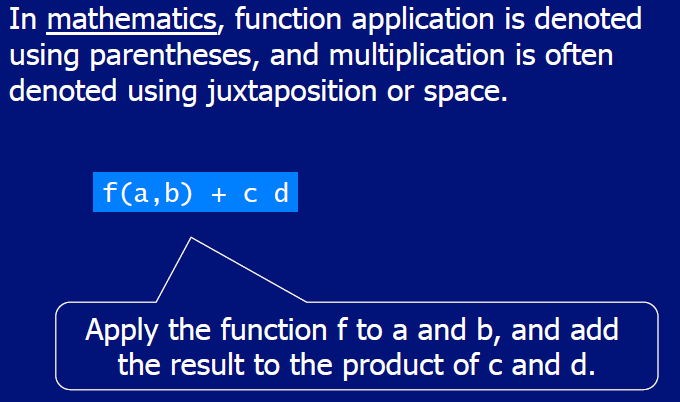
## Prelude

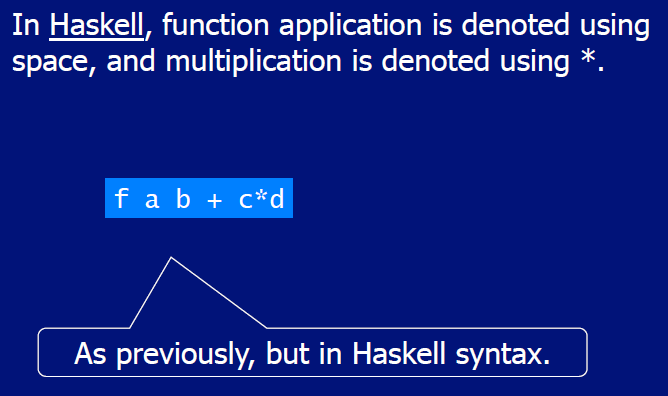
* Haskell comes with a large number of standard library functions. In addition to the familiar numeric functions such as + and \*, the library also provides many useful functions on lists.

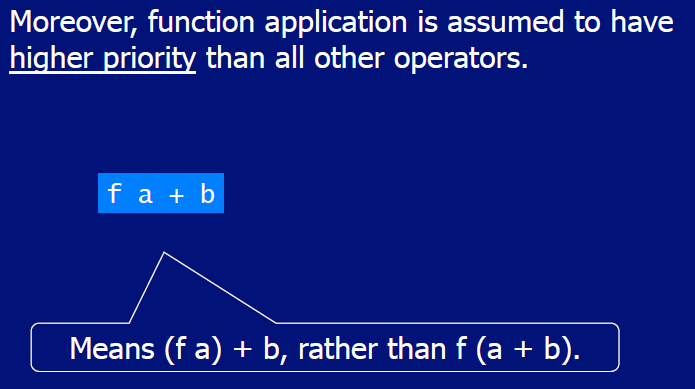
## Examples

* 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]

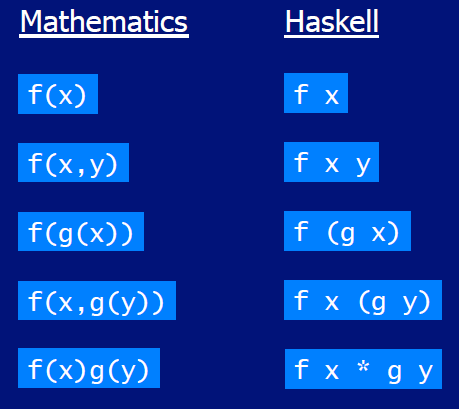
## Function Application



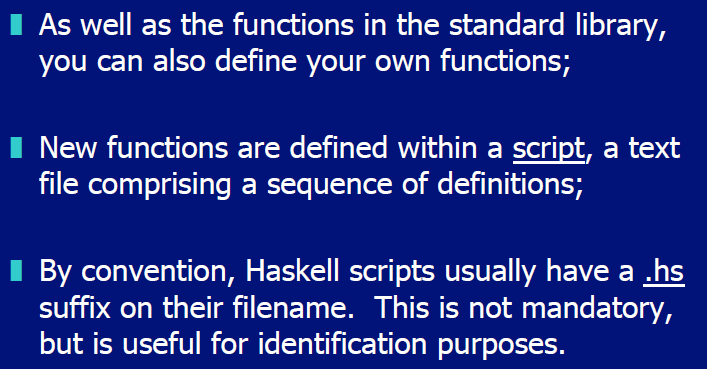




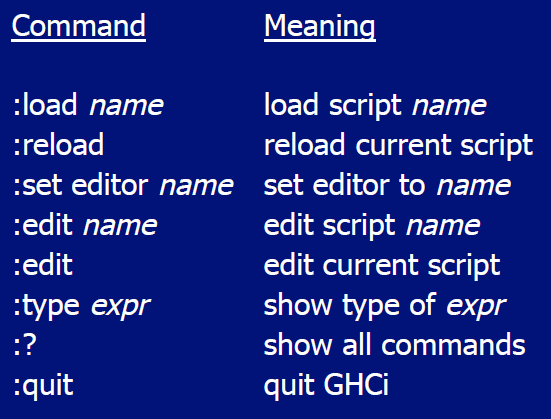
## Examples



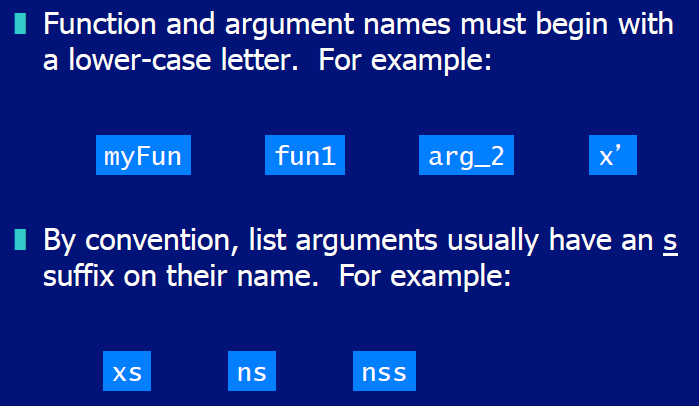
## Haskell Scripts



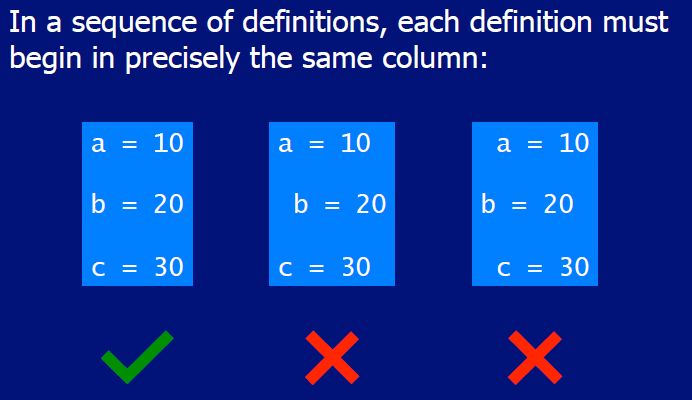
## GHCI Commands

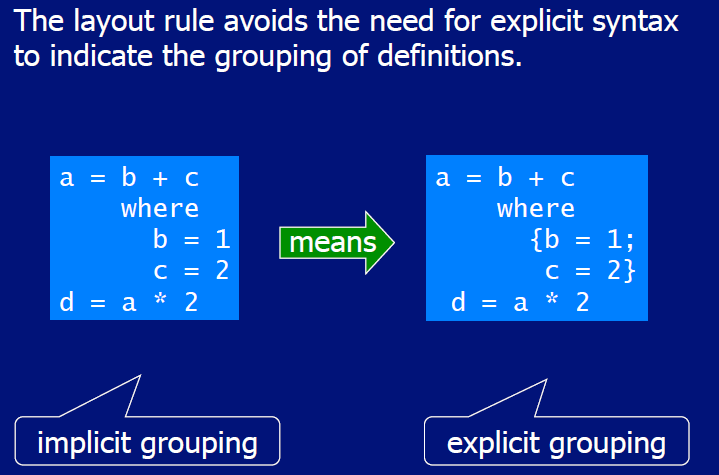


## Name Requirements



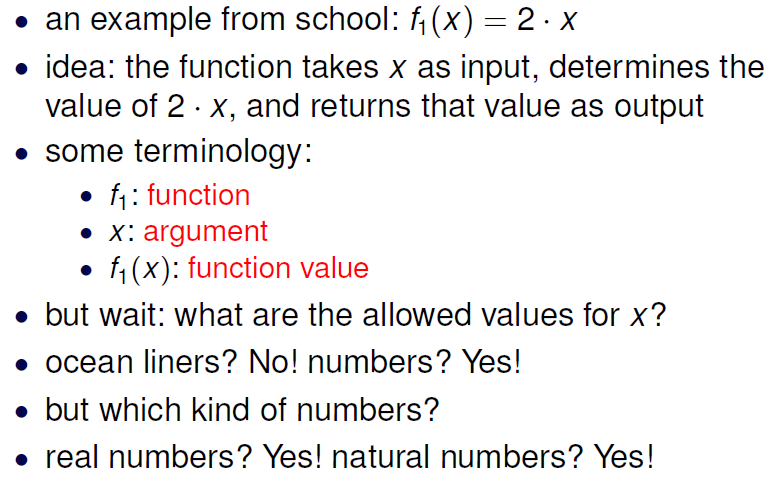
## Layout Rule

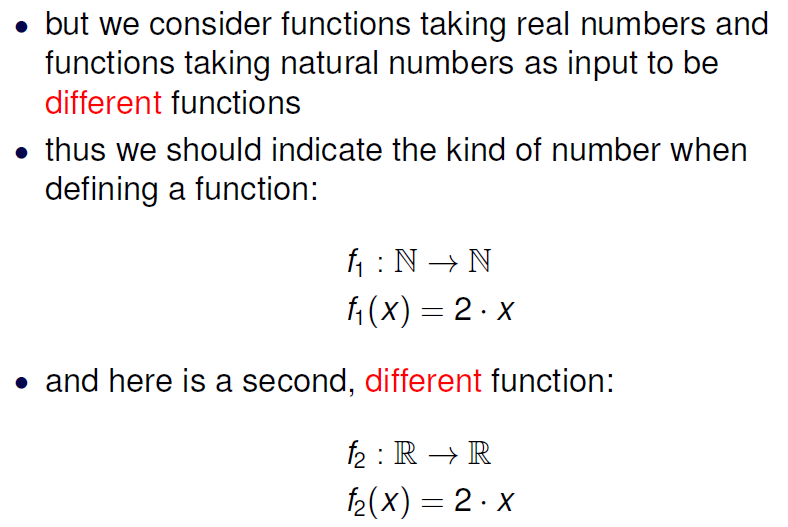


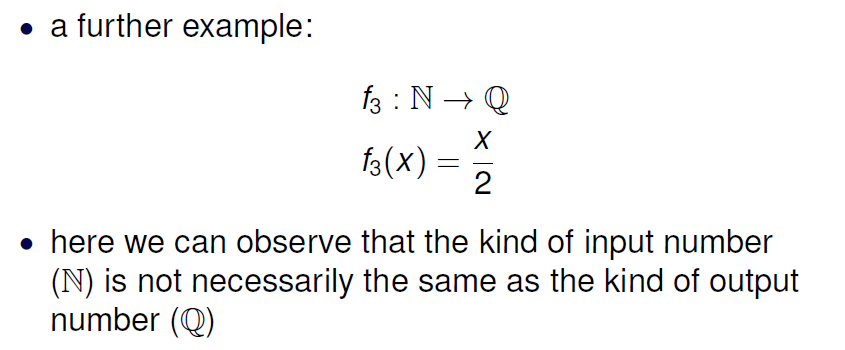


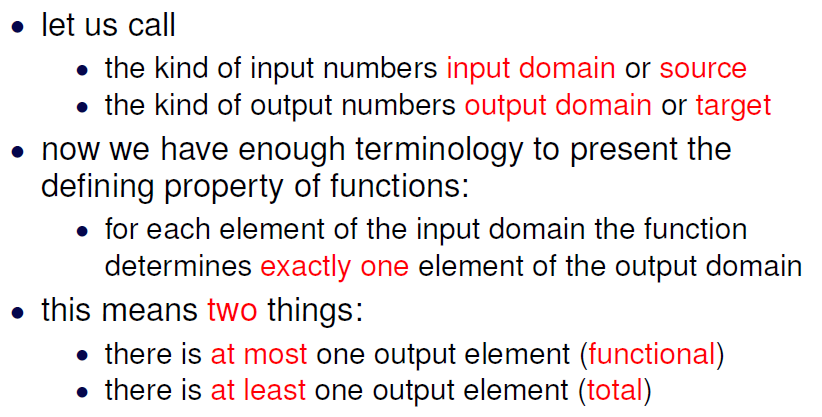
# Woche 2

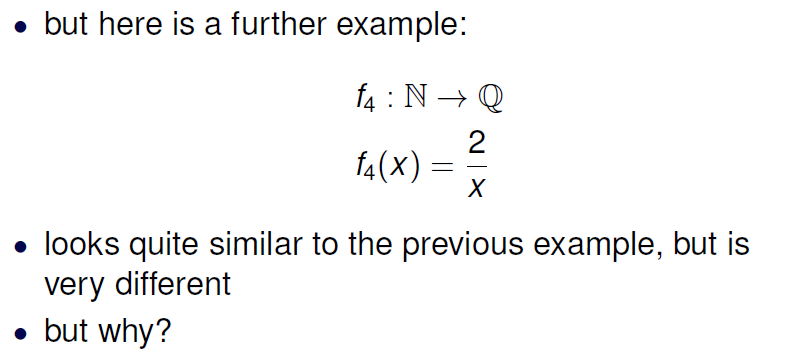
## What is a Function?

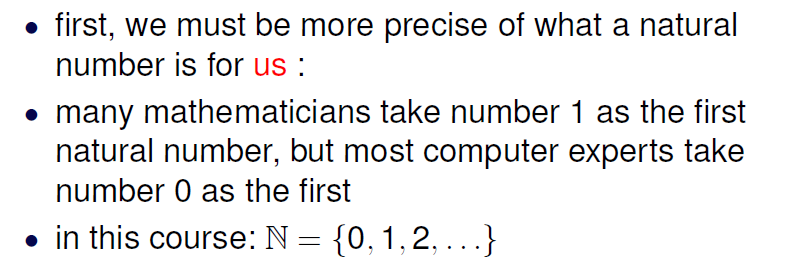


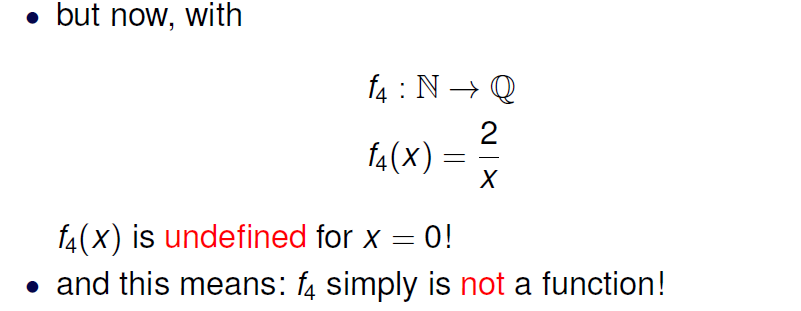


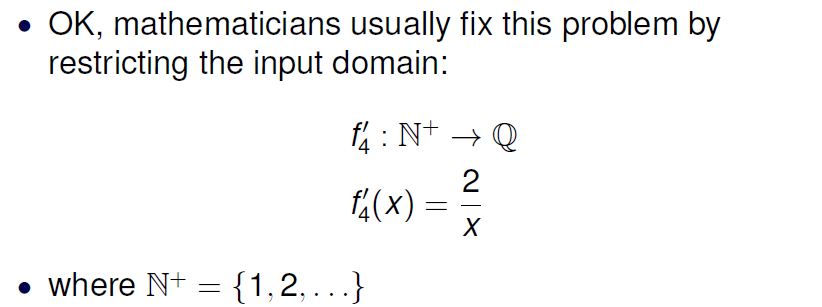




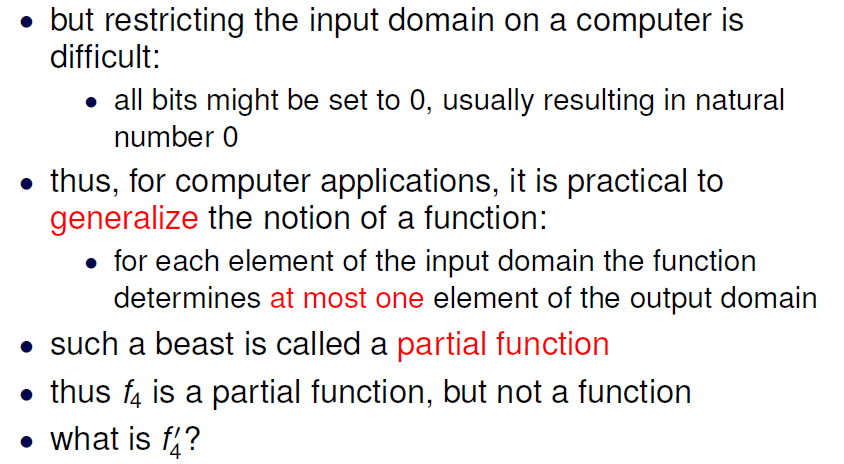




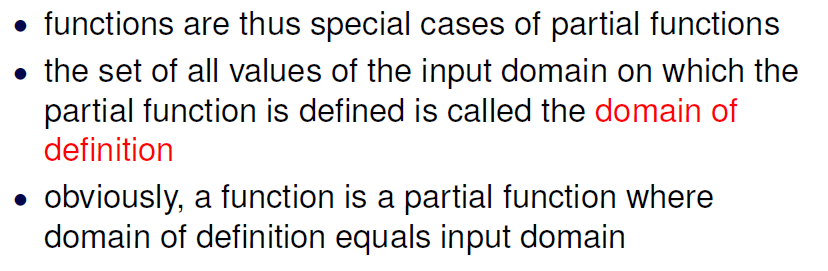




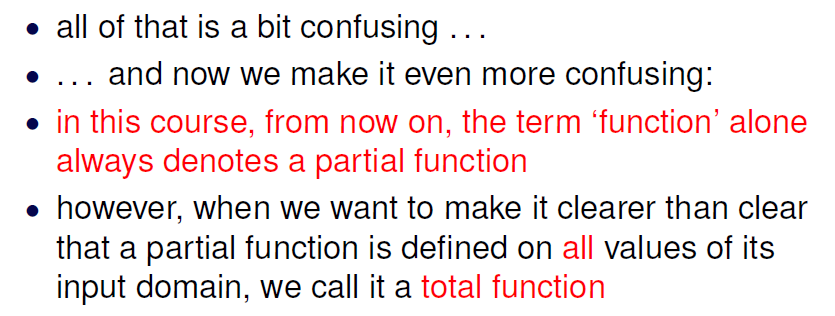
## What is a Partial Function?



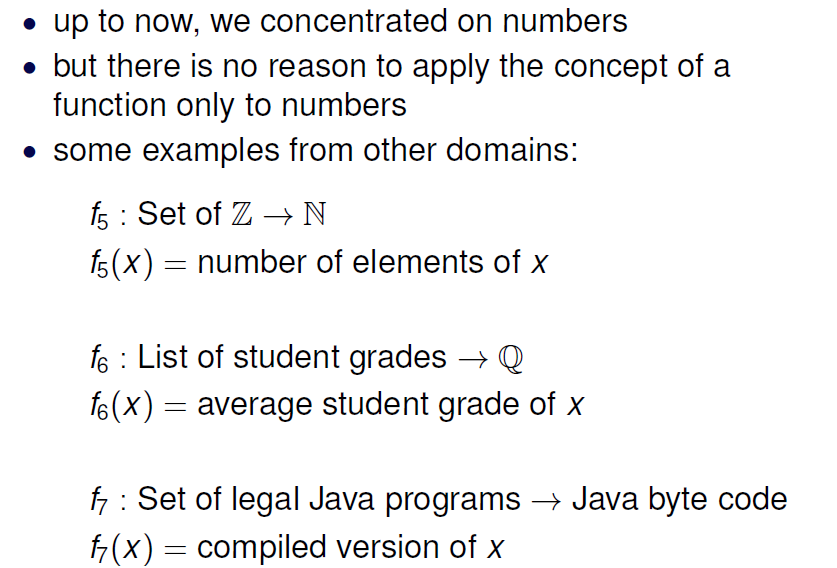
## Domain of Definition

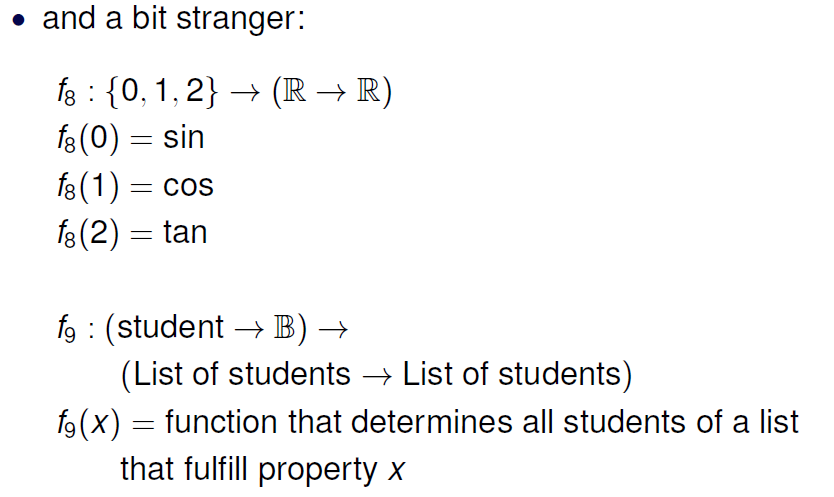


## Confusion



## Additional types



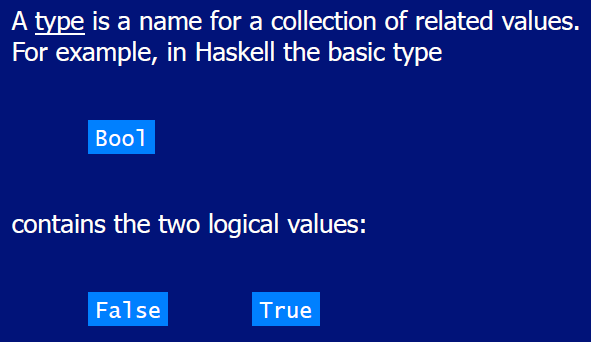


## Summary

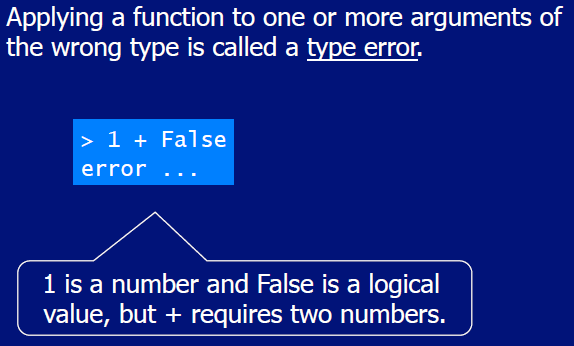


# Woche 3

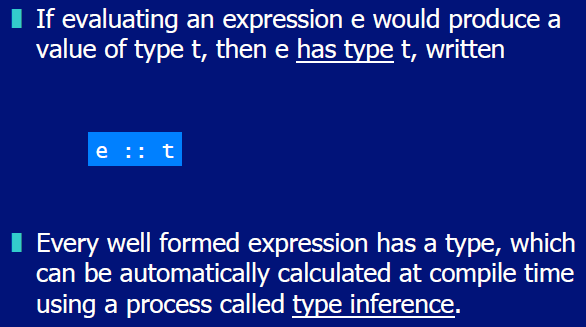
## What is a Type

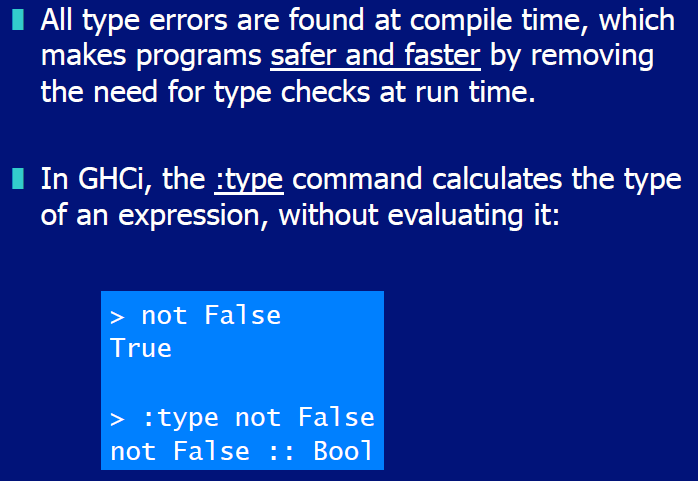


## Type Errors

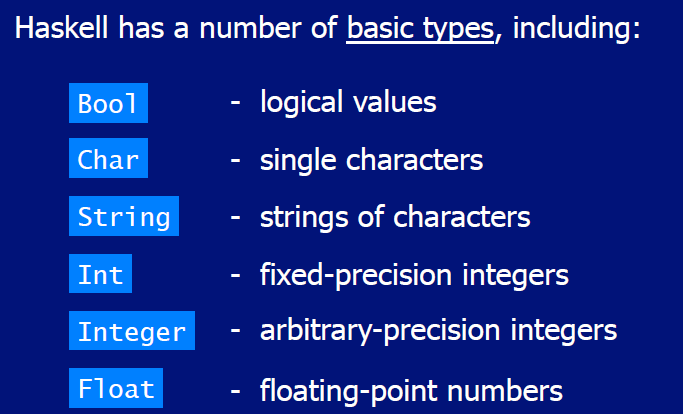


## Types in Haskell

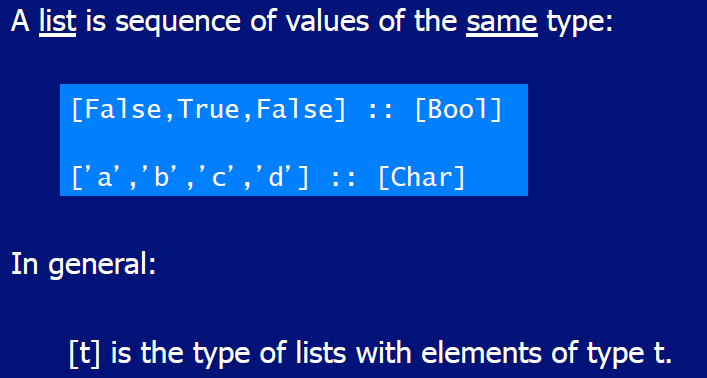


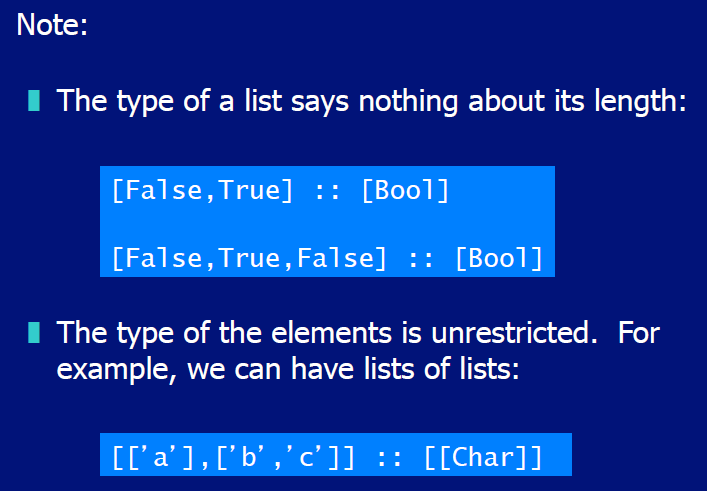


## Basic Types

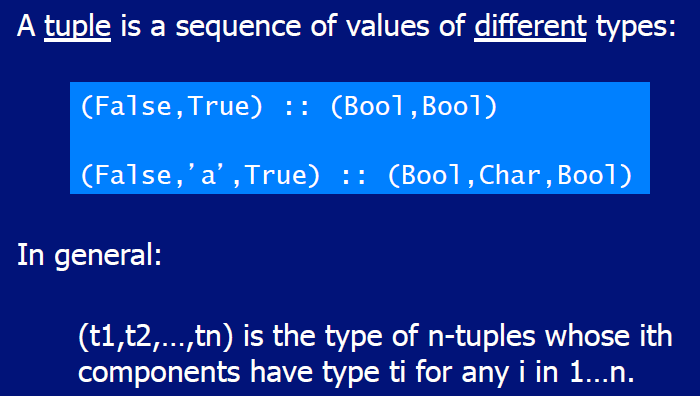


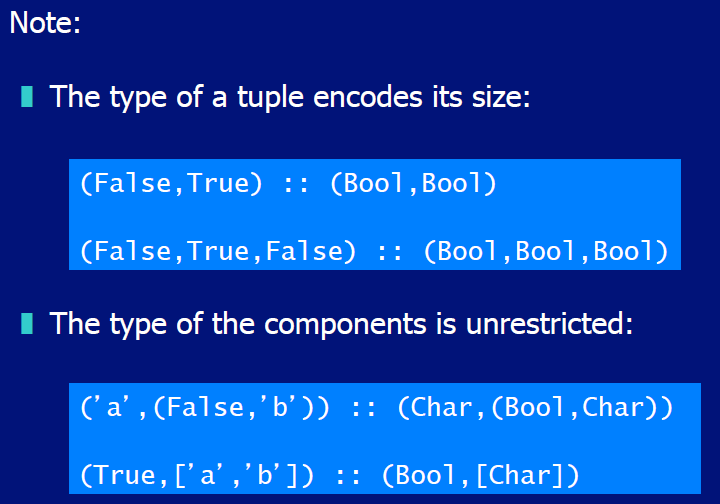
## List Types



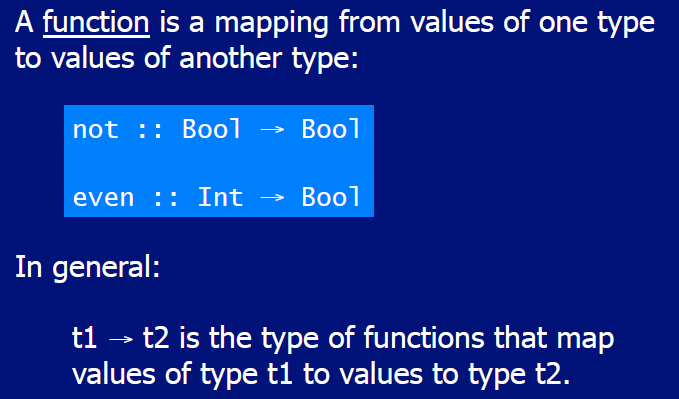


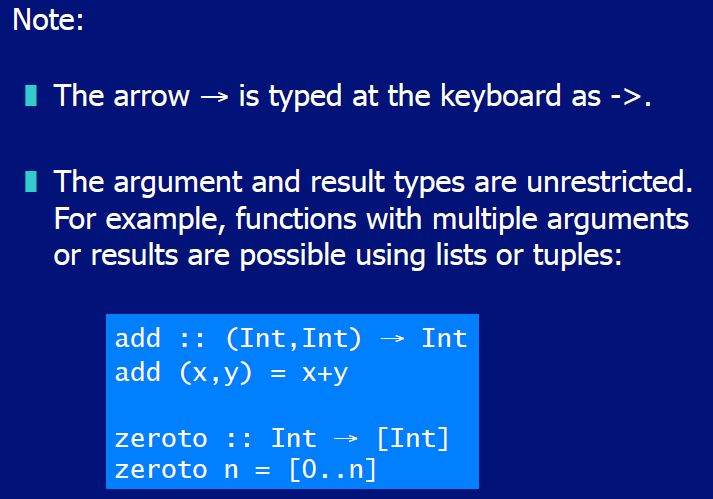
## Tuple Types



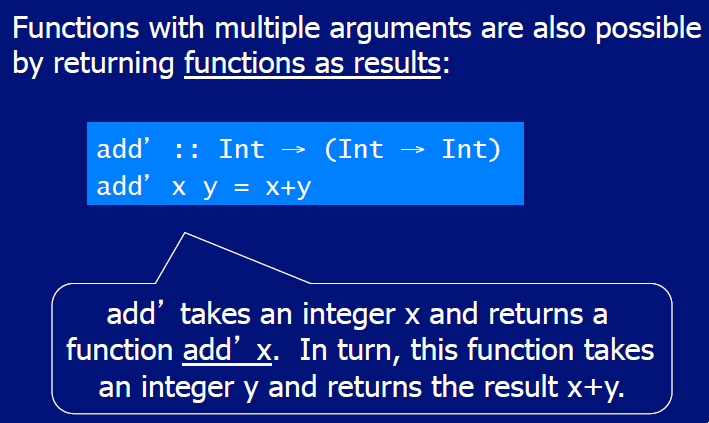


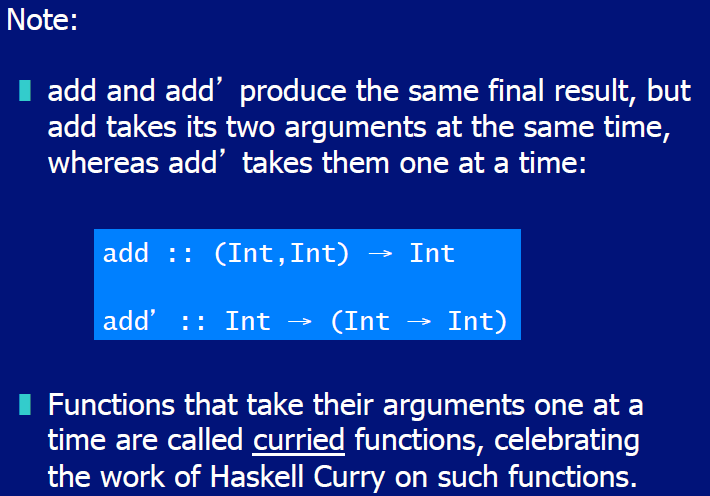
## Function Types

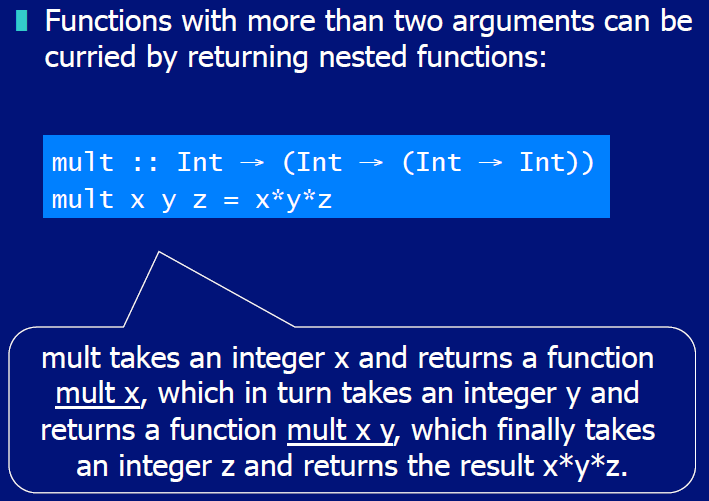




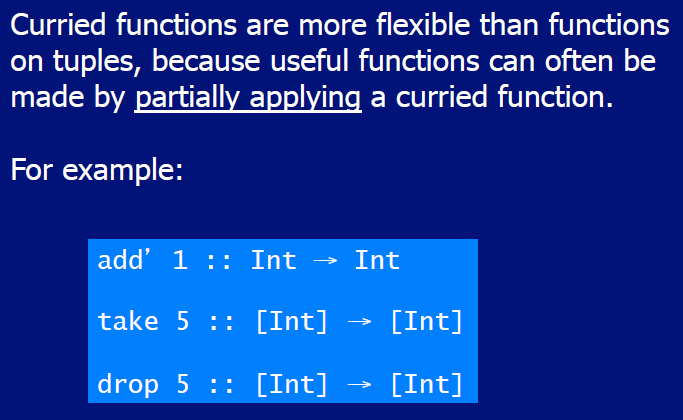
## Curried Functions



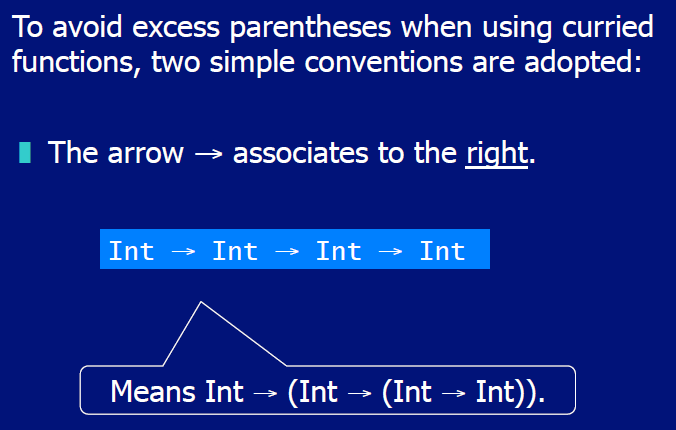


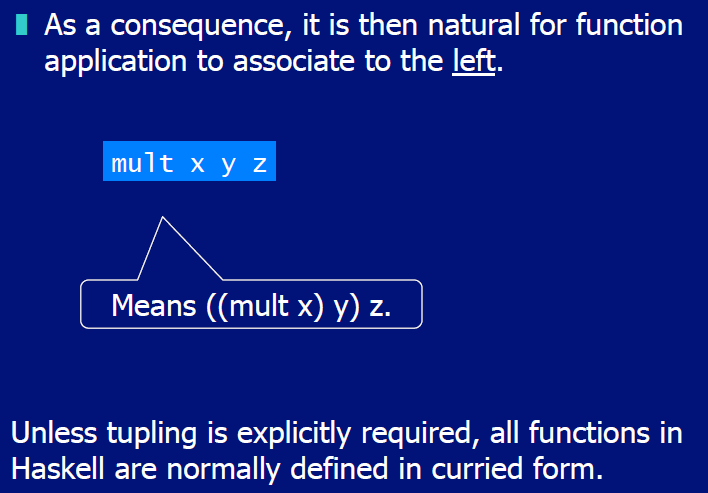


## Why is Currying useful

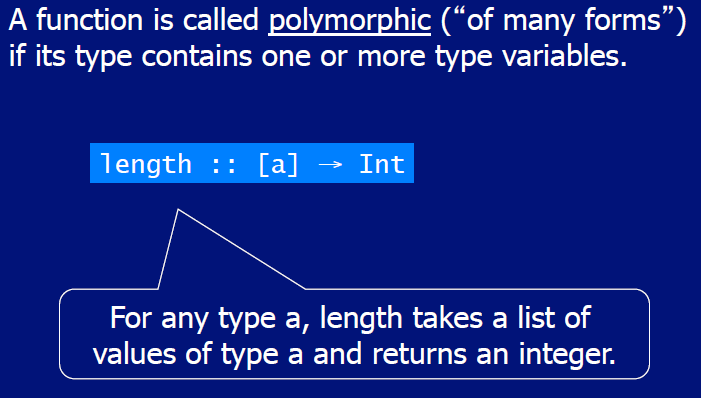


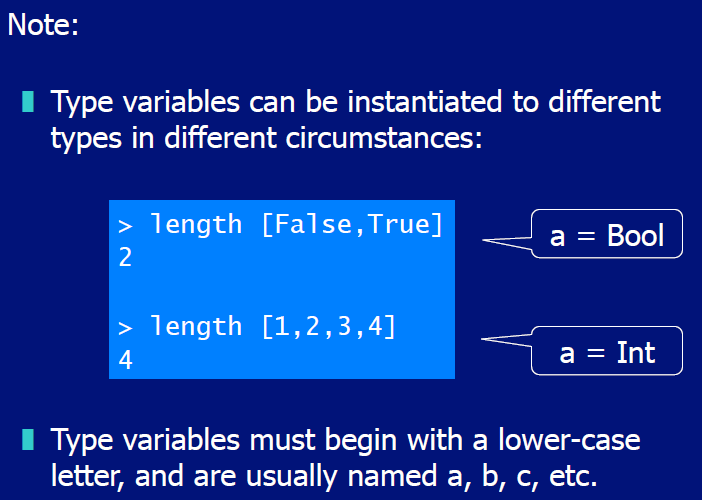
## Currying Conventions

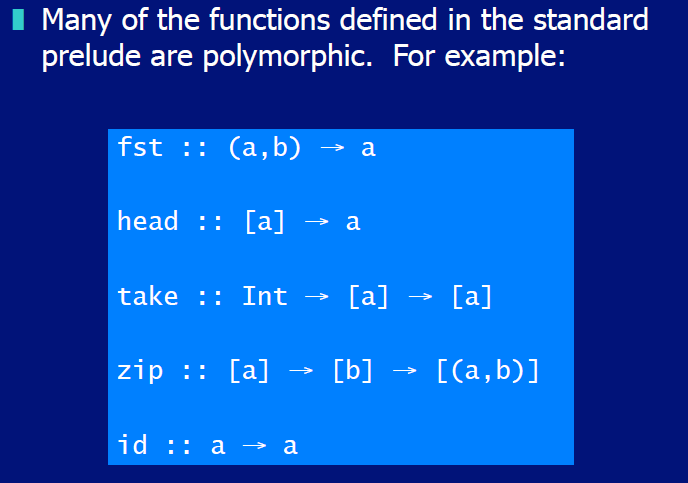




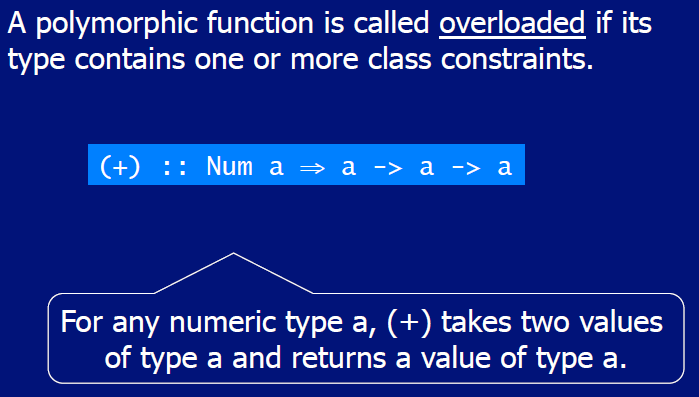
## Polymorphic Functions

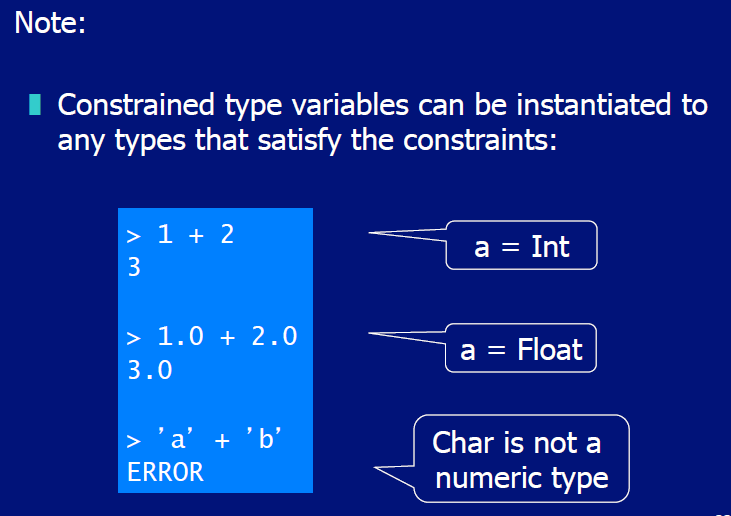


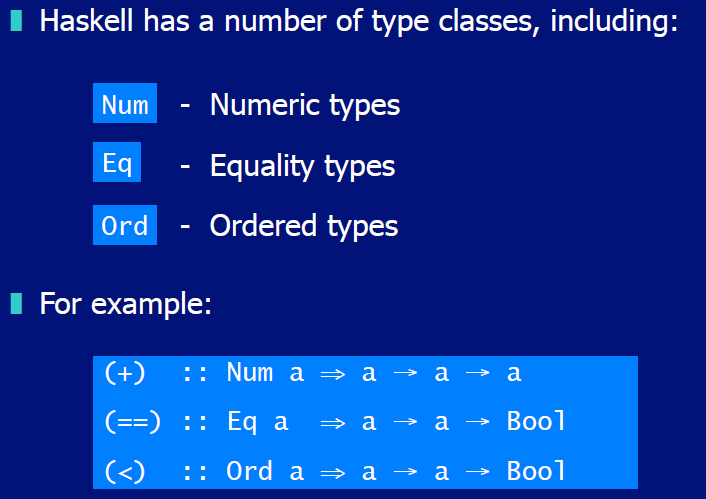




## Overloaded Functions





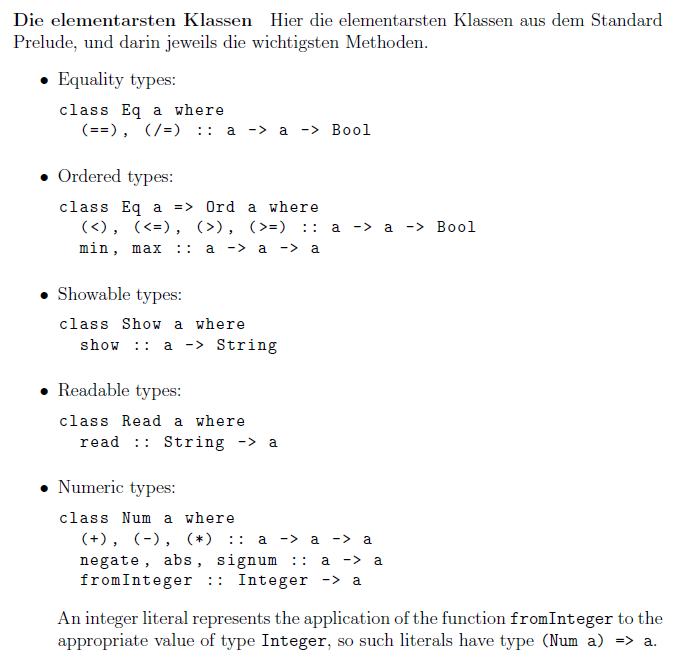


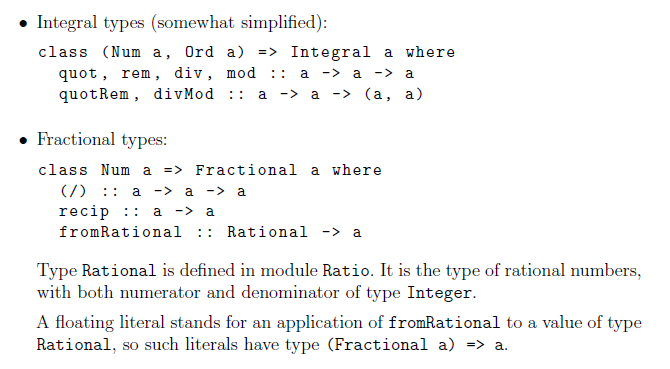
## Hints and Tips



# Woche 4

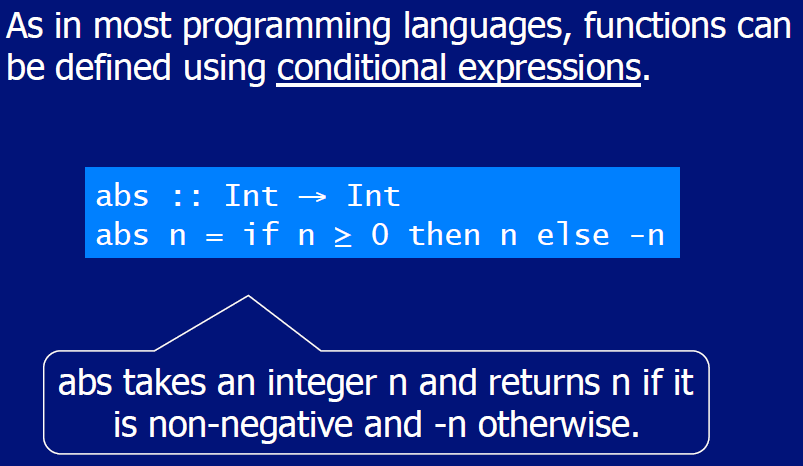
## Typeclasses

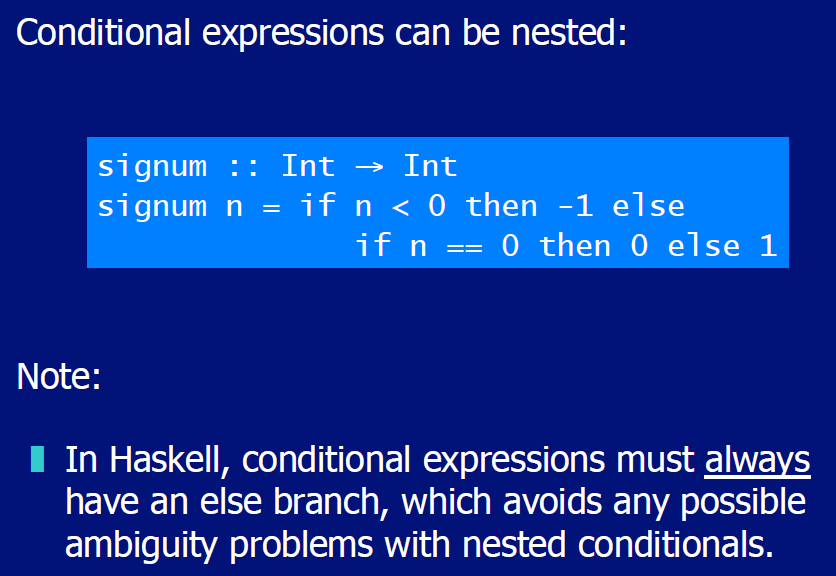




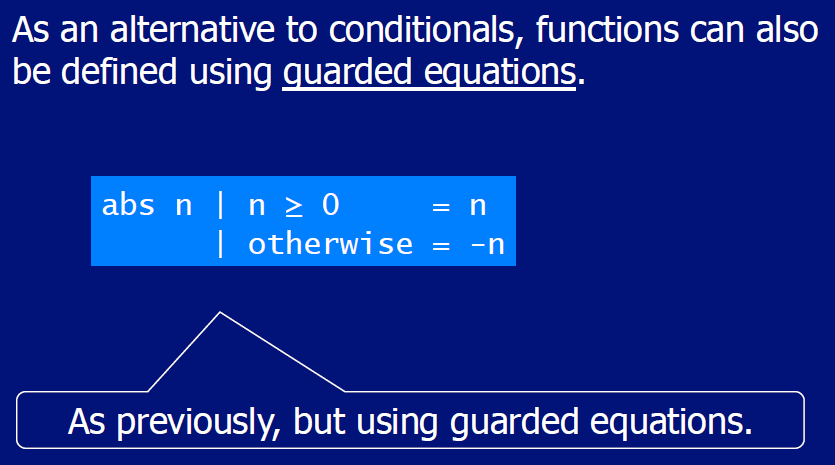
# Woche 5

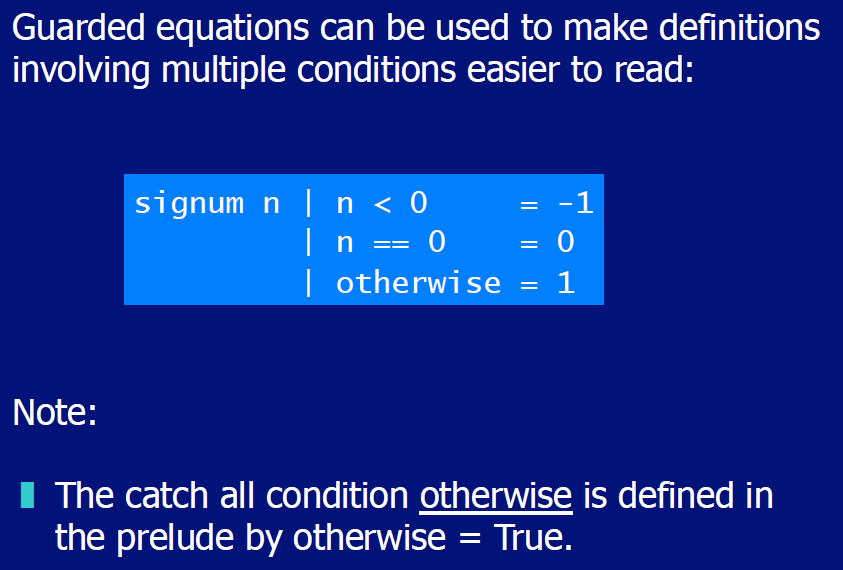
## Conditional Expressions



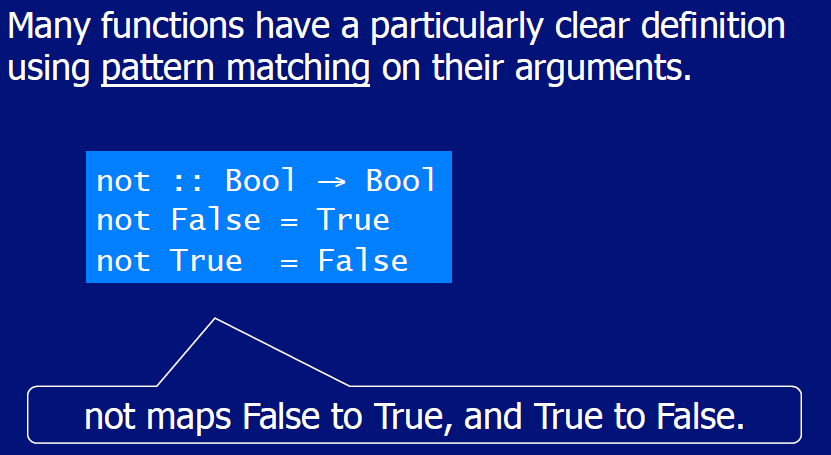


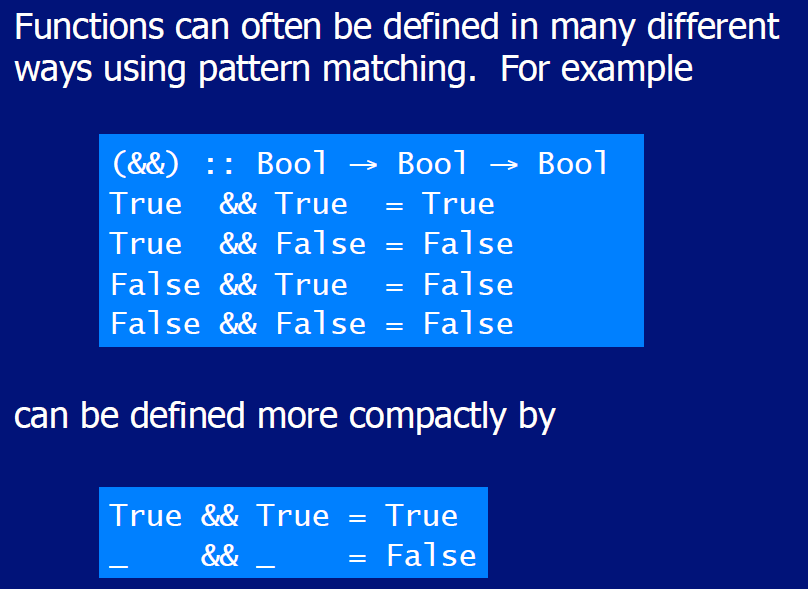
## Guarded Equations

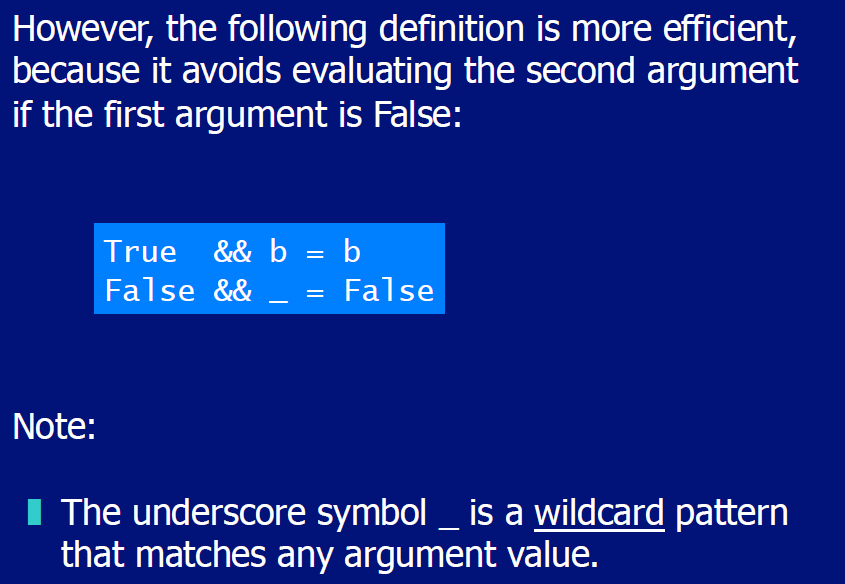


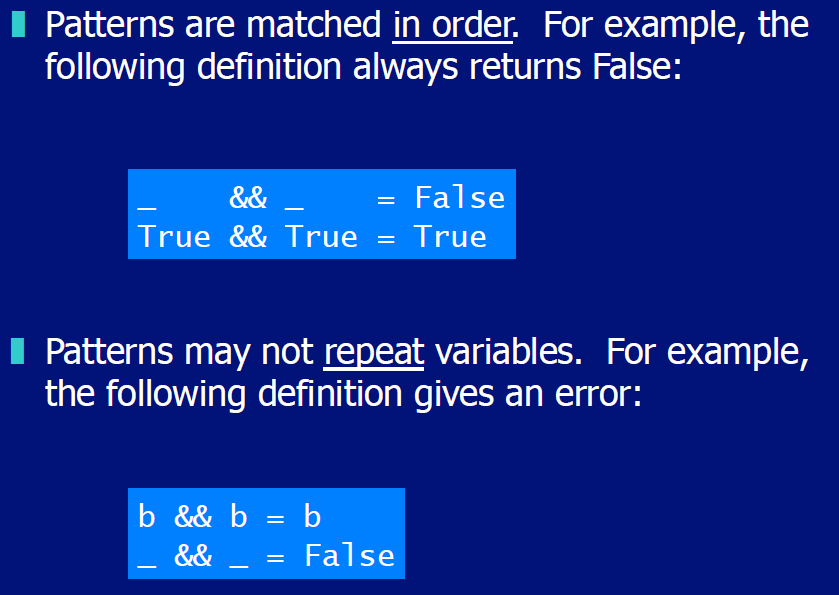


## Pattern Matching

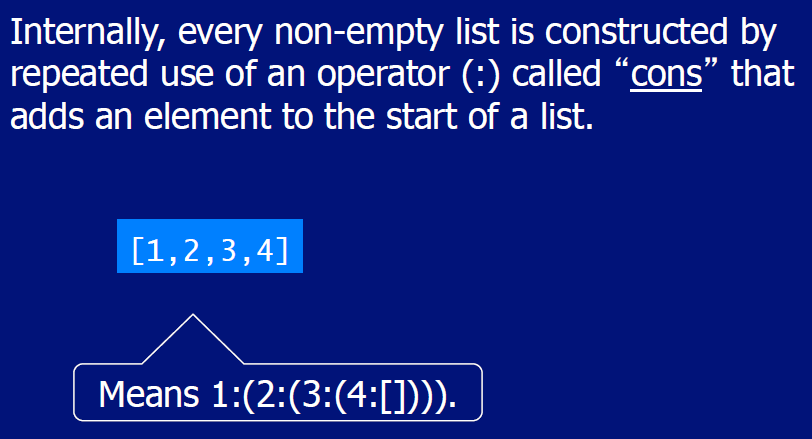


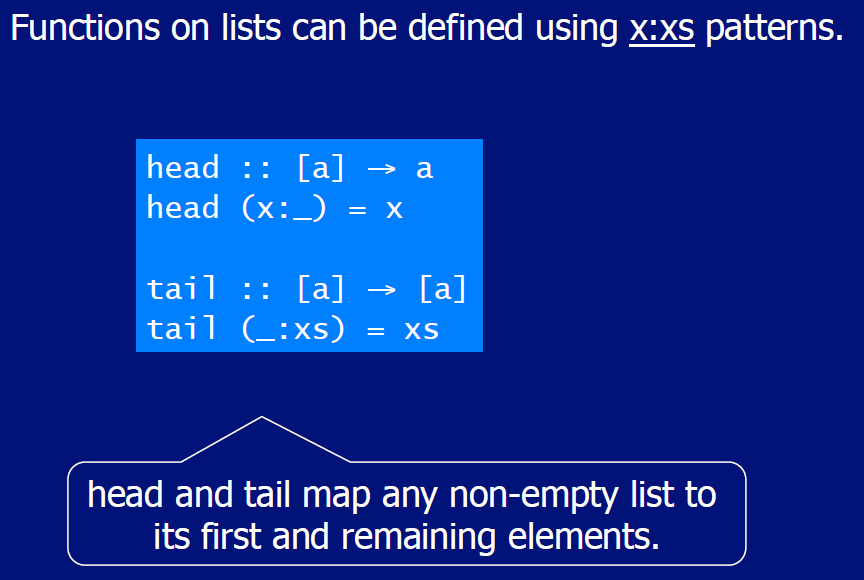


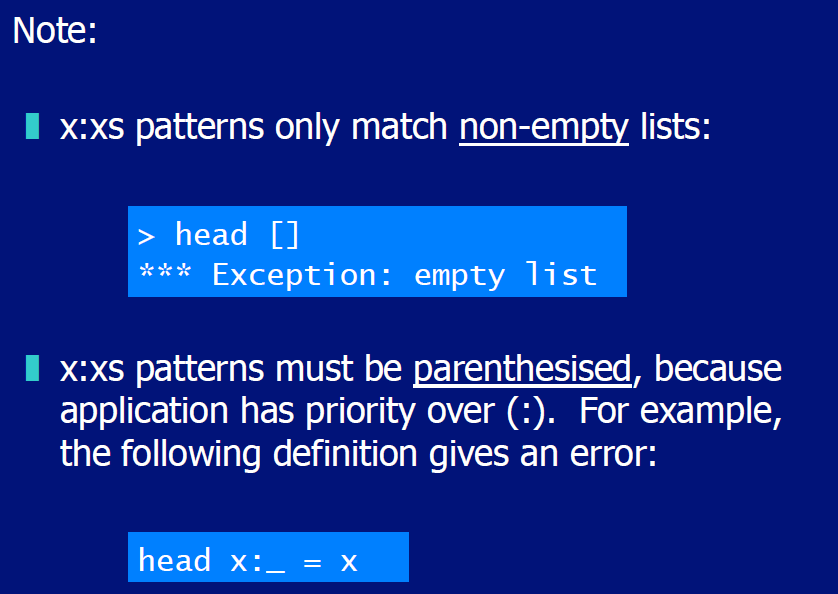




## List Patterns

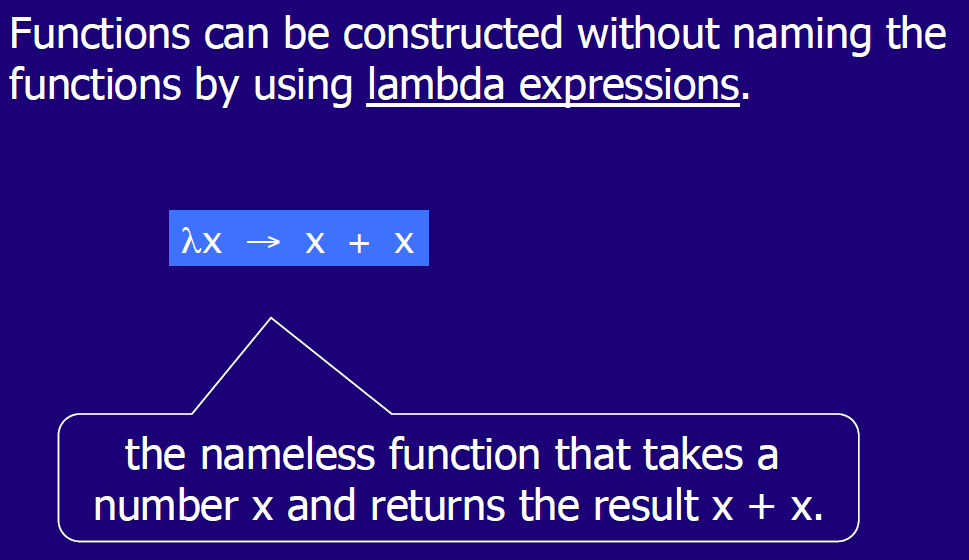


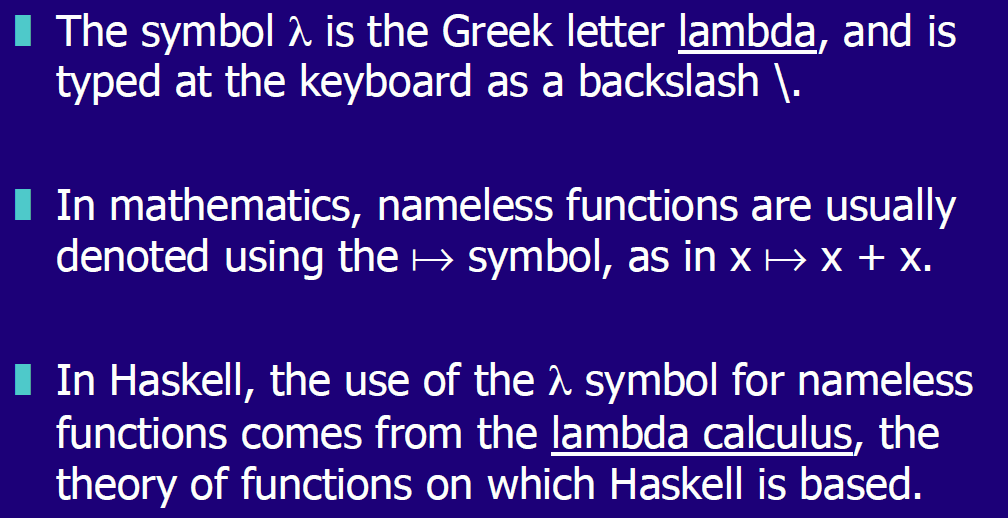




# Woche 6

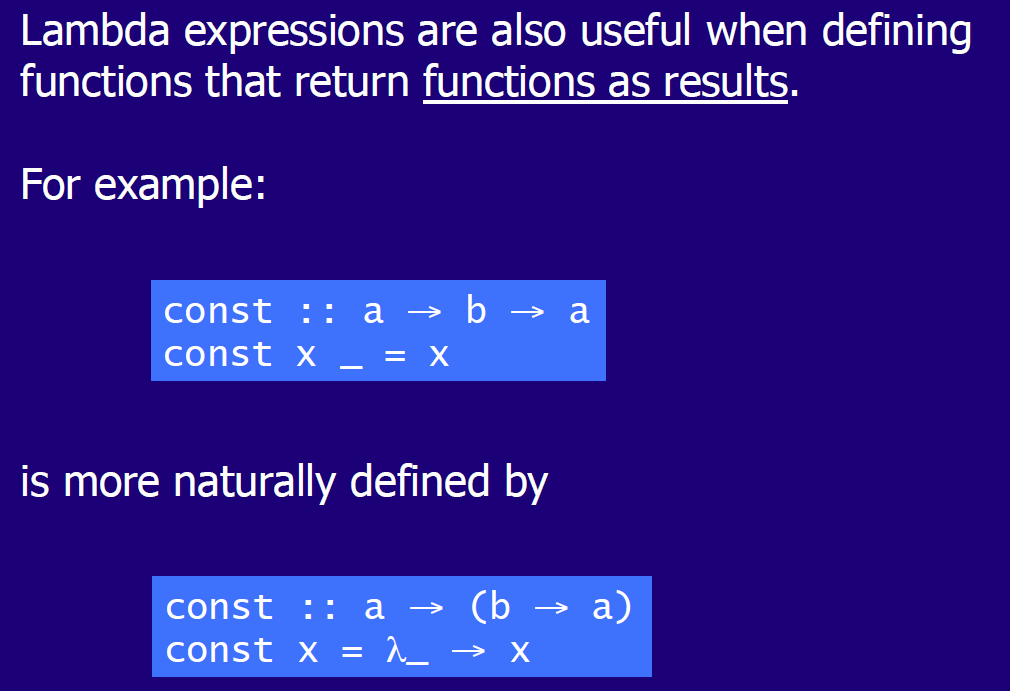
## Lambda Expressions

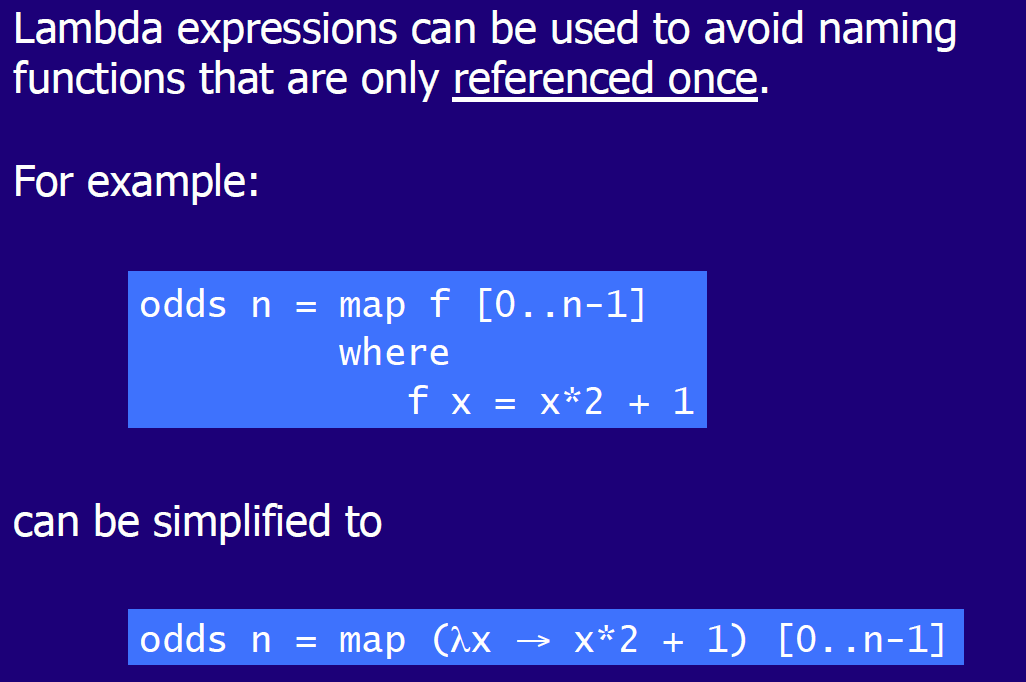




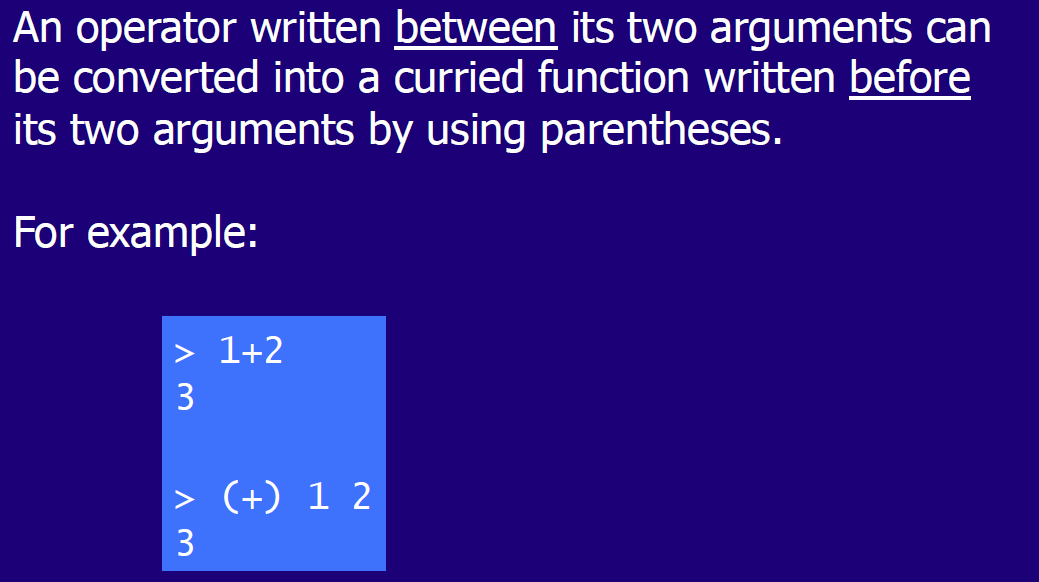
## Why are Lambdas useful

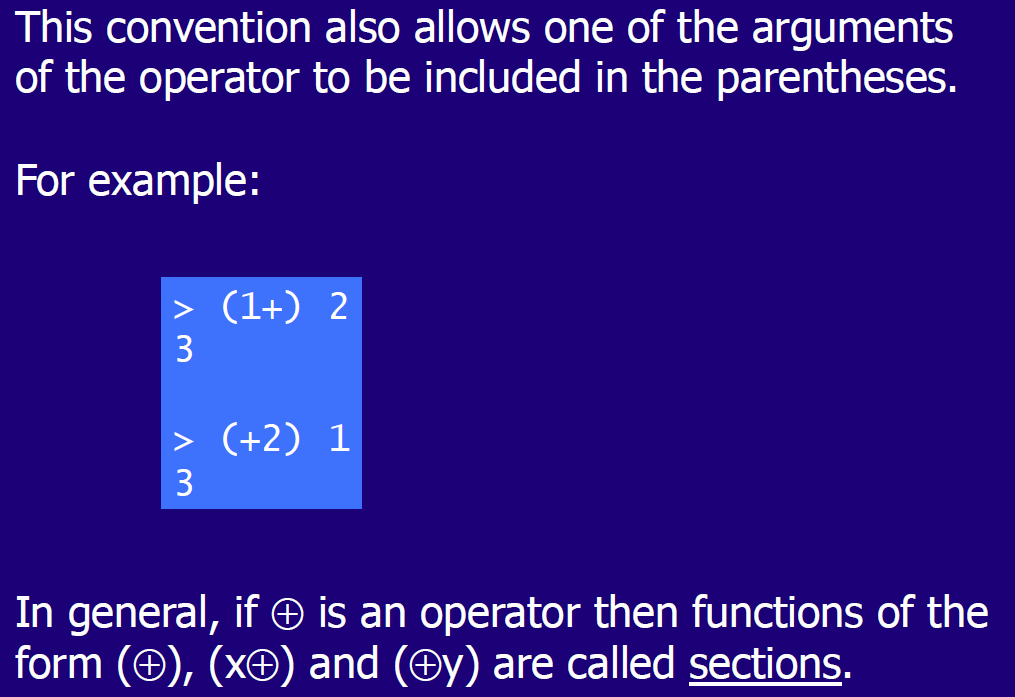




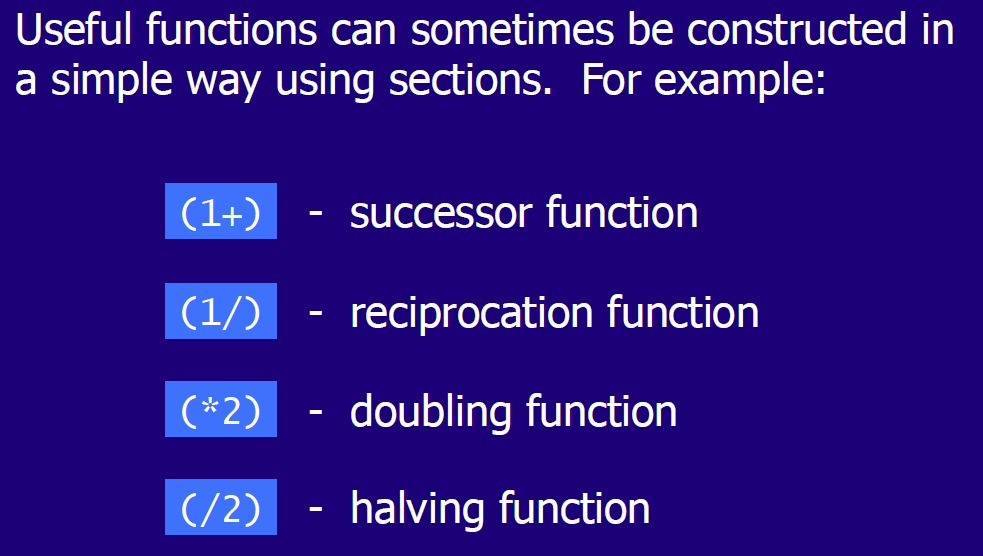


## Operator Sections

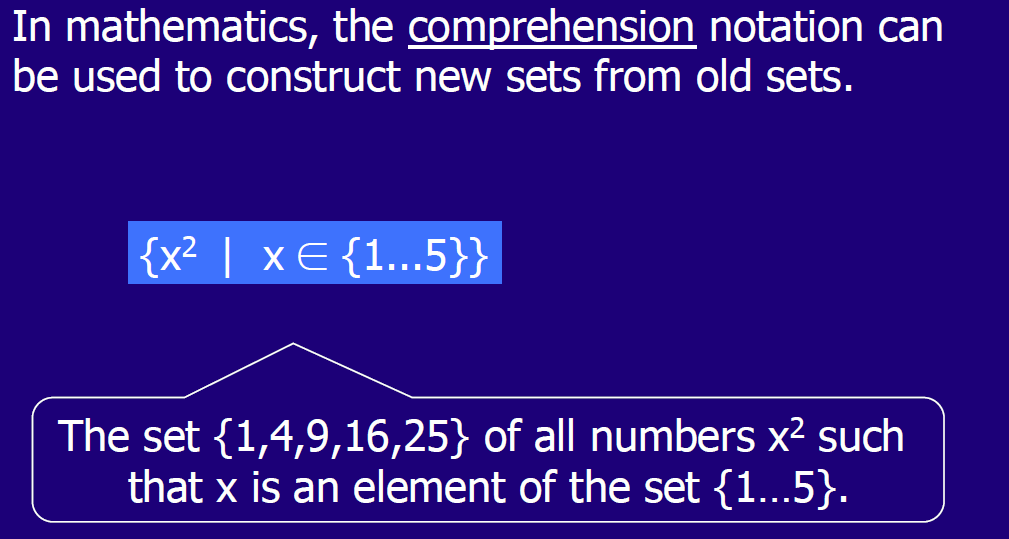




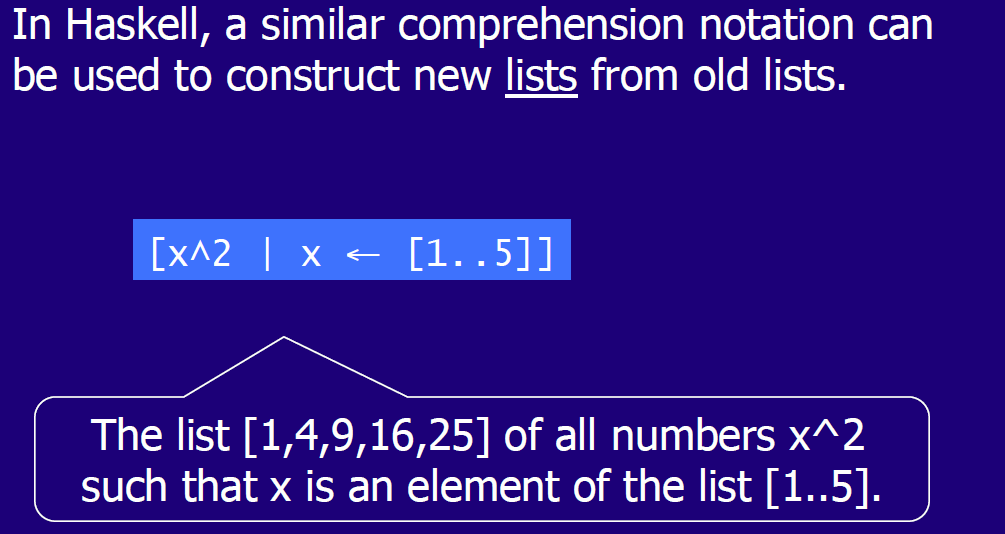
## Why are Sections useful

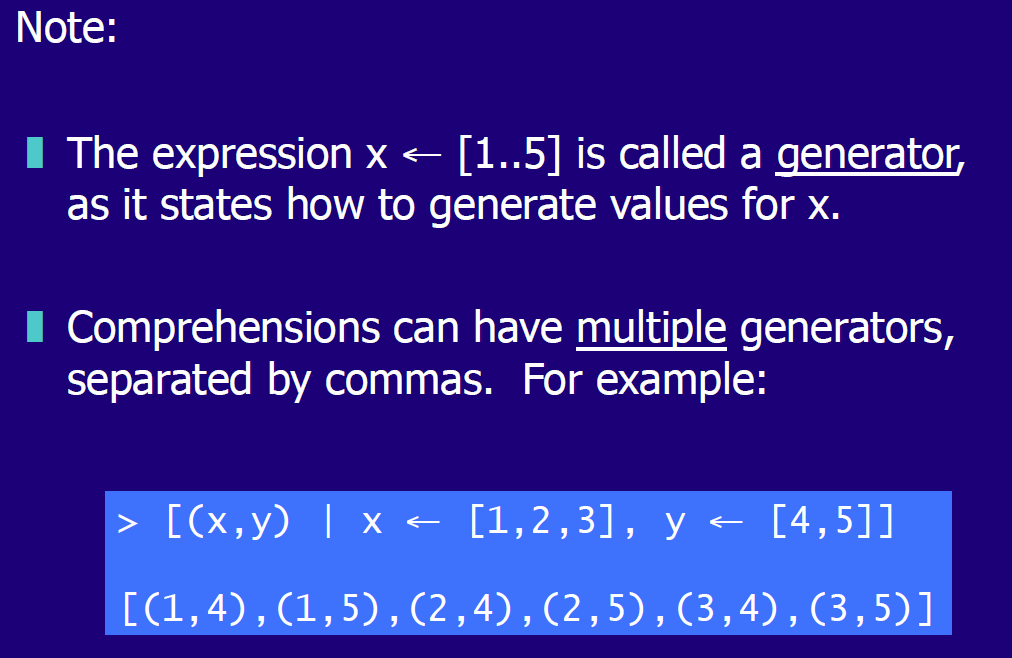


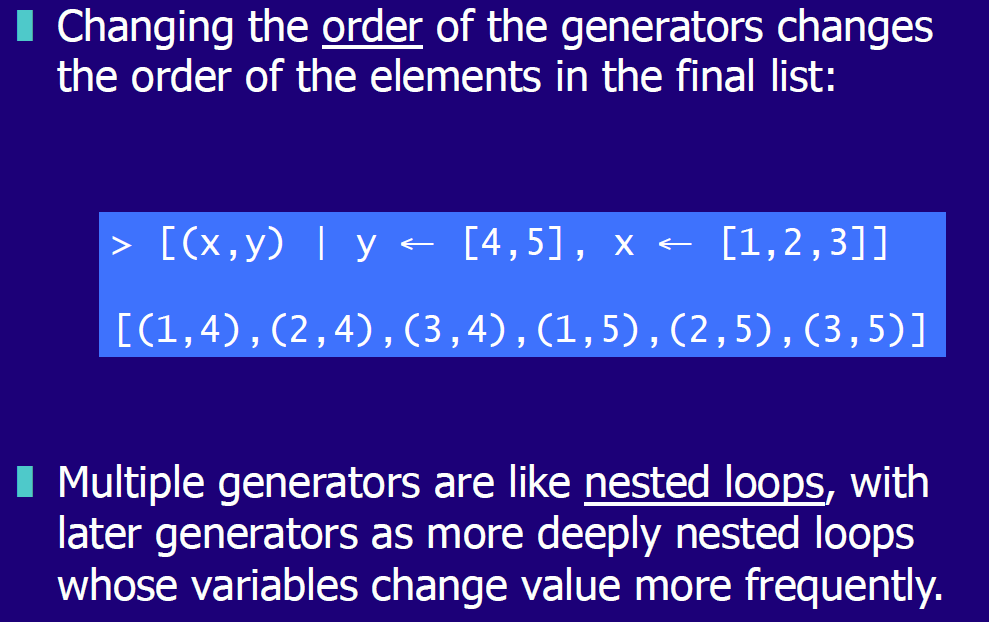
## Set Comprehensions

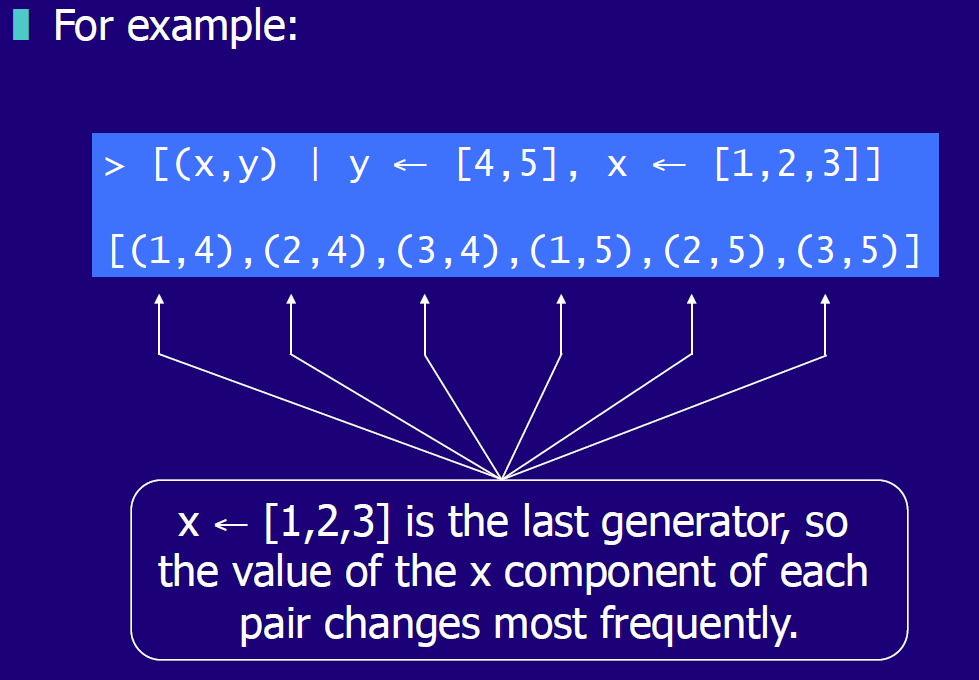


## List Comprehensions

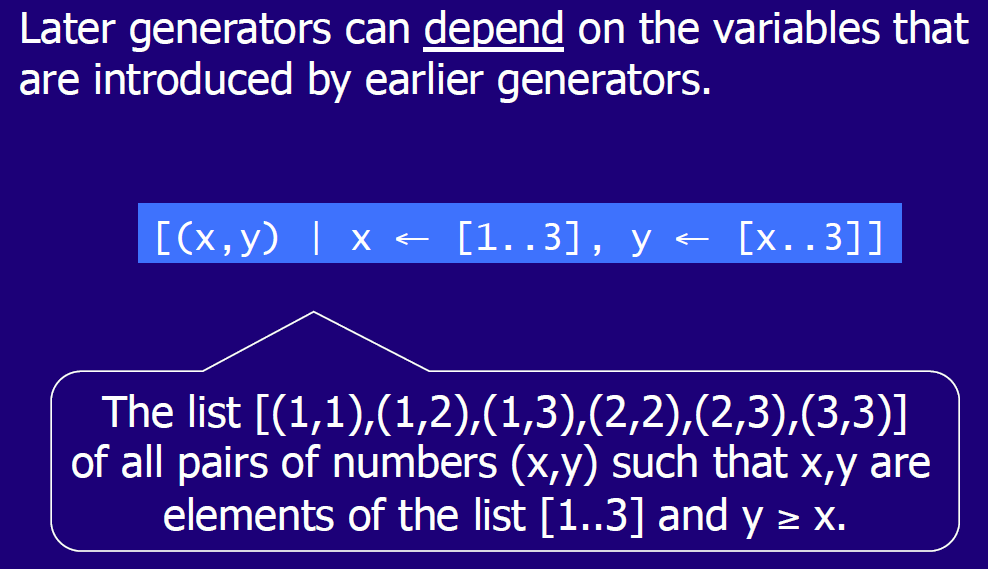


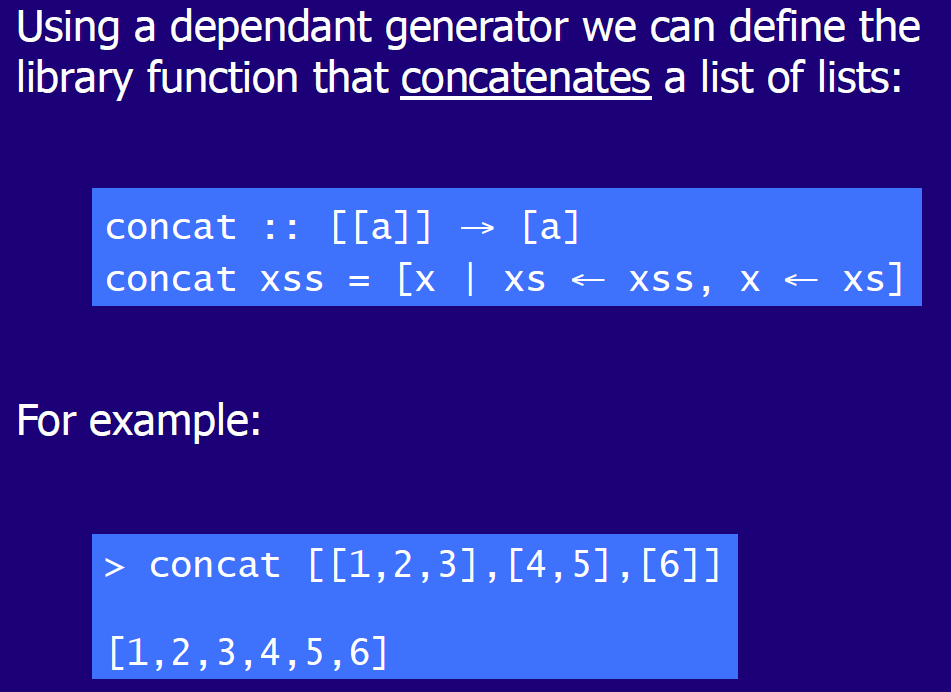




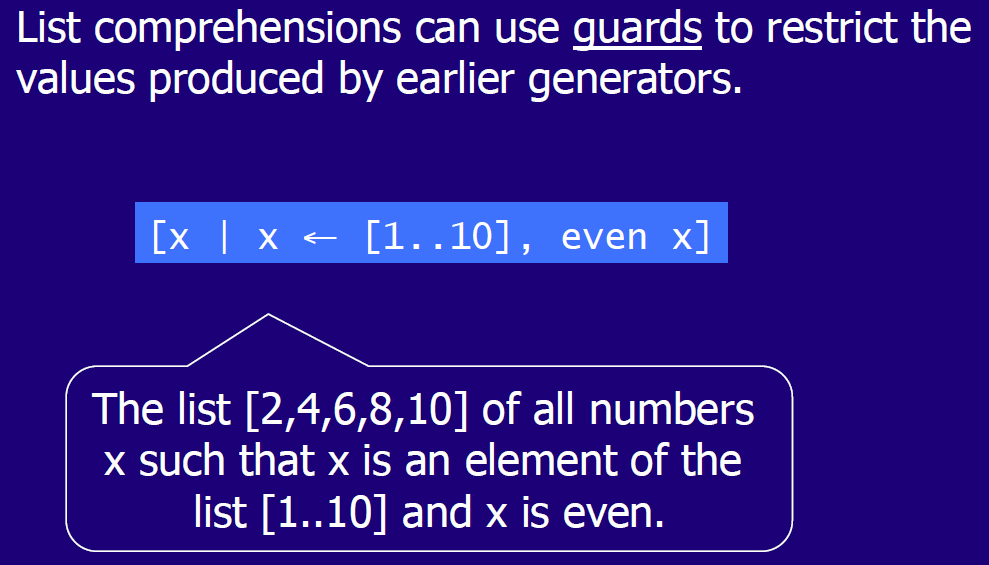


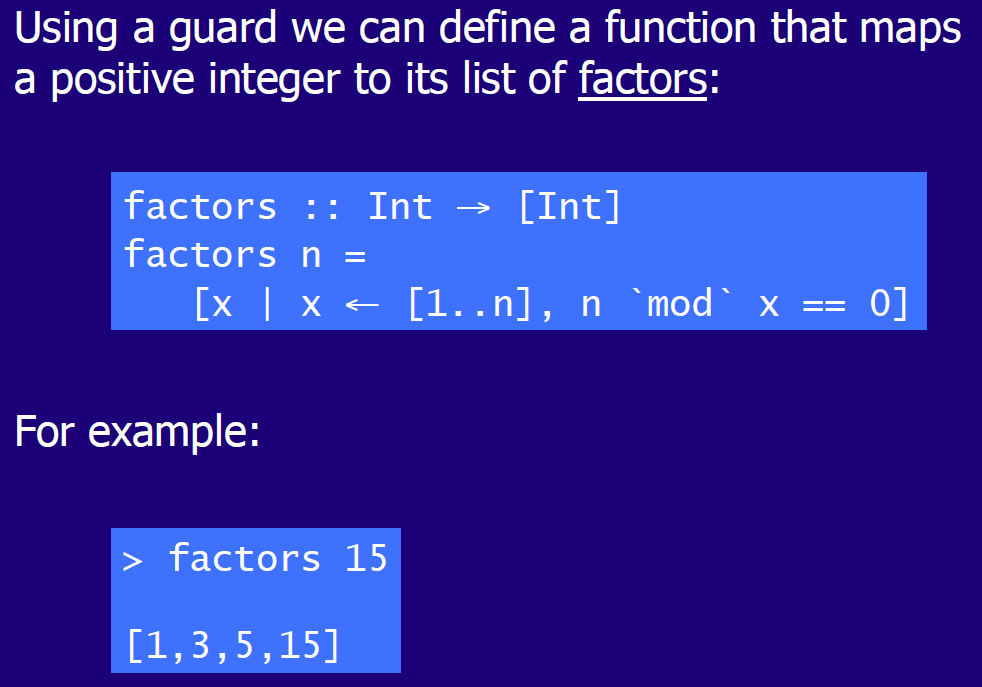
## Dependent Generators

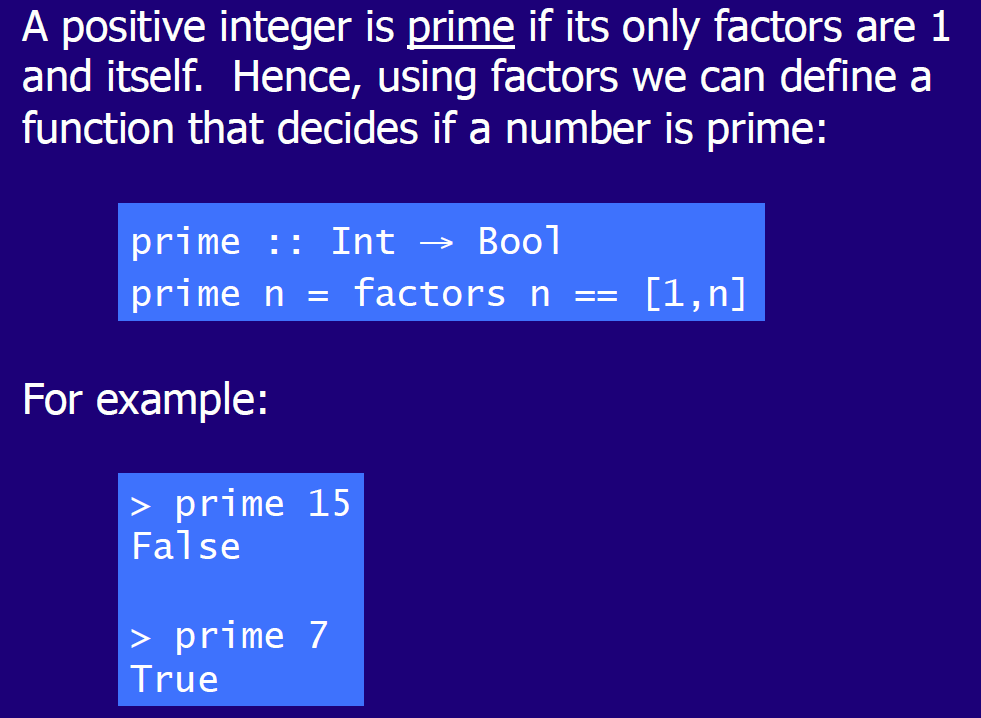


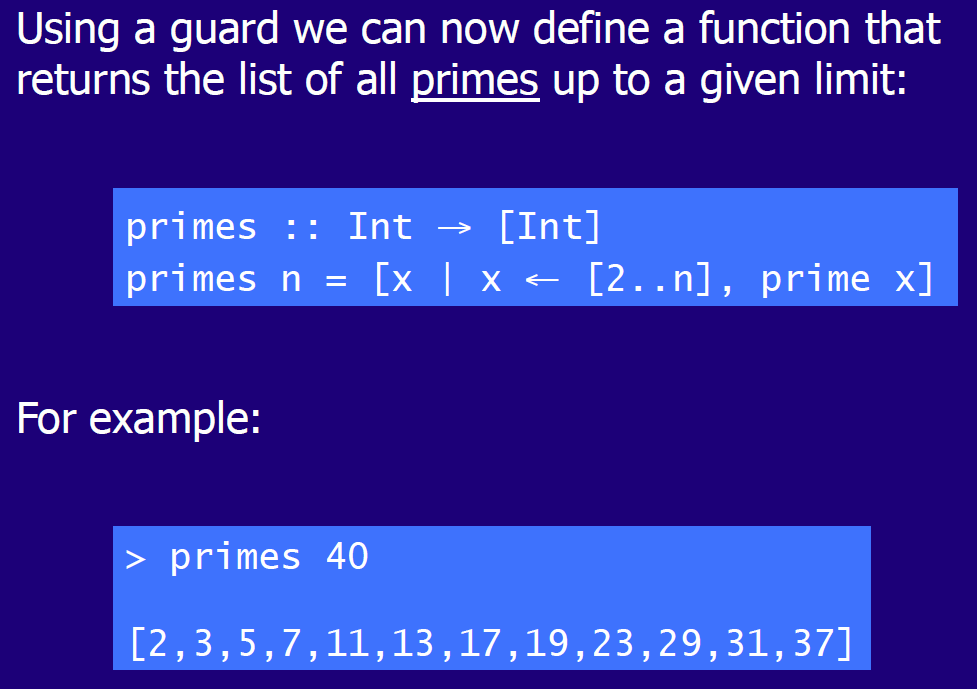


## Guards



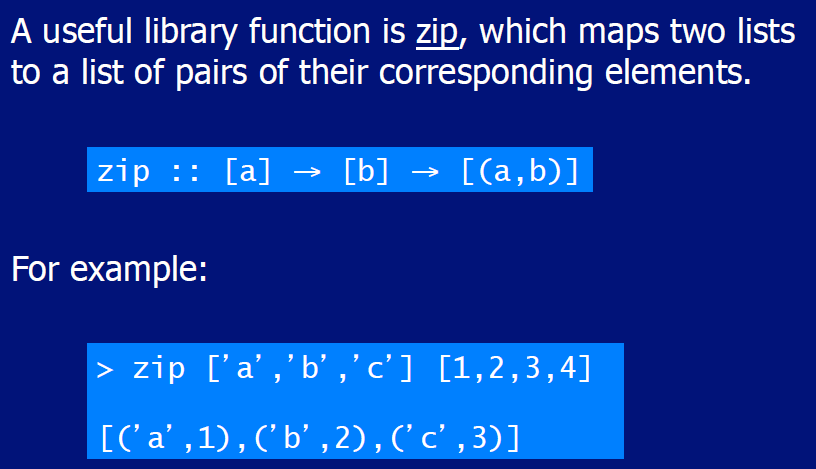


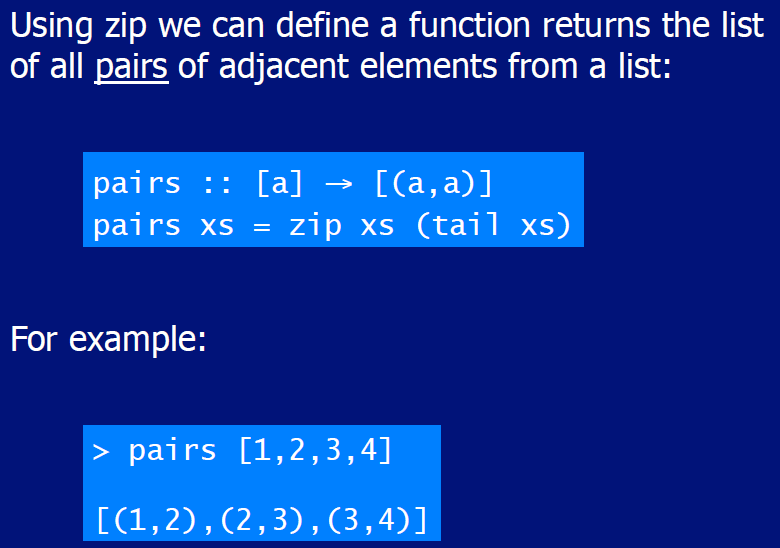


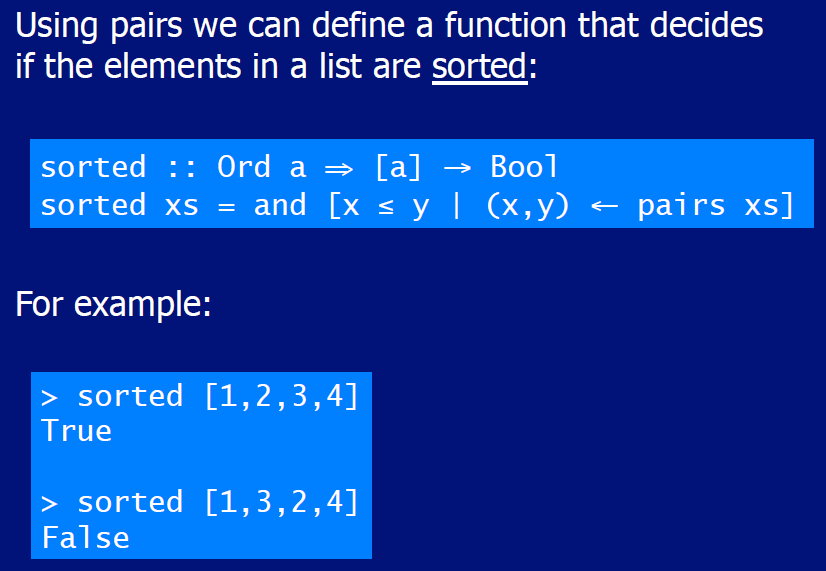


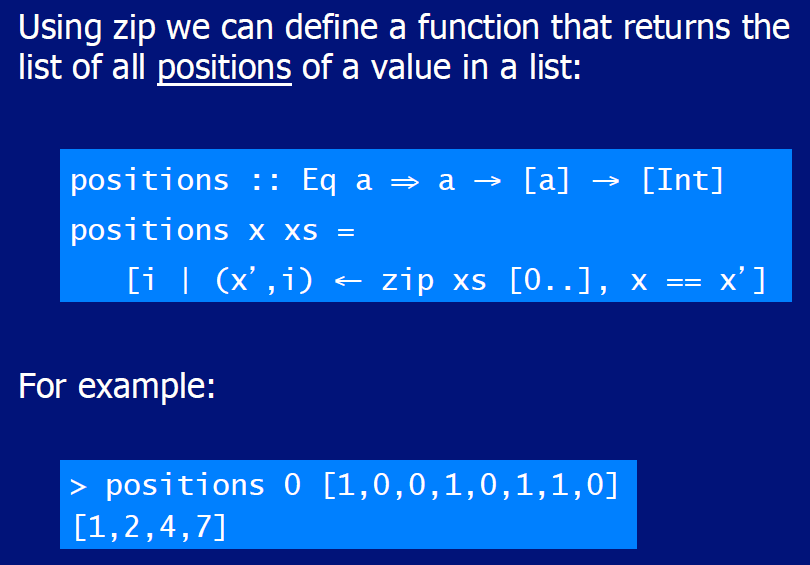
# Woche 7

## Zip Function

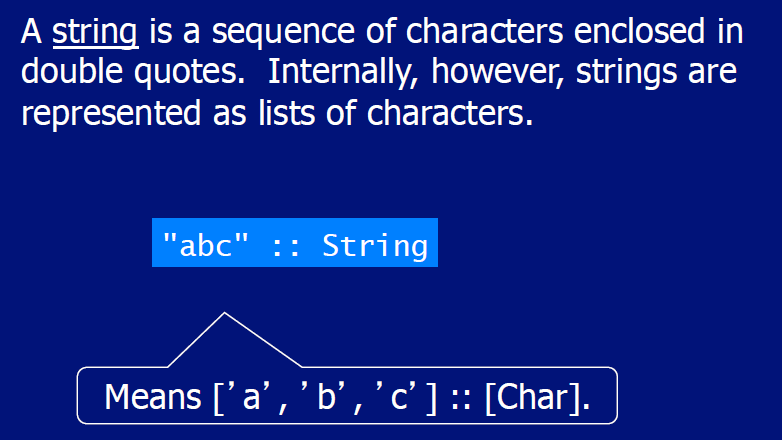


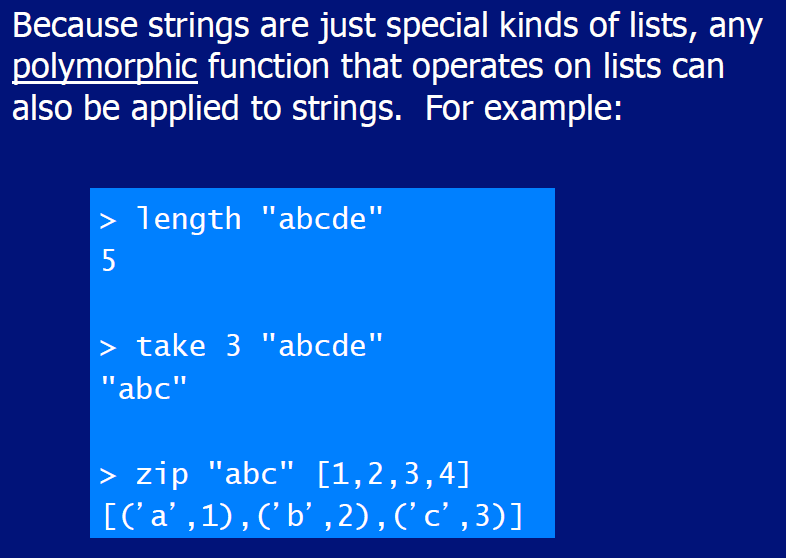


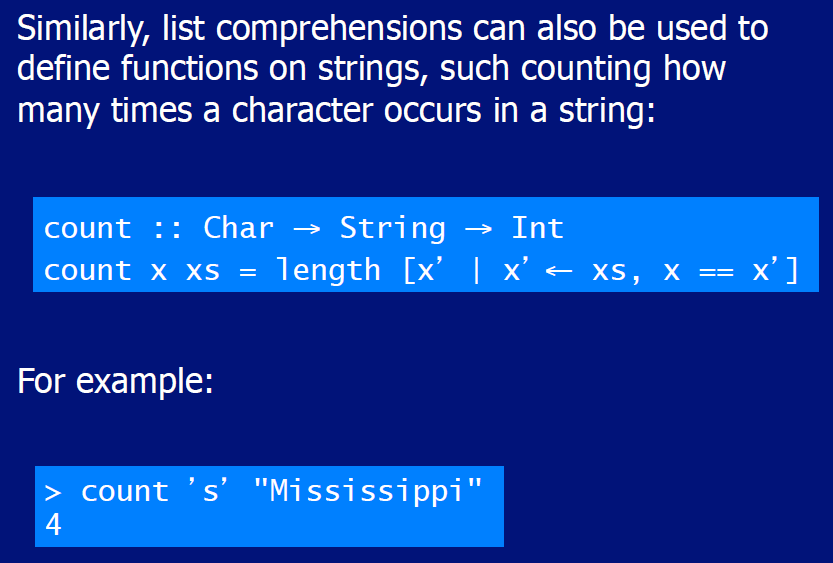




## String Comparison





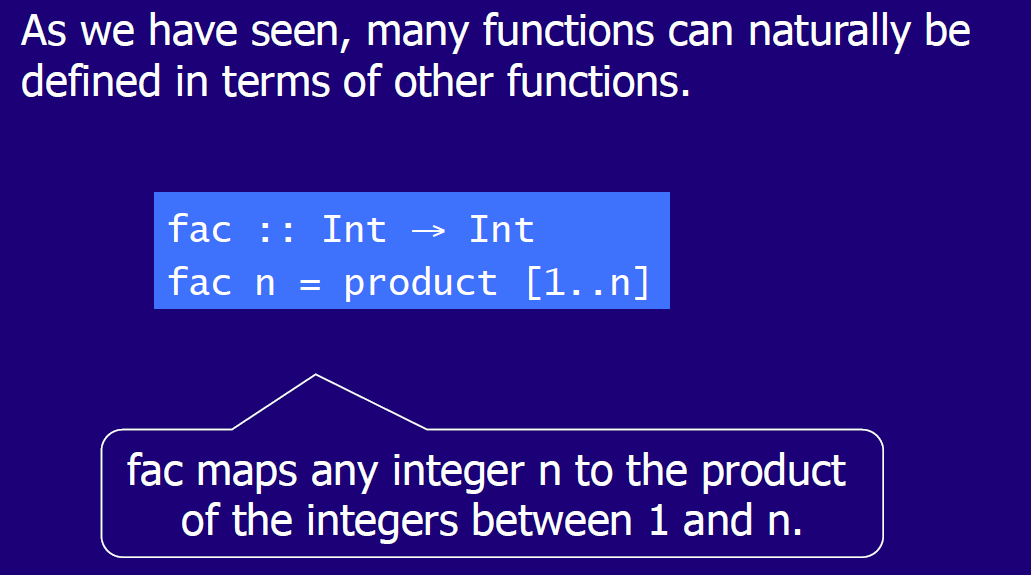


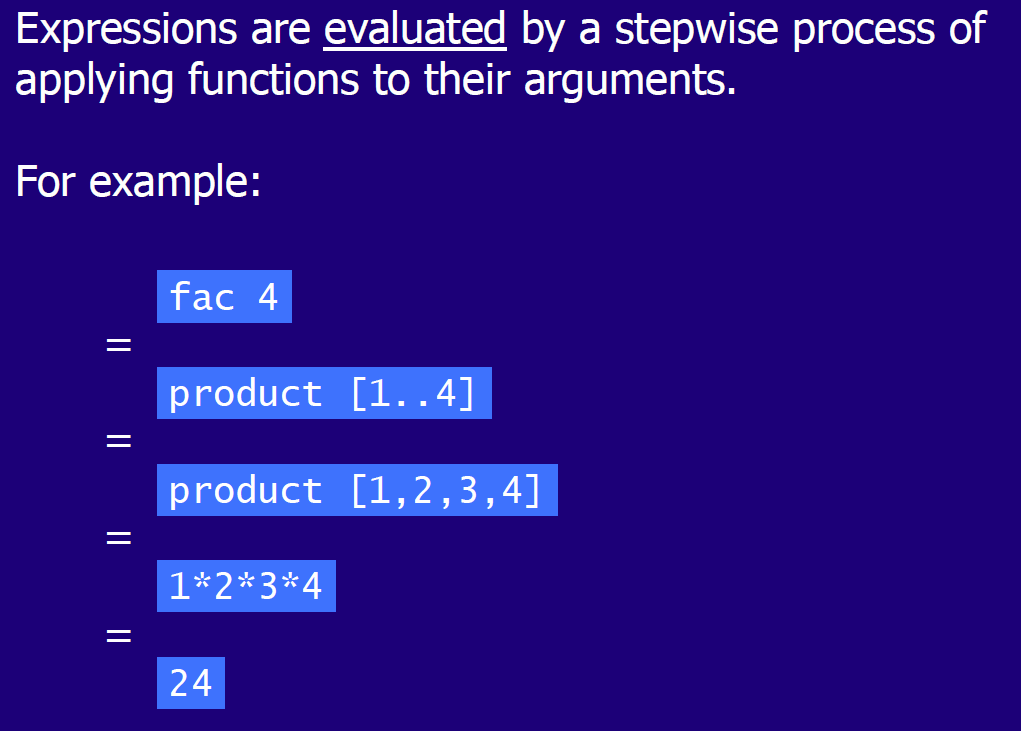
# Woche 8

In Woche 8 fand die erste Prüfung statt und somit fiel der Unterricht aus.

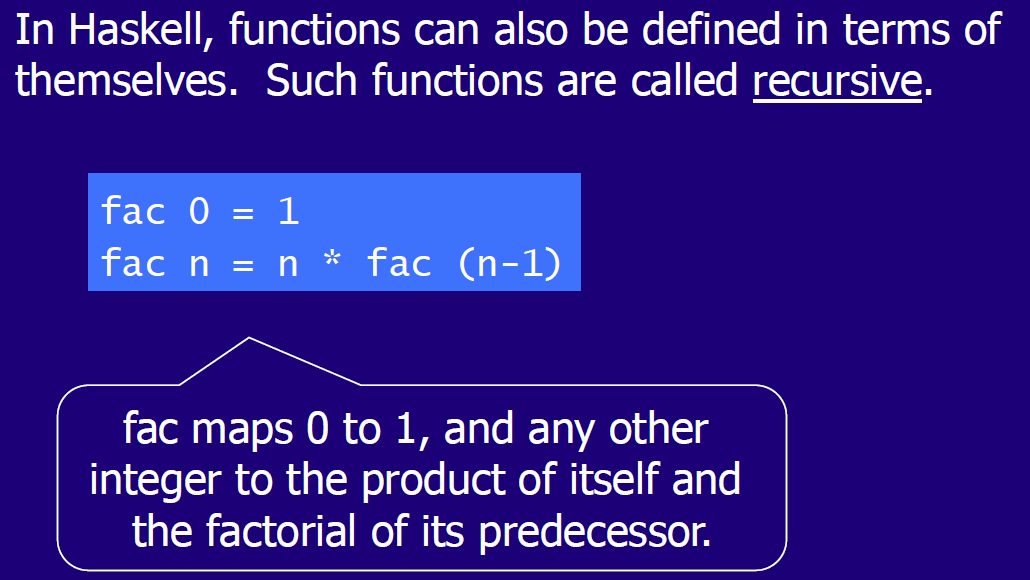
# Woche 9

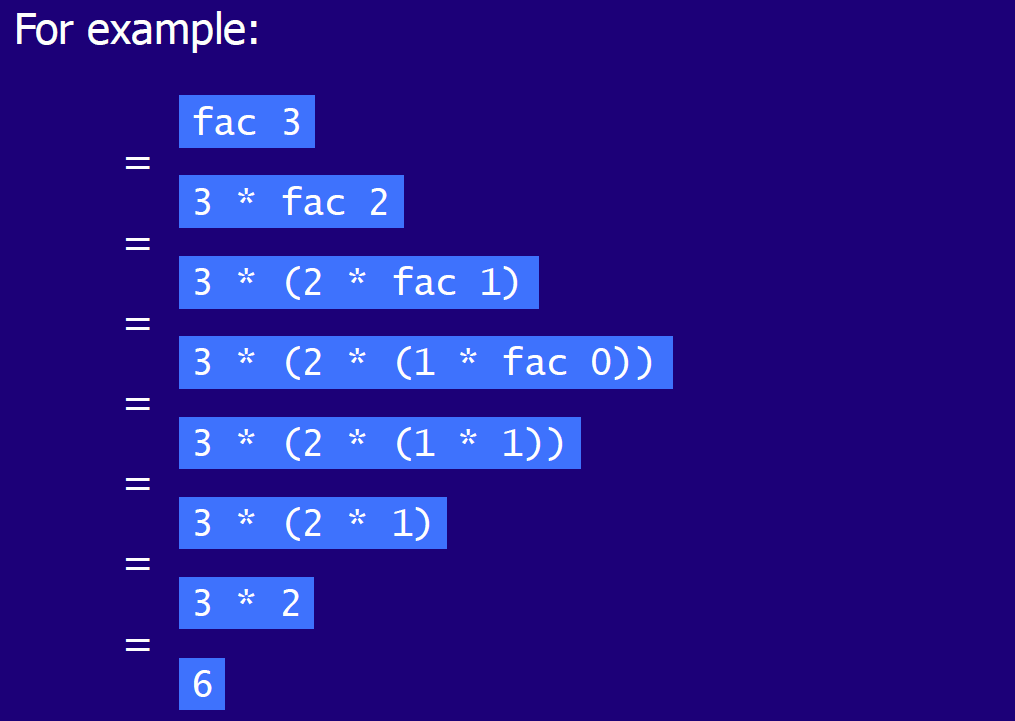
## Introduction to Recursion

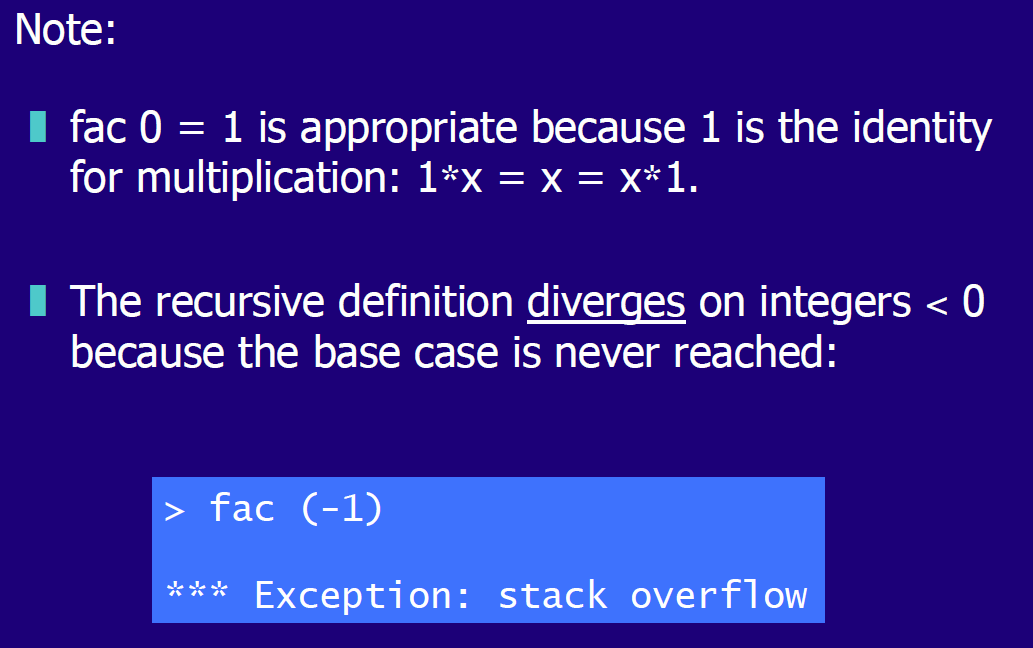




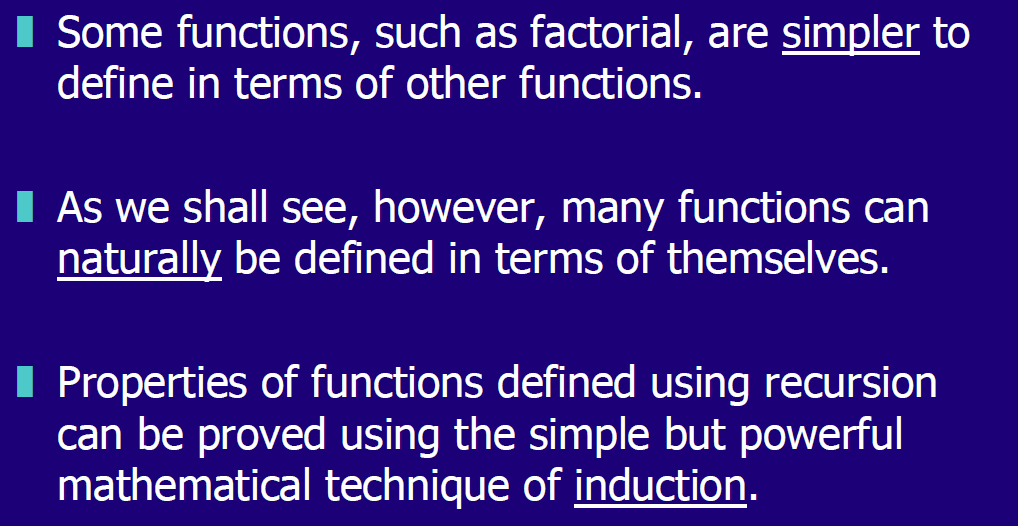
## Recursive Functions



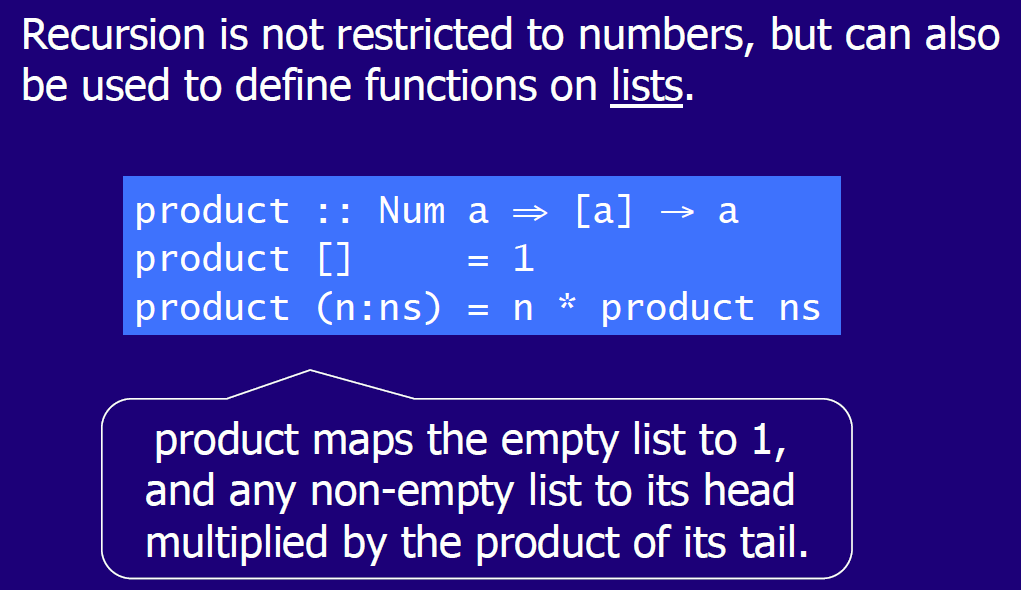


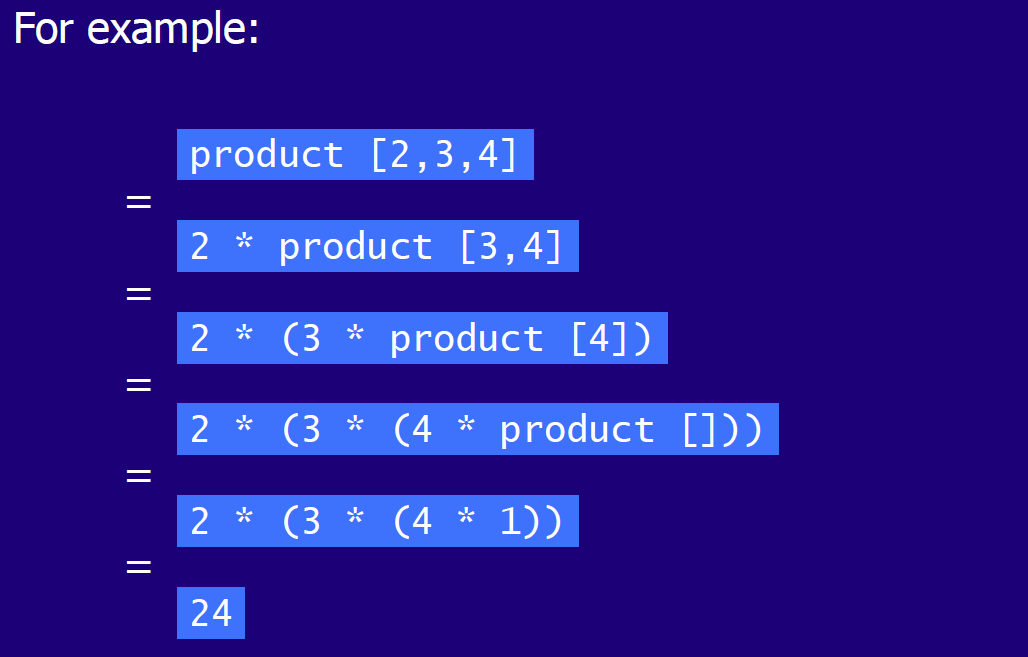


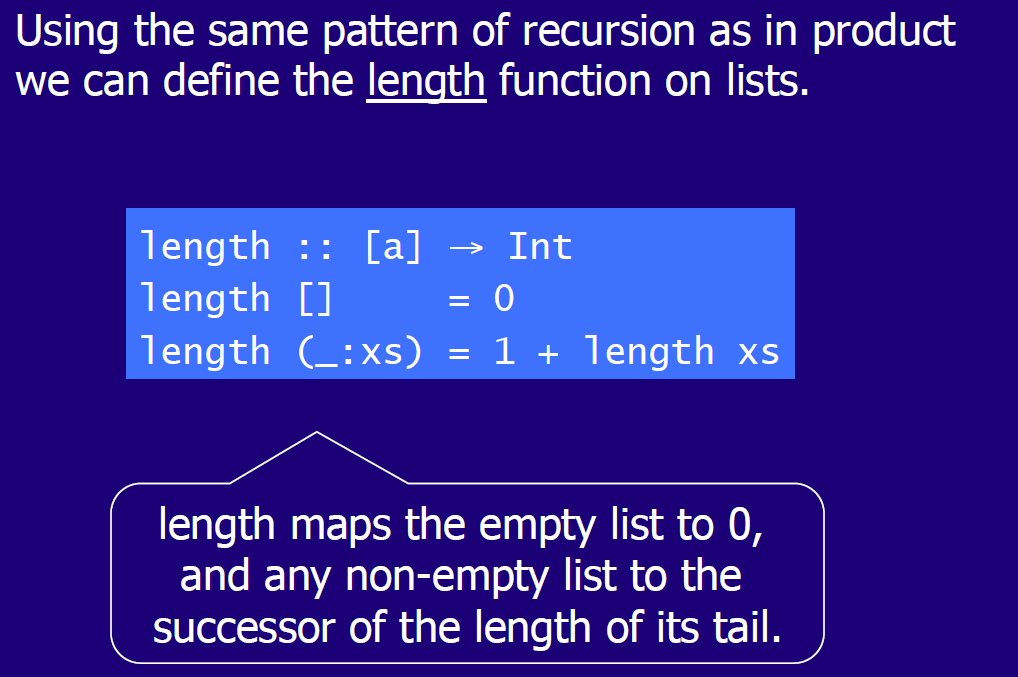
## Why is Recursion useful

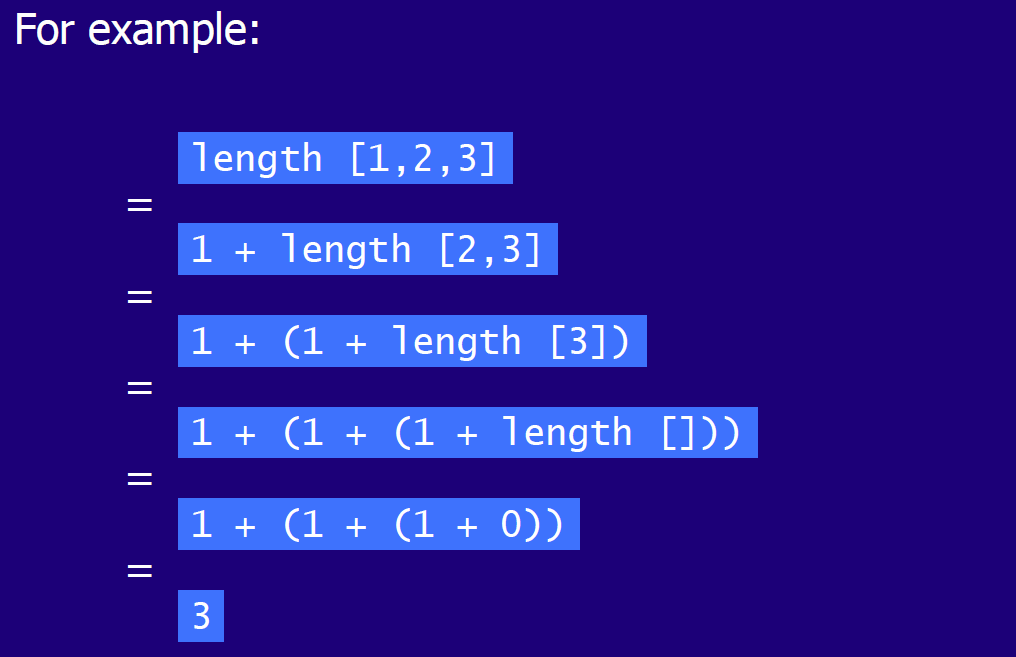


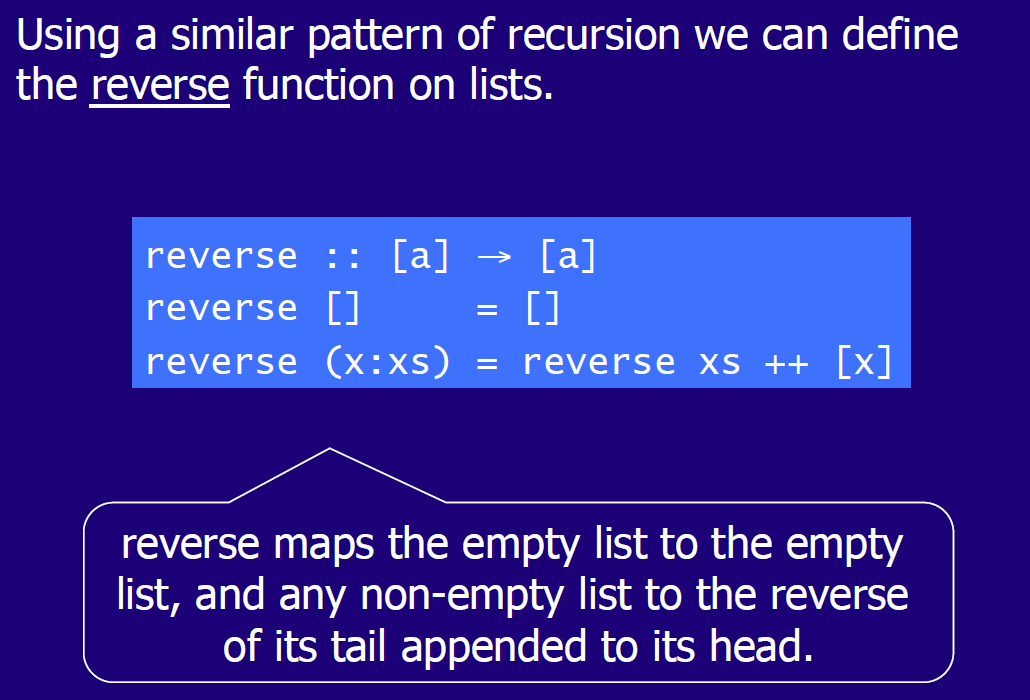
## Recursion on Lists

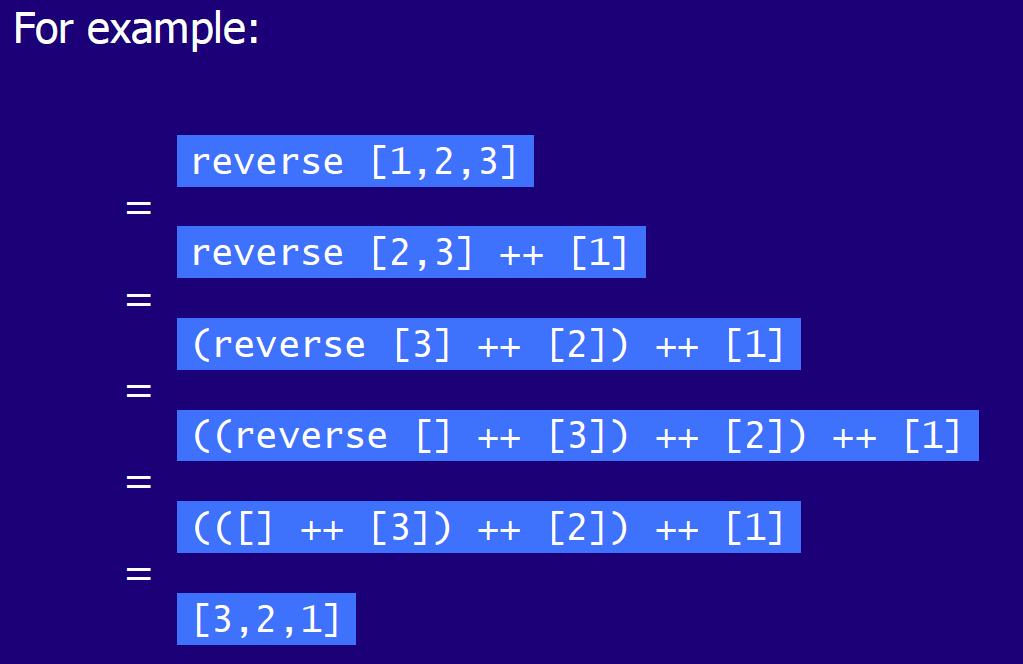




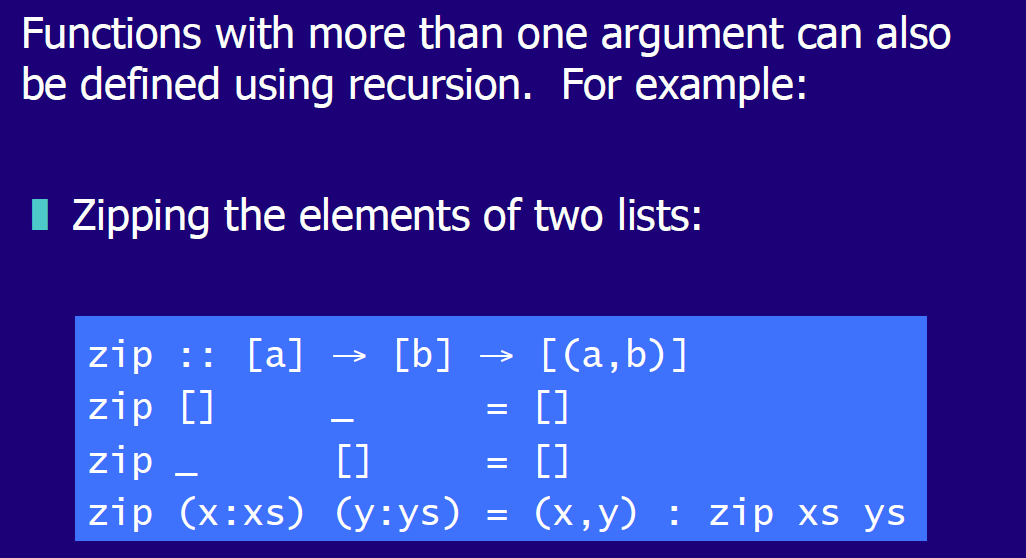


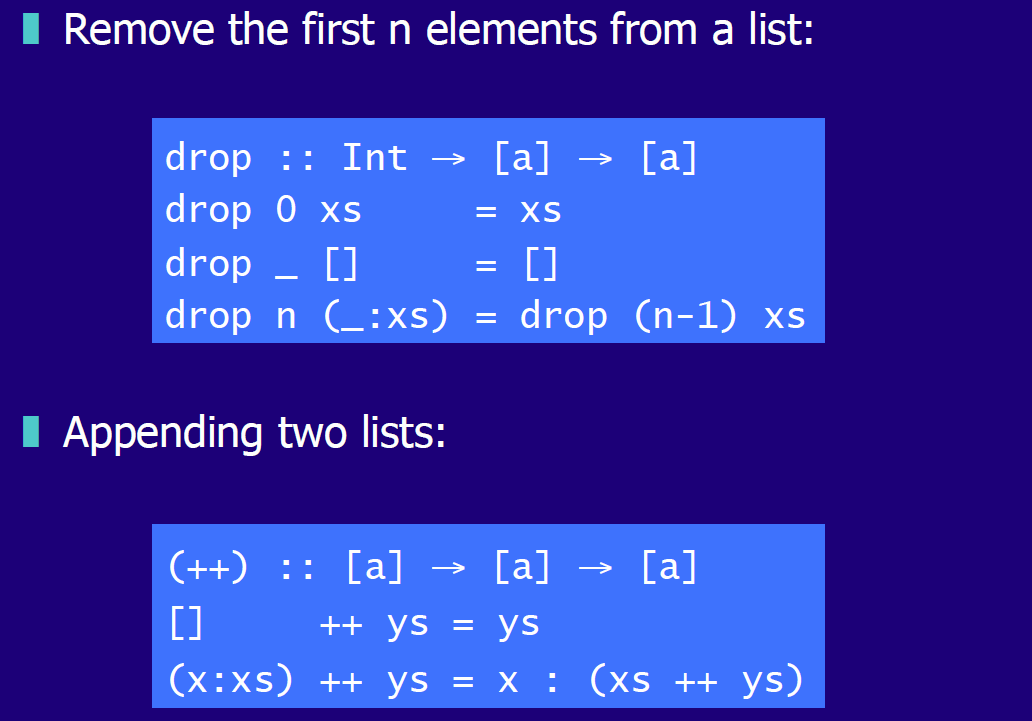




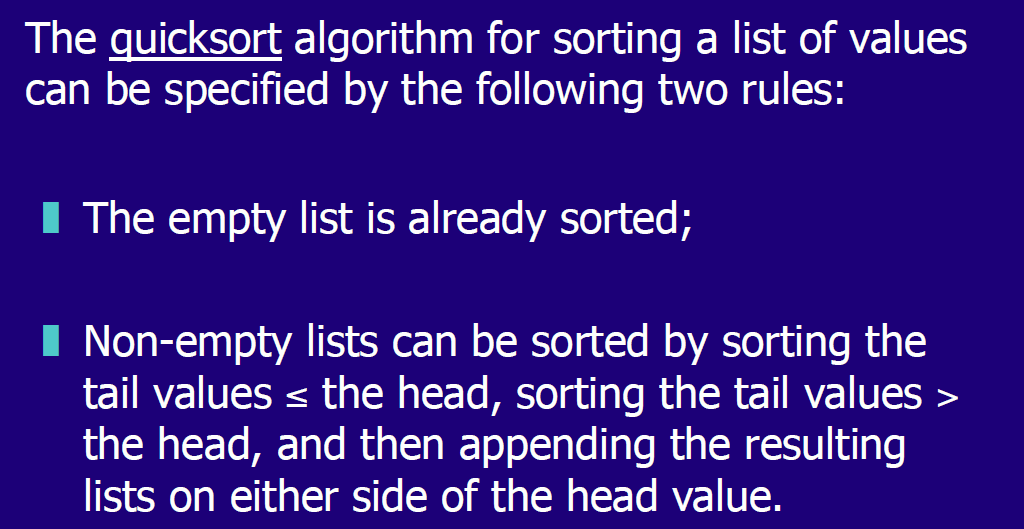


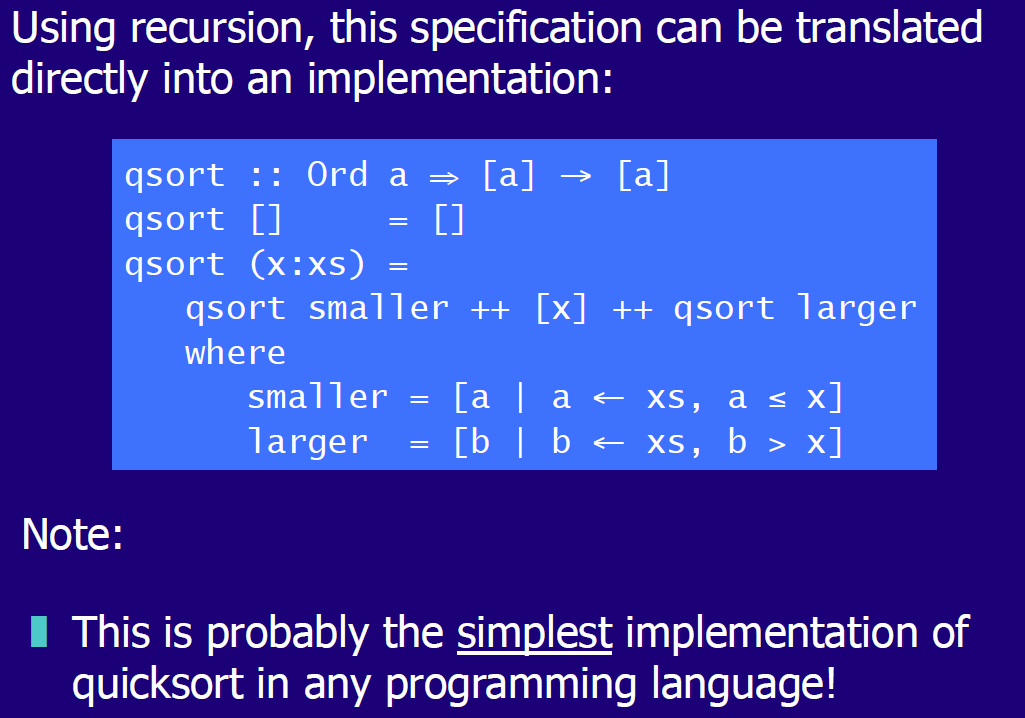
## Multiple Arguments

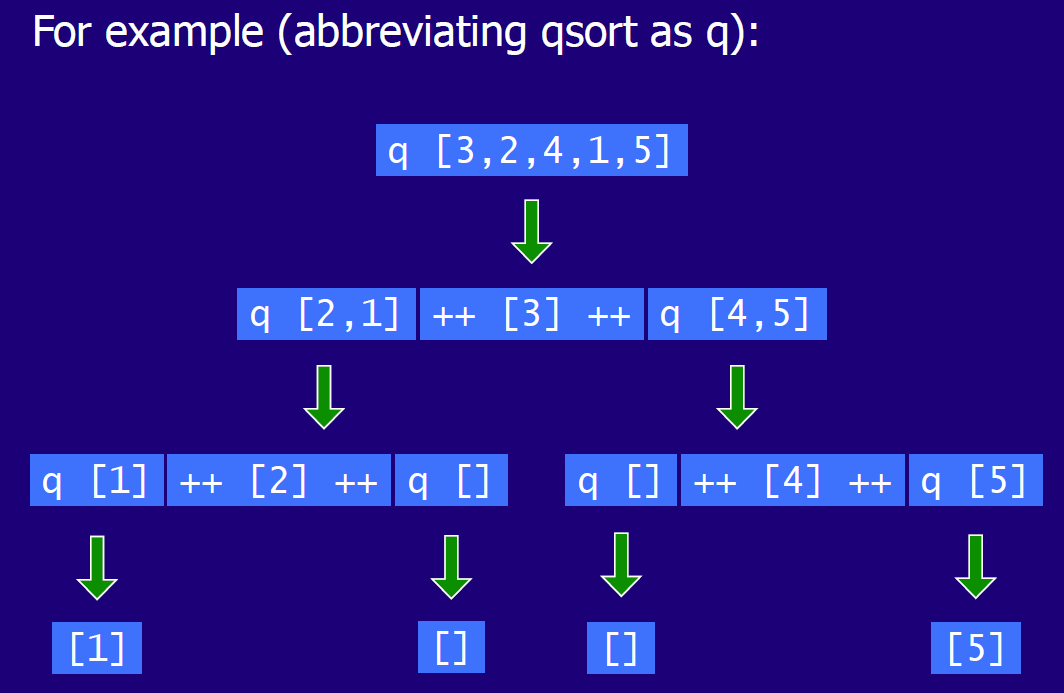




## Quicksort







# Woche 10