Moduldokumentation

Modul Functional Programming (fprog)

Simon Wächter

2016

Inhalt

[1 Einleitung 4](#_Toc472846914)

[1.1 Einleitung 4](#_Toc472846915)

[1.2 Lernziele 4](#_Toc472846916)

[1.3 Prüfungen 4](#_Toc472846917)

[2 Woche 1 5](#_Toc472846918)

[2.1 What is a Functional Language 5](#_Toc472846919)

[2.2 Example 5](#_Toc472846920)

[2.3 History 5](#_Toc472846921)

[2.4 Glasgow Haskell Compiler 5](#_Toc472846922)

[2.5 Prelude 5](#_Toc472846923)

[2.6 Examples 6](#_Toc472846924)

[2.7 Function Application 6](#_Toc472846925)

[2.8 Examples 7](#_Toc472846926)

[2.9 Haskell Scripts 7](#_Toc472846927)

[2.10 GHCI Commands 7](#_Toc472846928)

[2.11 Name Requirements 8](#_Toc472846929)

[2.12 Layout Rule 8](#_Toc472846930)

[3 Woche 2 9](#_Toc472846931)

[3.1 What is a Function? 9](#_Toc472846932)

[3.2 What is a Partial Function? 11](#_Toc472846933)

[3.3 Domain of Definition 11](#_Toc472846934)

[3.4 Confusion 11](#_Toc472846935)

[3.5 Additional types 12](#_Toc472846936)

[3.6 Summary 12](#_Toc472846937)

[4 Woche 3 13](#_Toc472846938)

[4.1 What is a Type 13](#_Toc472846939)

[4.2 Type Errors 13](#_Toc472846940)

[4.3 Types in Haskell 13](#_Toc472846941)

[4.4 Basic Types 14](#_Toc472846942)

[4.5 List Types 14](#_Toc472846943)

[4.6 Tuple Types 15](#_Toc472846944)

[4.7 Function Types 16](#_Toc472846945)

[4.8 Curried Functions 16](#_Toc472846946)

[4.9 Why is Currying useful 17](#_Toc472846947)

[4.10 Currying Conventions 18](#_Toc472846948)

[4.11 Polymorphic Functions 18](#_Toc472846949)

[4.12 Overloaded Functions 19](#_Toc472846950)

[4.13 Hints and Tips 20](#_Toc472846951)

[5 Woche 4 21](#_Toc472846952)

[5.1 Typeclasses 21](#_Toc472846953)

[6 Woche 5 22](#_Toc472846954)

[6.1 Conditional Expressions 22](#_Toc472846955)

[6.2 Guarded Equations 22](#_Toc472846956)

[6.3 Pattern Matching 23](#_Toc472846957)

[6.4 List Patterns 24](#_Toc472846958)

[7 Woche 6 26](#_Toc472846959)

[7.1 Lambda Expressions 26](#_Toc472846960)

[7.2 Why are Lambdas useful 26](#_Toc472846961)

[7.3 Operator Sections 28](#_Toc472846962)

[7.4 Why are Sections useful 28](#_Toc472846963)

[7.5 Set Comprehensions 29](#_Toc472846964)

[7.6 List Comprehensions 29](#_Toc472846965)

[7.7 Dependent Generators 30](#_Toc472846966)

[7.8 Guards 31](#_Toc472846967)

[8 Woche 7 33](#_Toc472846968)

[8.1 Zip Function 33](#_Toc472846969)

[8.2 String Comparison 34](#_Toc472846970)

[9 Woche 8 36](#_Toc472846971)

[10 Woche 9 & 10 37](#_Toc472846972)

[10.1 Introduction to Recursion 37](#_Toc472846973)

[10.2 Recursive Functions 37](#_Toc472846974)

[10.3 Why is Recursion useful 38](#_Toc472846975)

[10.4 Recursion on Lists 39](#_Toc472846976)

[10.5 Multiple Arguments 41](#_Toc472846977)

[10.6 Quicksort 41](#_Toc472846978)

[11 Woche 11 43](#_Toc472846979)

[12 Woche 12 & 13 & 14 44](#_Toc472846980)

[12.1 Introduction to Higher Order Functions 44](#_Toc472846981)

[12.2 Why are they useful 44](#_Toc472846982)

[12.3 Map Function 44](#_Toc472846983)

[12.4 Filter Function 45](#_Toc472846984)

[12.5 Foldr Function 46](#_Toc472846985)

[12.6 Other Foldr Examples 48](#_Toc472846986)

[12.7 Why is Foldr useful 49](#_Toc472846987)

[12.8 Other Library Functions 49](#_Toc472846988)

[13 Woche 15 52](#_Toc472846989)

[13.1 Type Declarations 52](#_Toc472846990)

[13.2 Data Declarations 53](#_Toc472846991)

[13.3 Recursive Types 55](#_Toc472846992)

[13.4 Arithmetic Expressions 57](#_Toc472846993)

[13.5 Binary Trees 59](#_Toc472846994)

[14 Woche 16 61](#_Toc472846995)

# 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

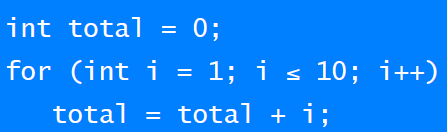
## What is a Functional Language

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;

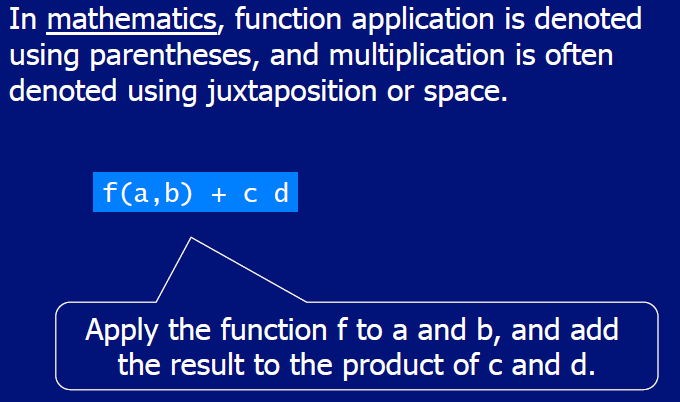
## Prelude

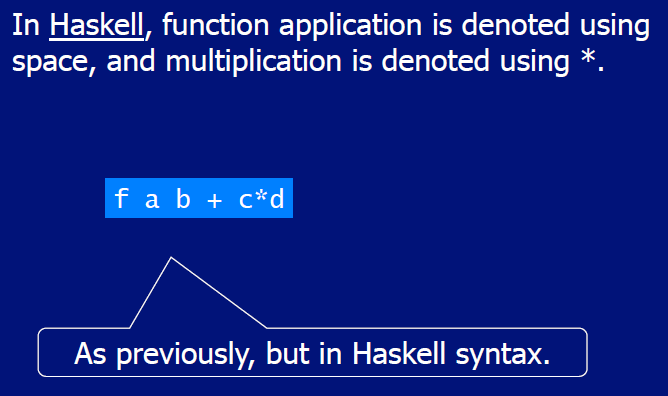
* Haskell comes with a large number of standard library functions. In addition to the familiar numeric functions such as + and \*, the library also provides many useful functions on lists.

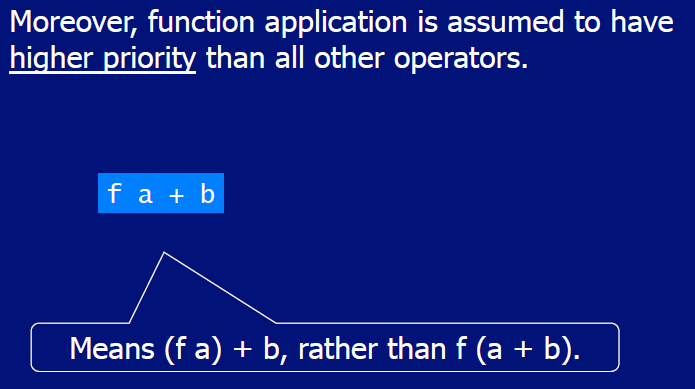
## Examples

* 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]

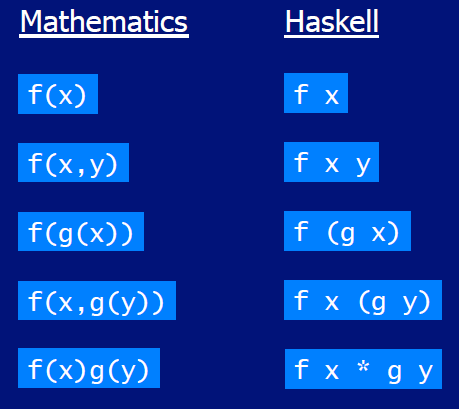
## Function Application



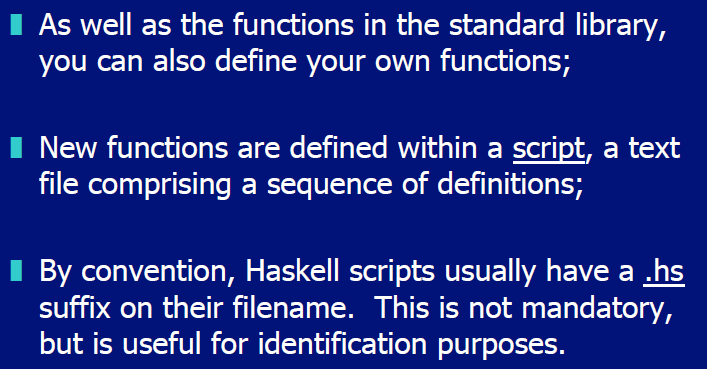




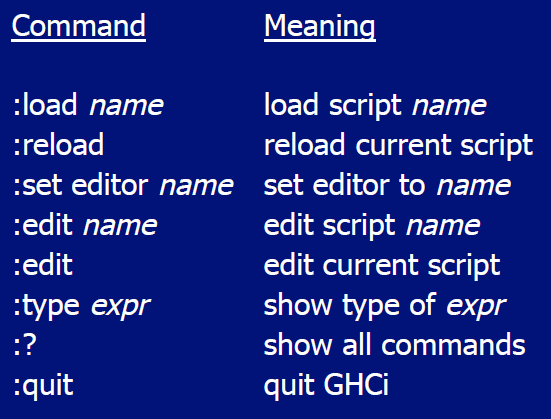
## Examples



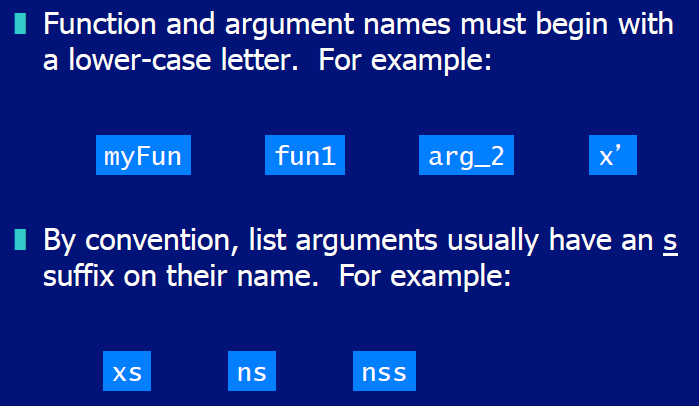
## Haskell Scripts



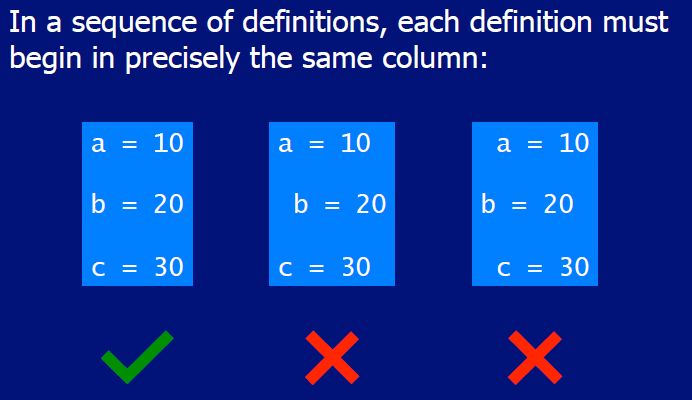
## GHCI Commands

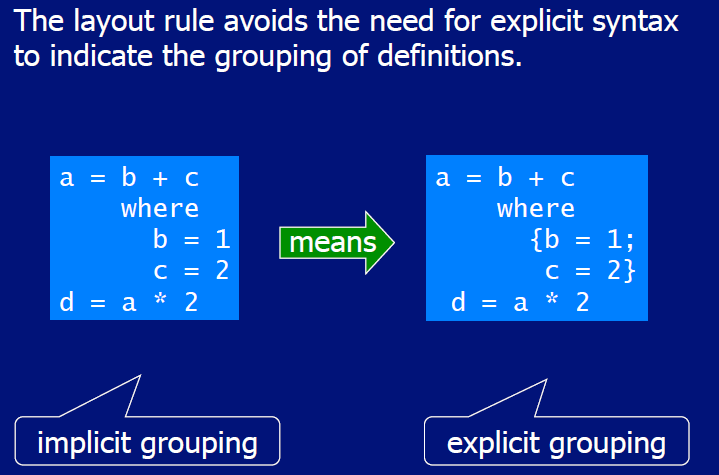


## Name Requirements



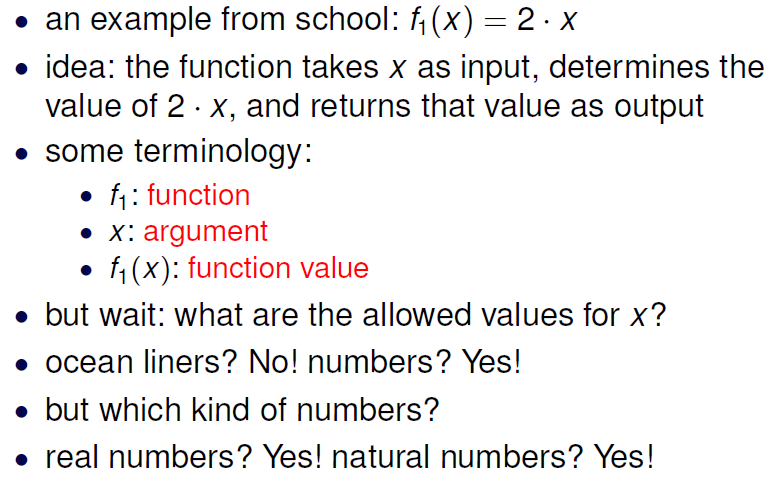
## Layout Rule

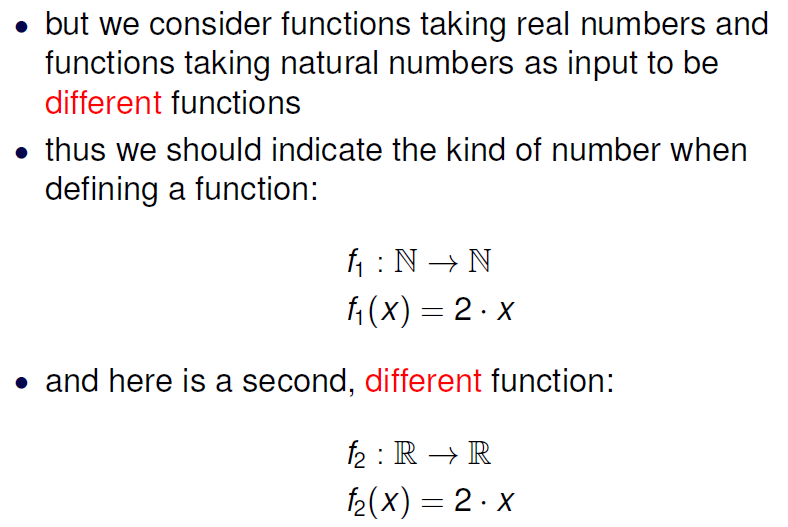


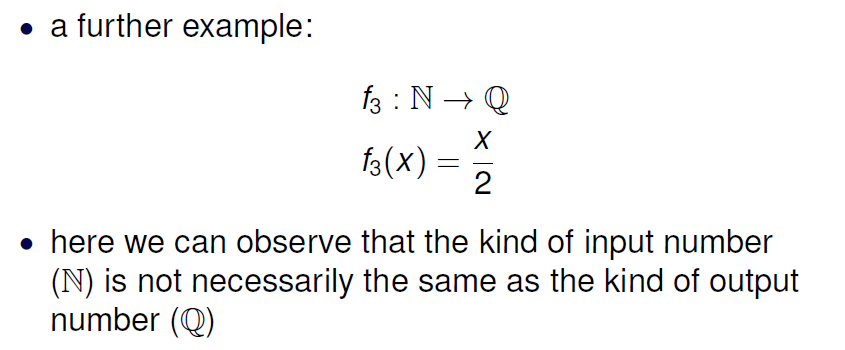


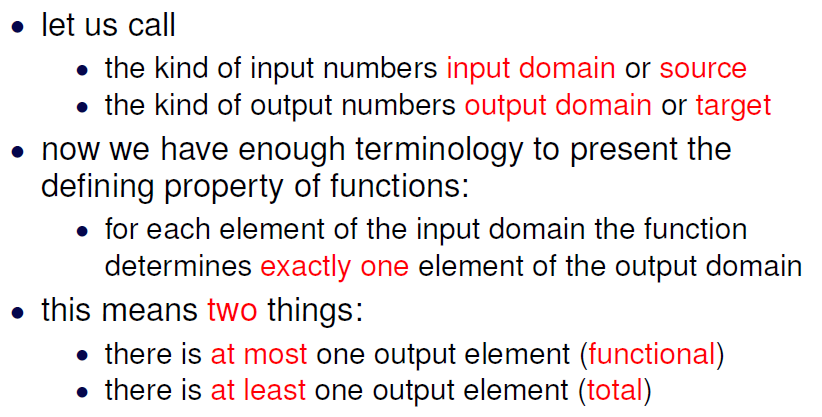
# Woche 2

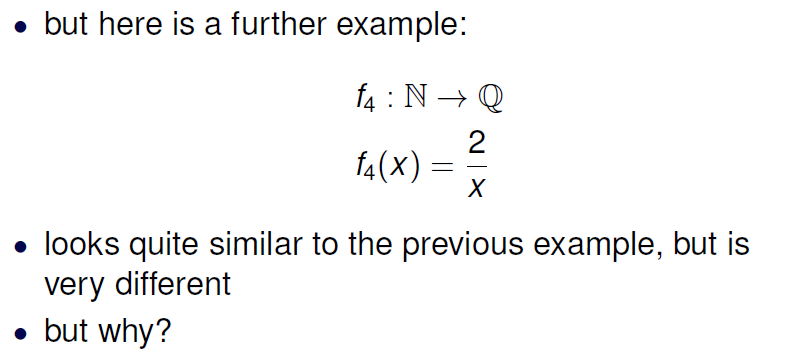
## What is a Function?

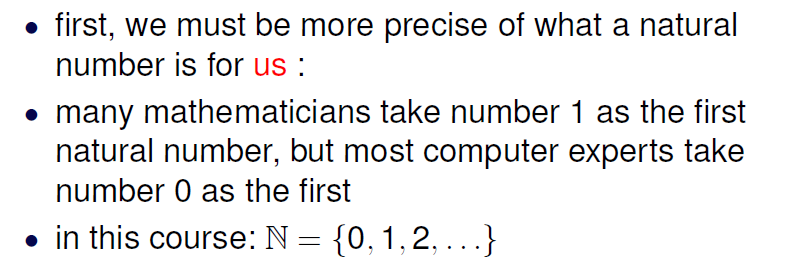


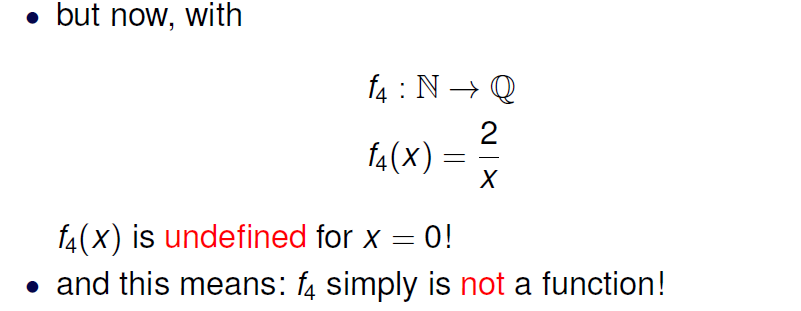


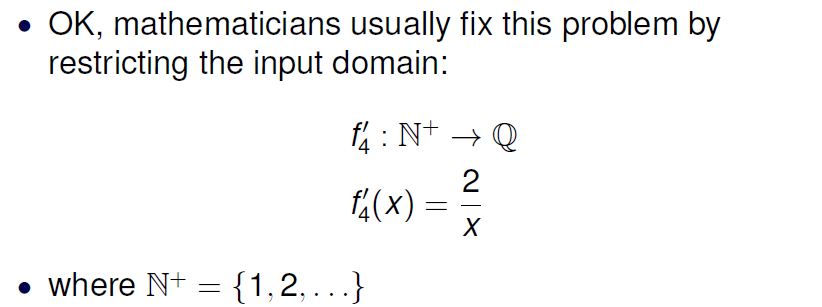




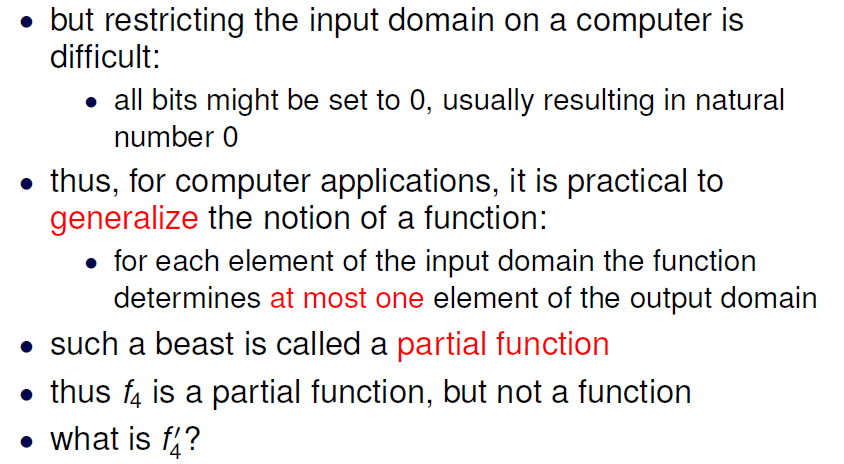




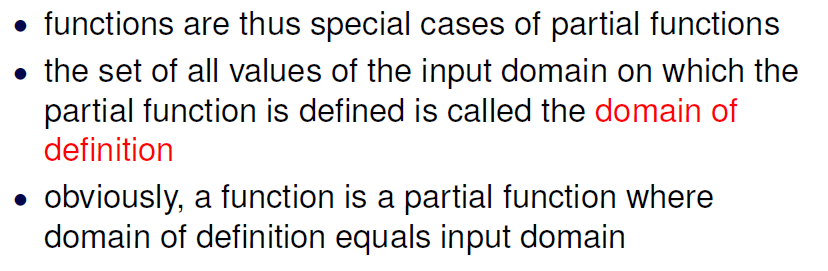




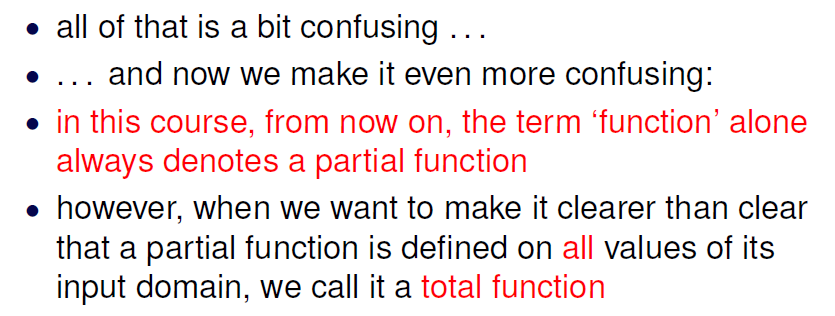
## What is a Partial Function?



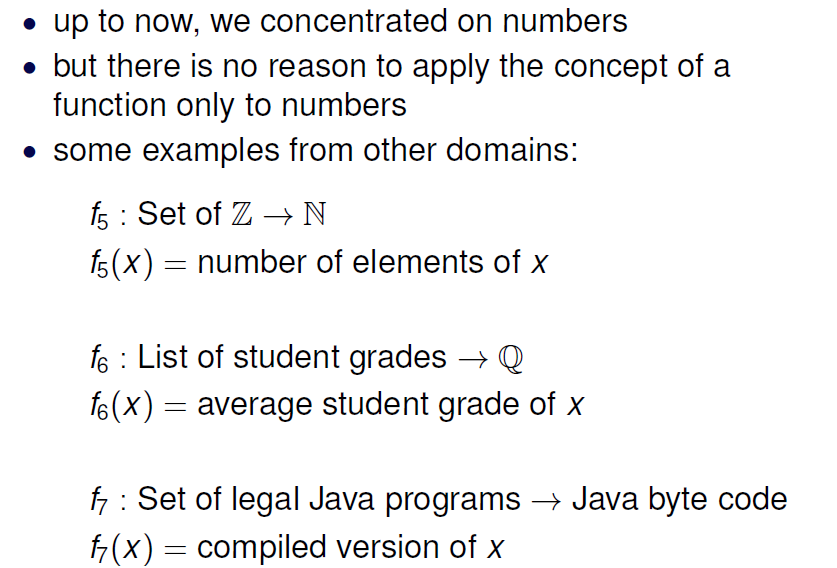
## Domain of Definition

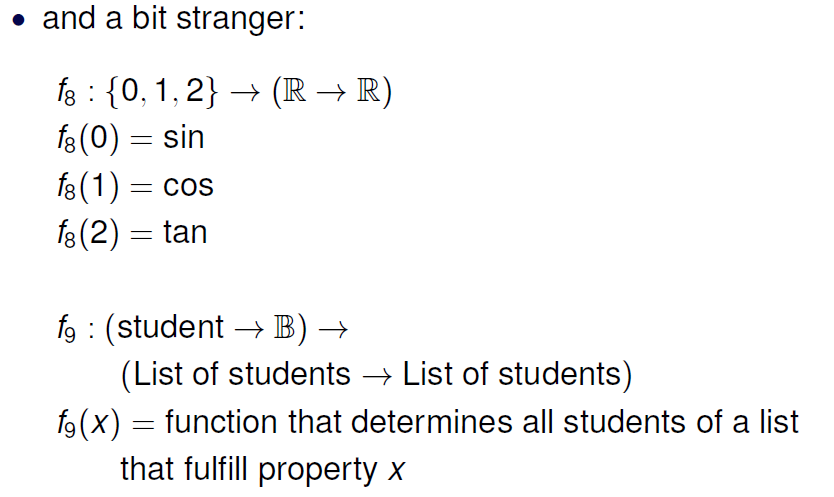


## Confusion



## Additional types



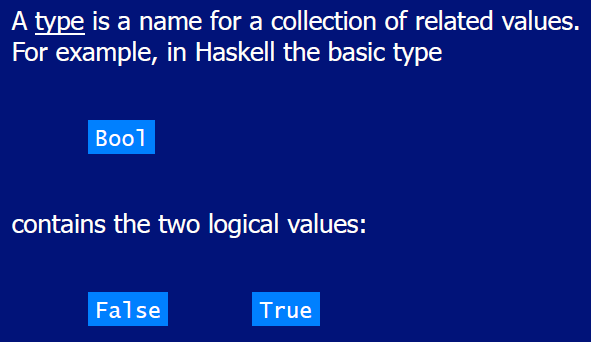


## Summary

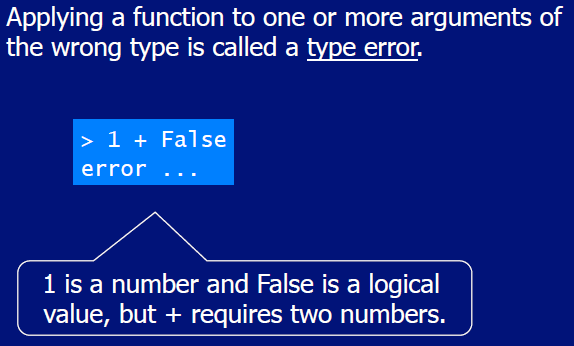


# Woche 3

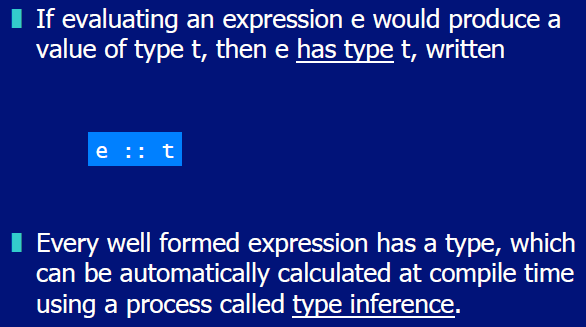
## What is a Type

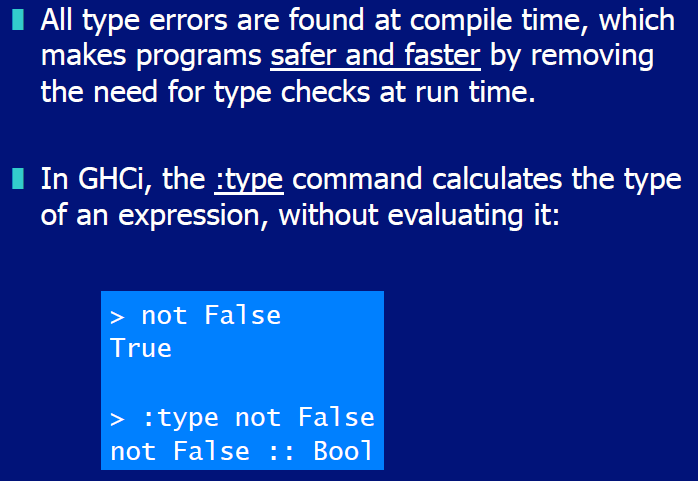


## Type Errors

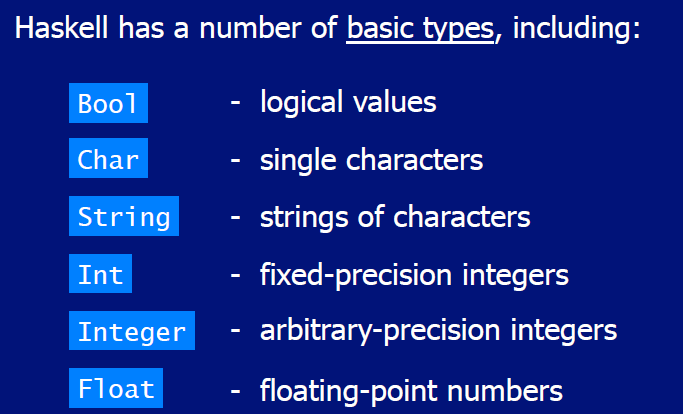


## Types in Haskell

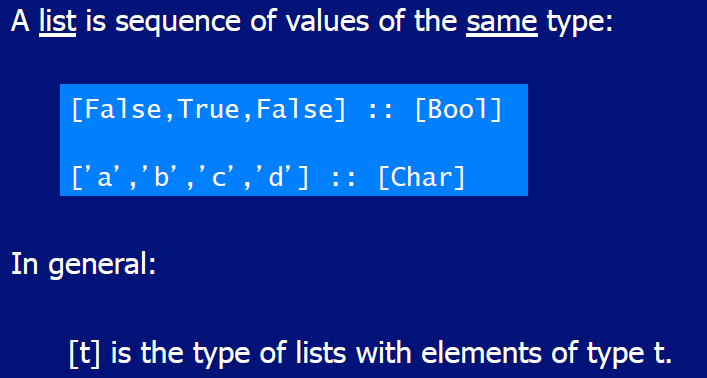


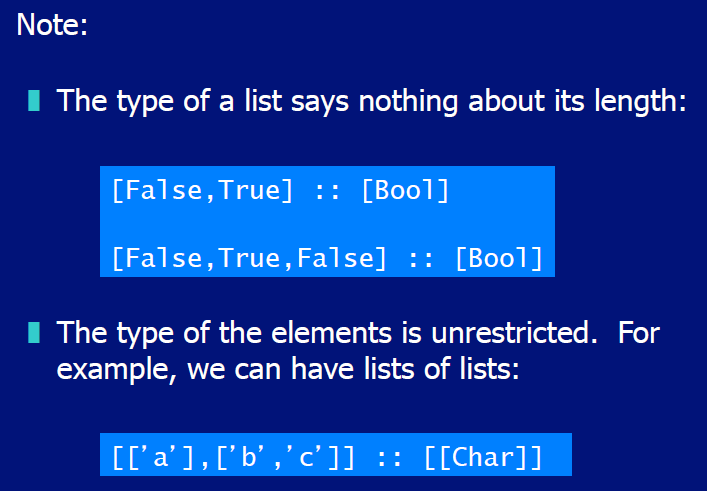


## Basic Types

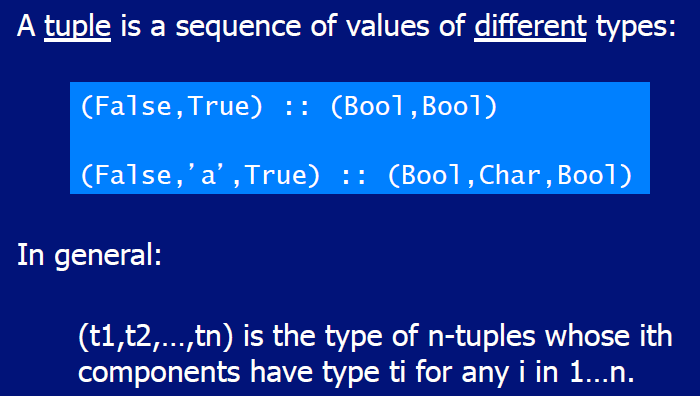


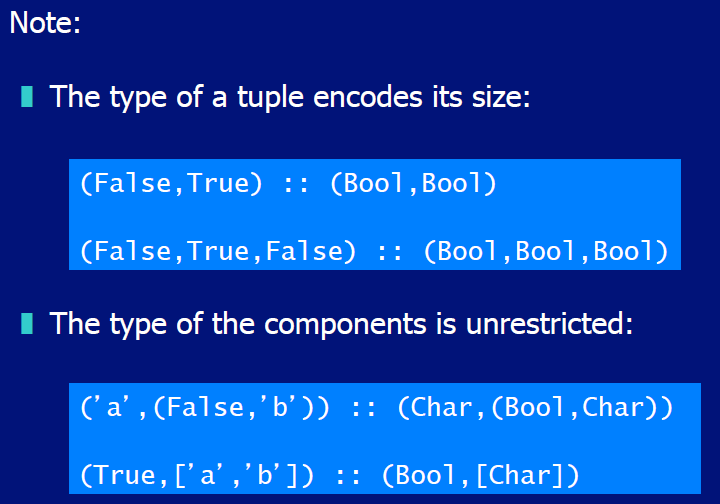
## List Types



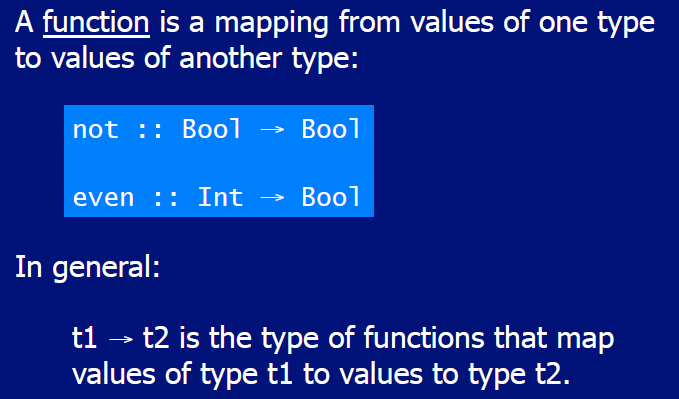


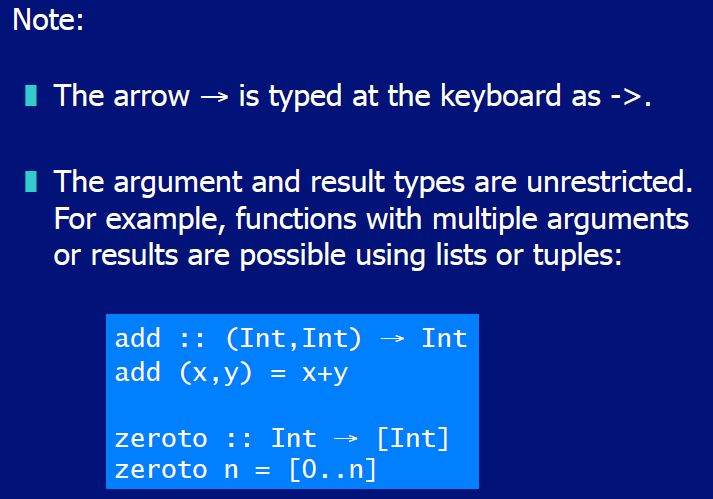
## Tuple Types



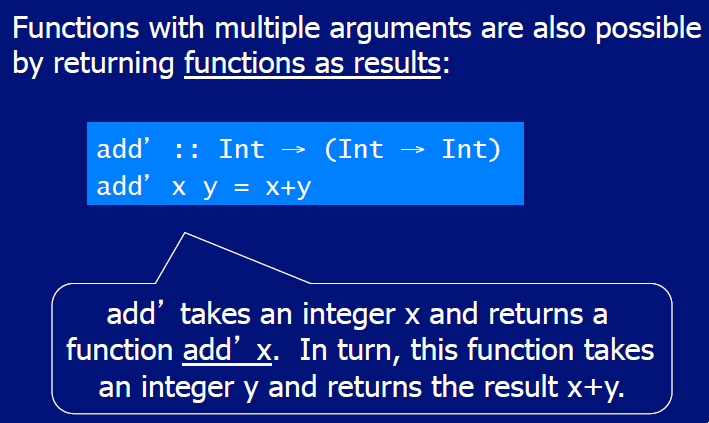


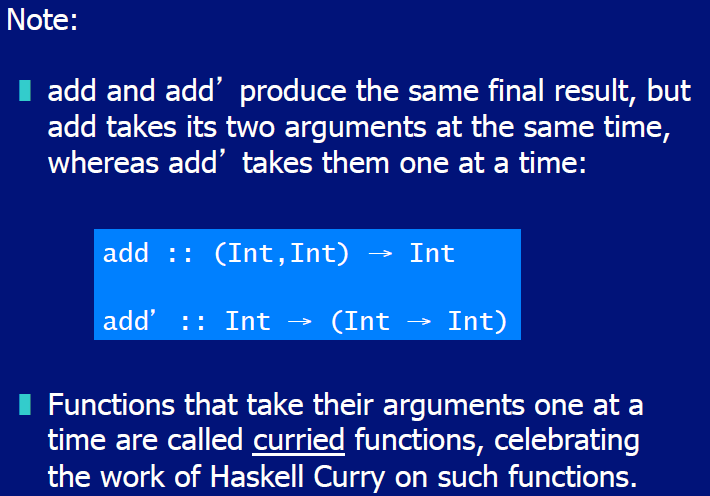
## Function Types

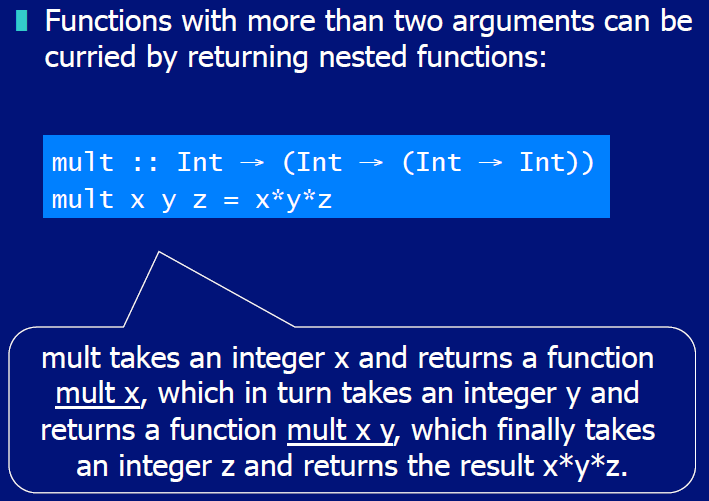




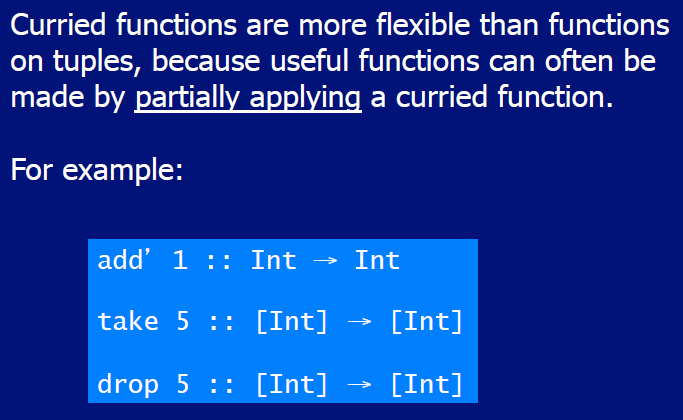
## Curried Functions



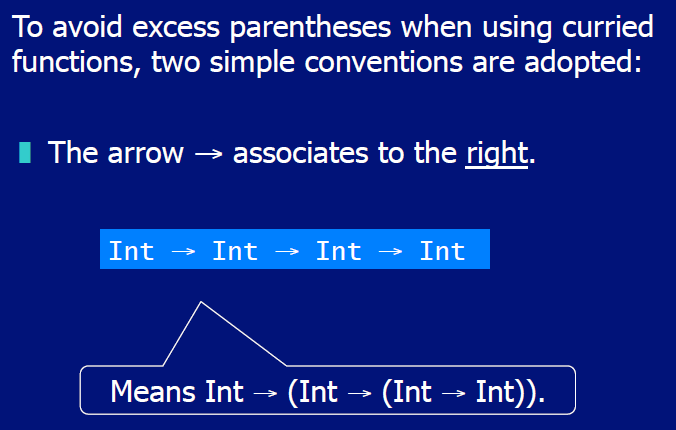


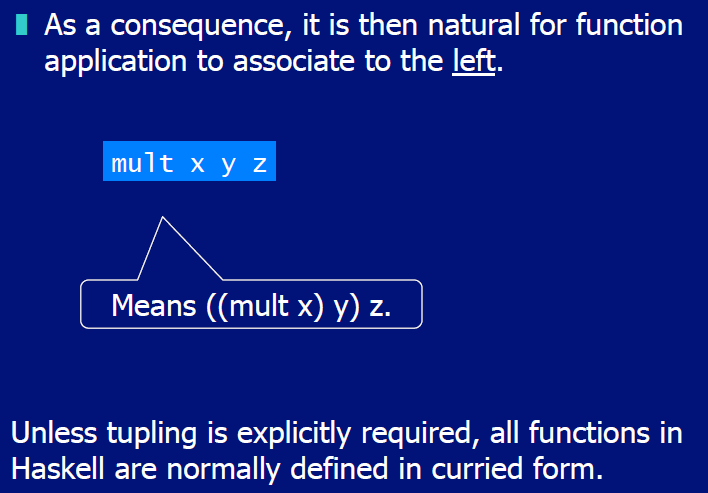


## Why is Currying useful

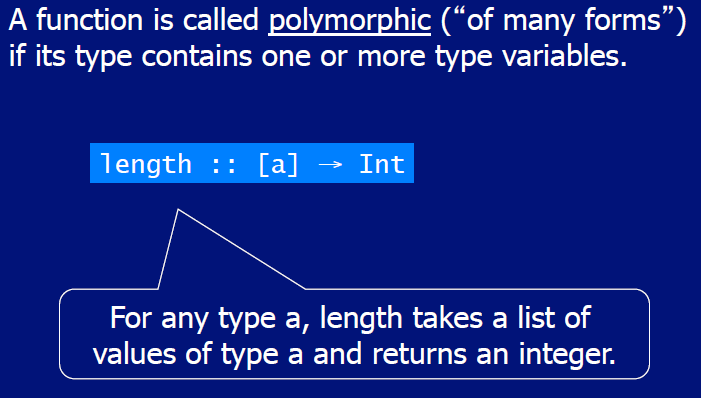


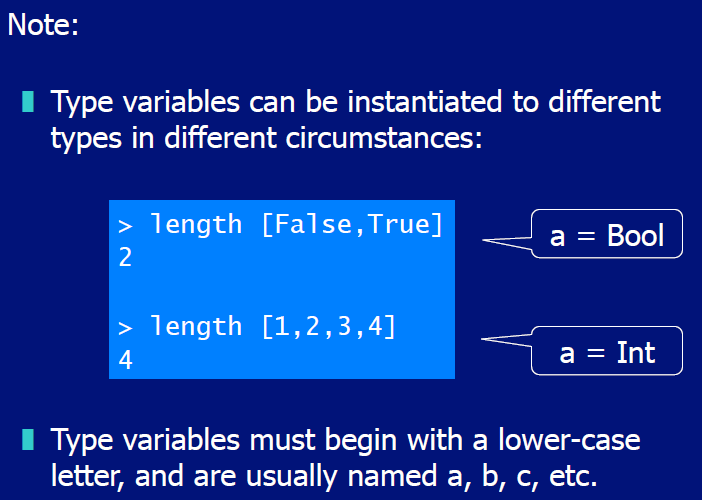
## Currying Conventions

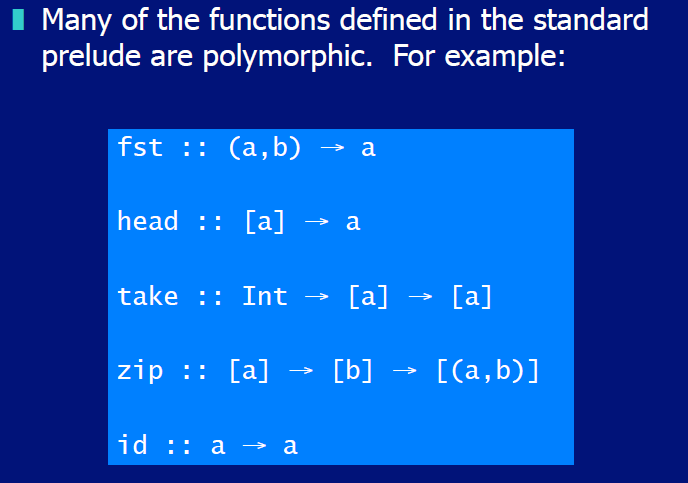




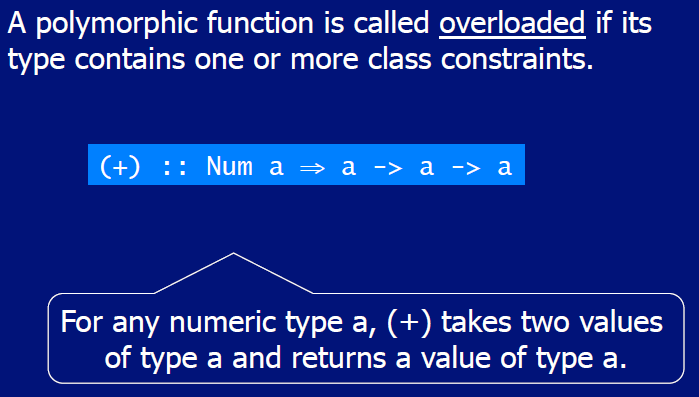
## Polymorphic Functions

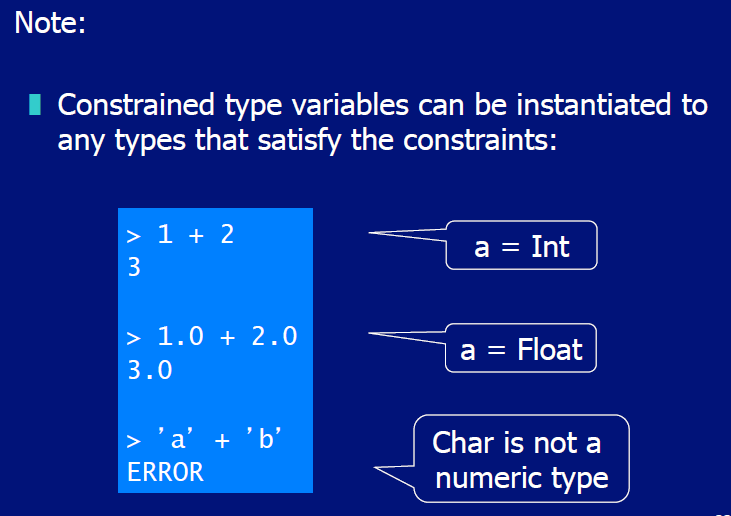


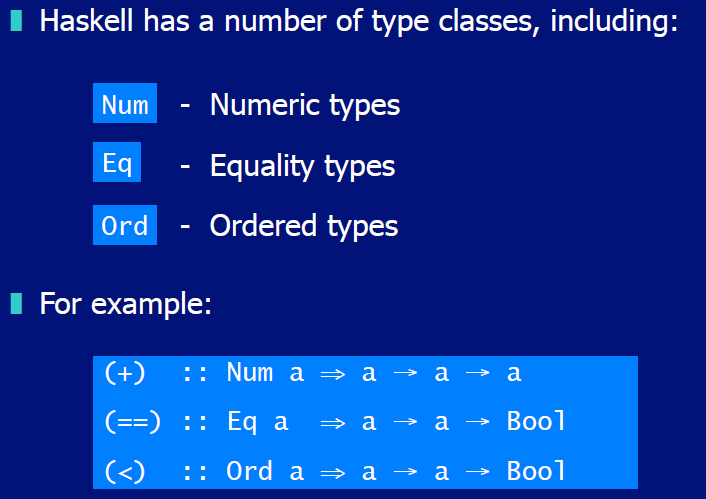




## Overloaded Functions





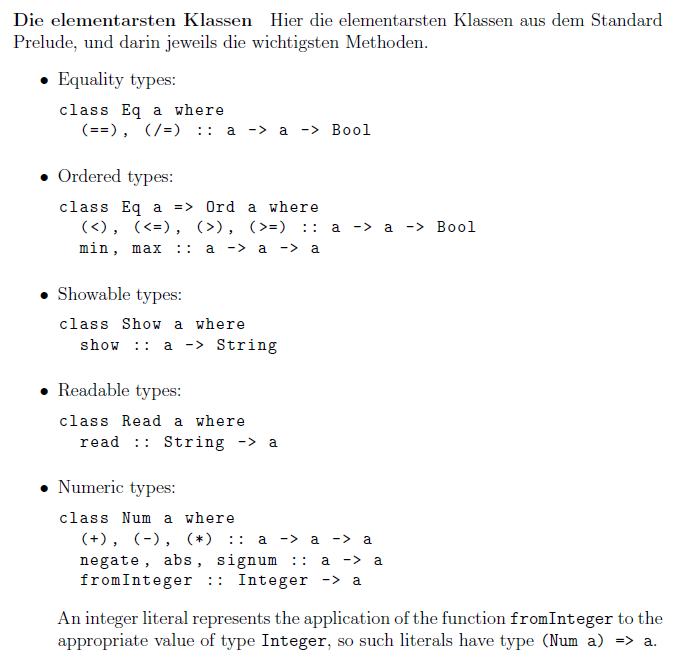


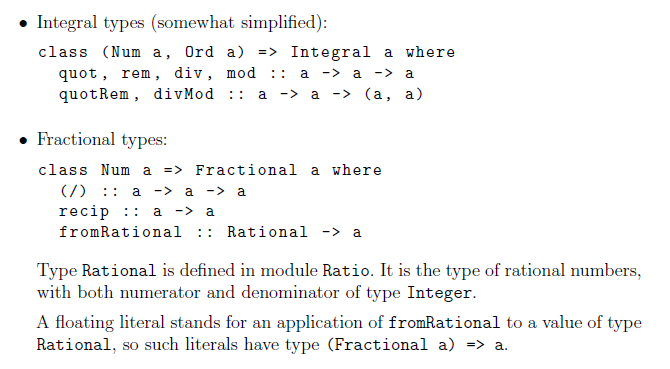
## Hints and Tips



# Woche 4

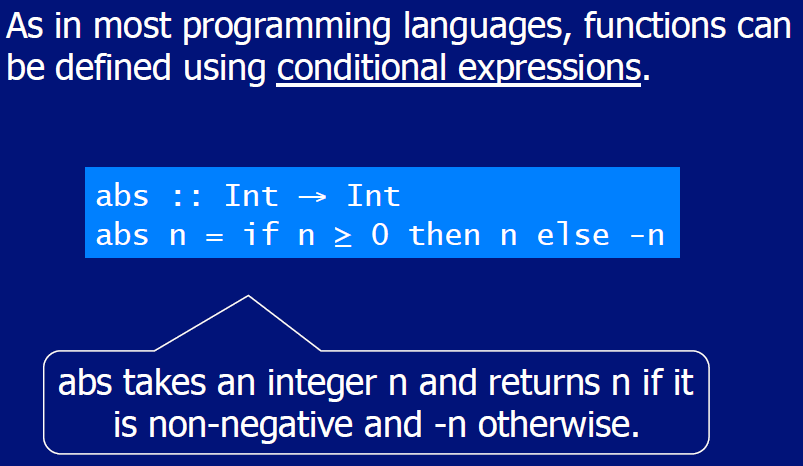
## Typeclasses

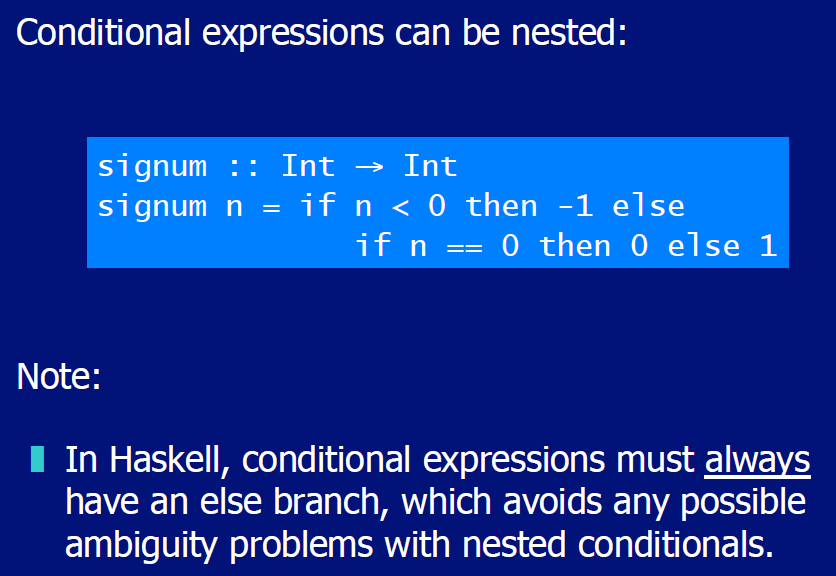




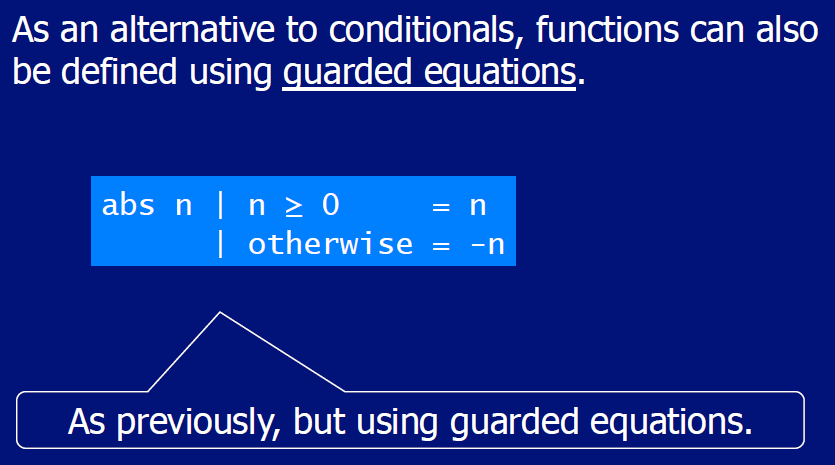
# Woche 5

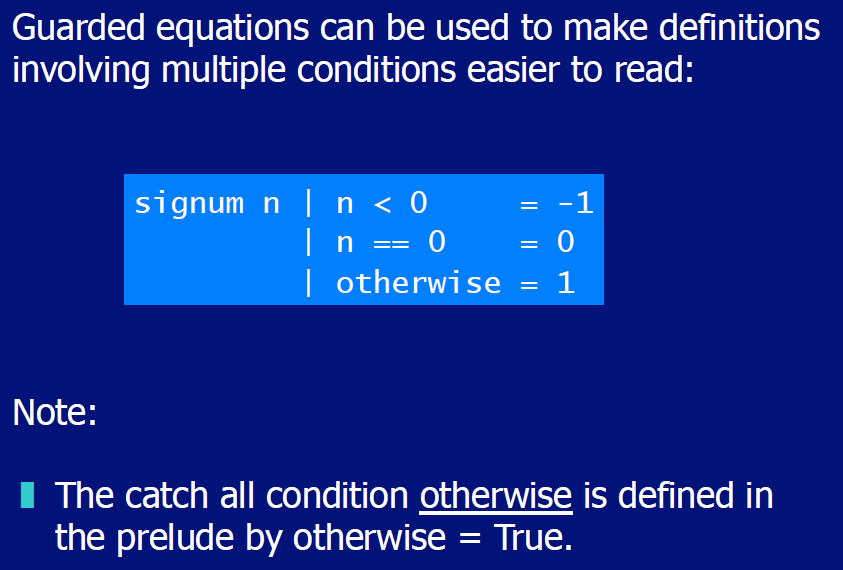
## Conditional Expressions



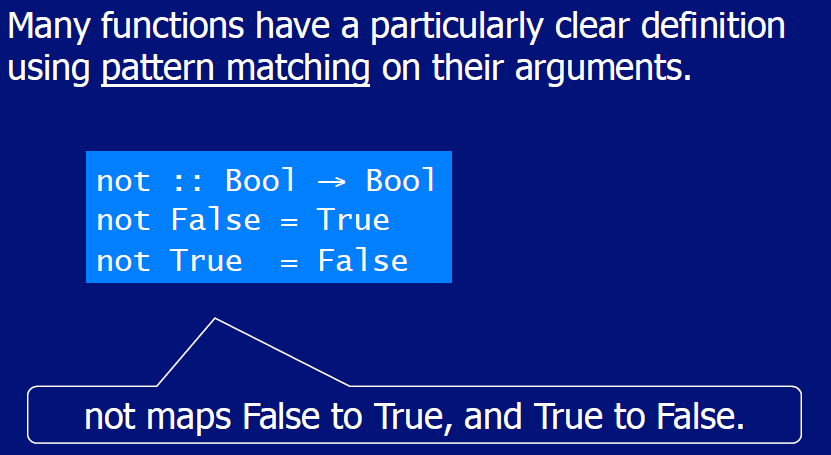


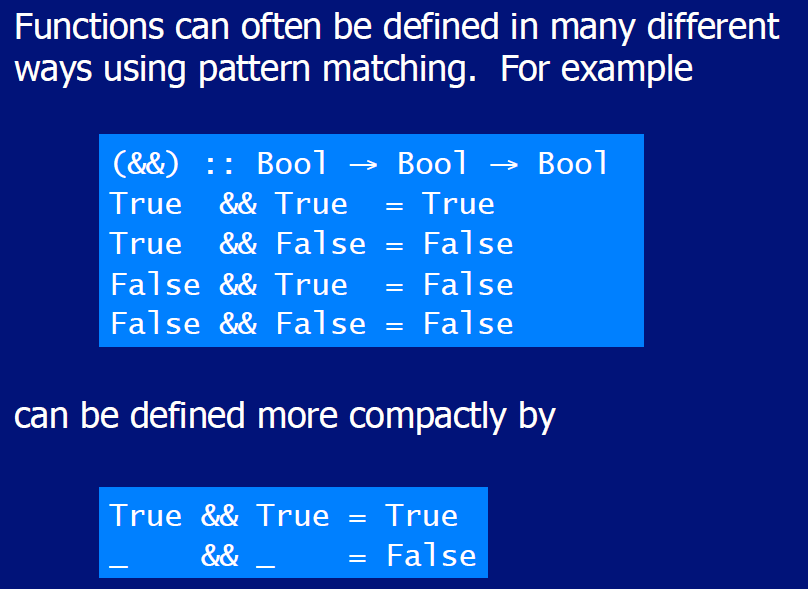
## Guarded Equations

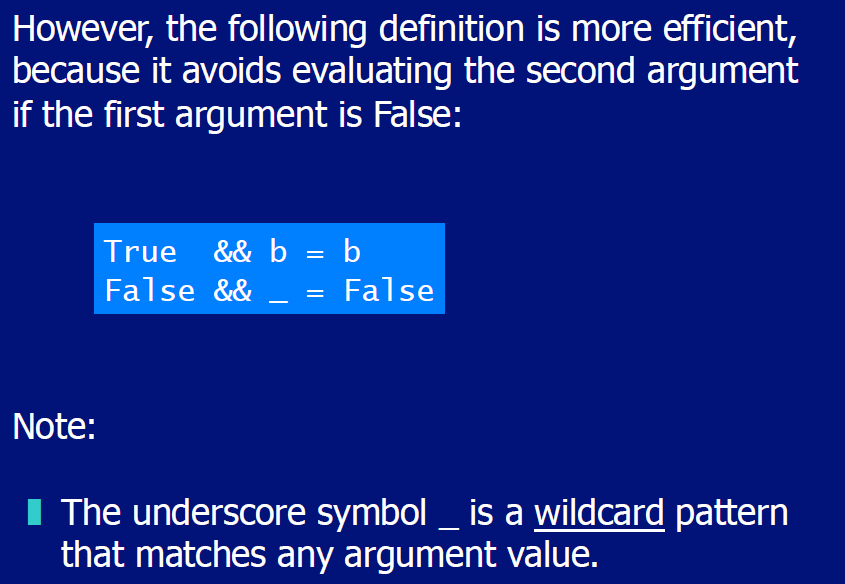


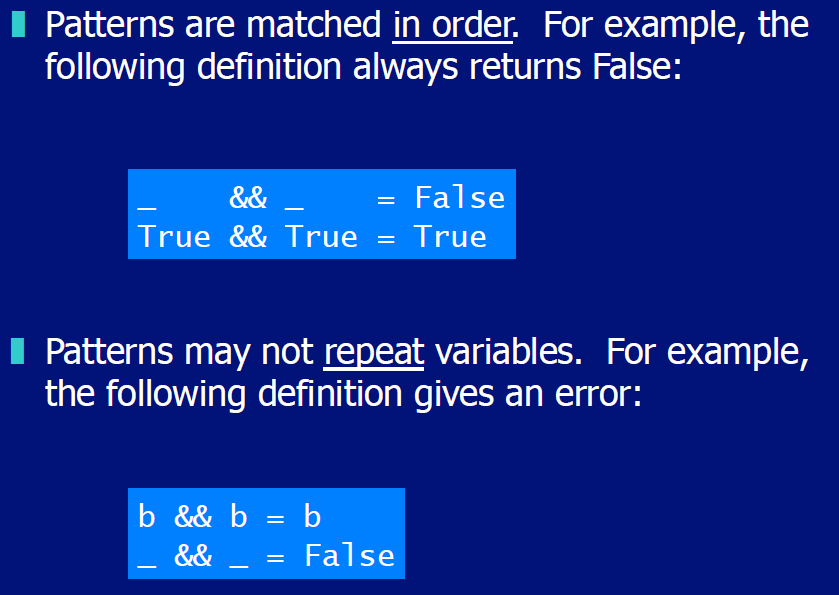


## Pattern Matching

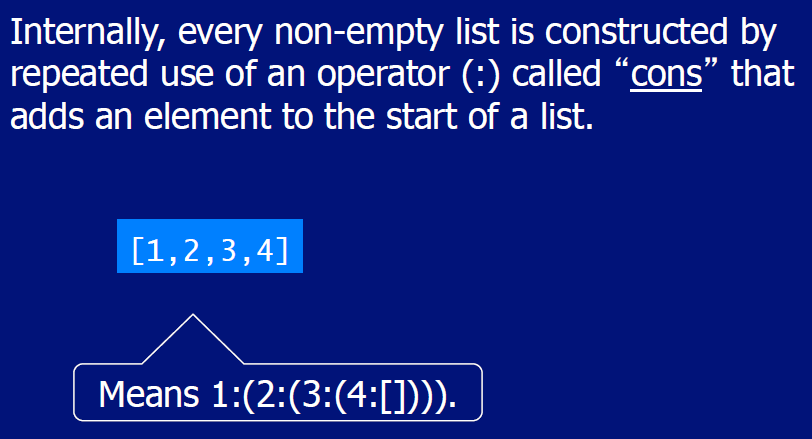


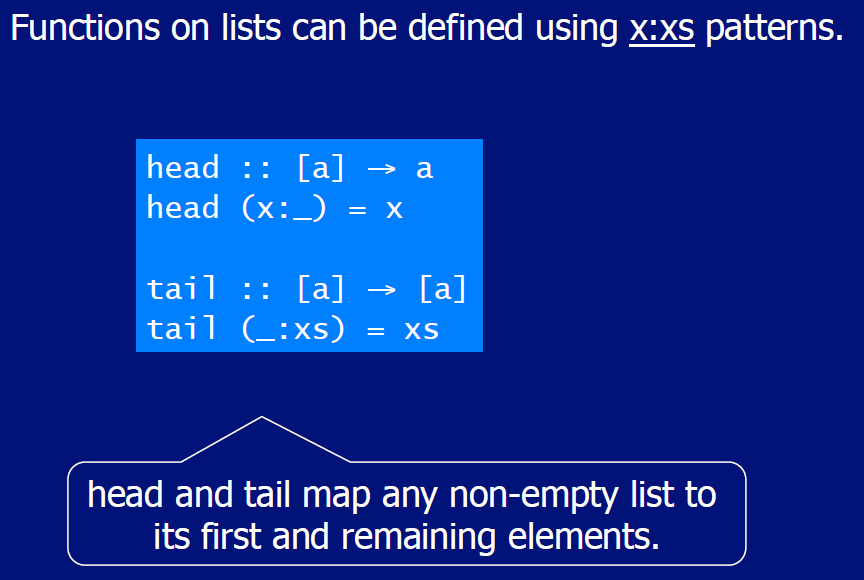


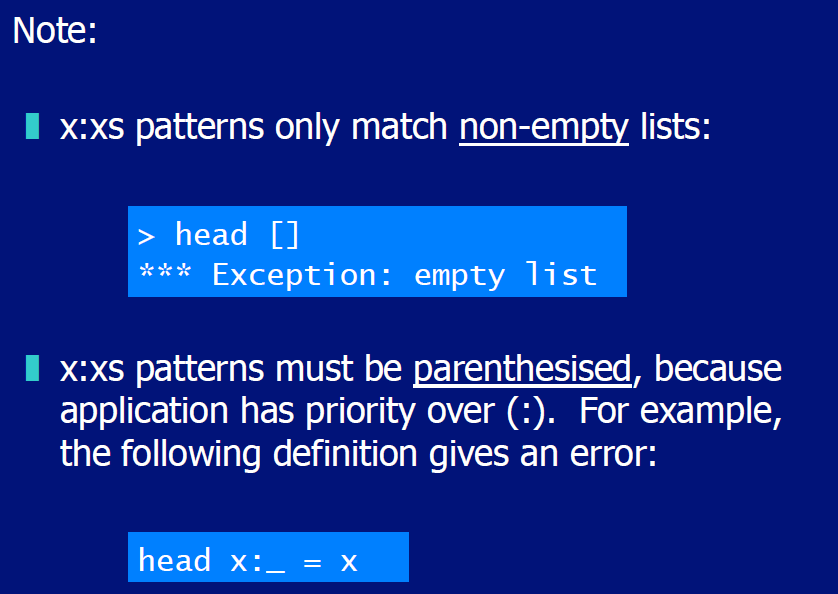




## List Patterns

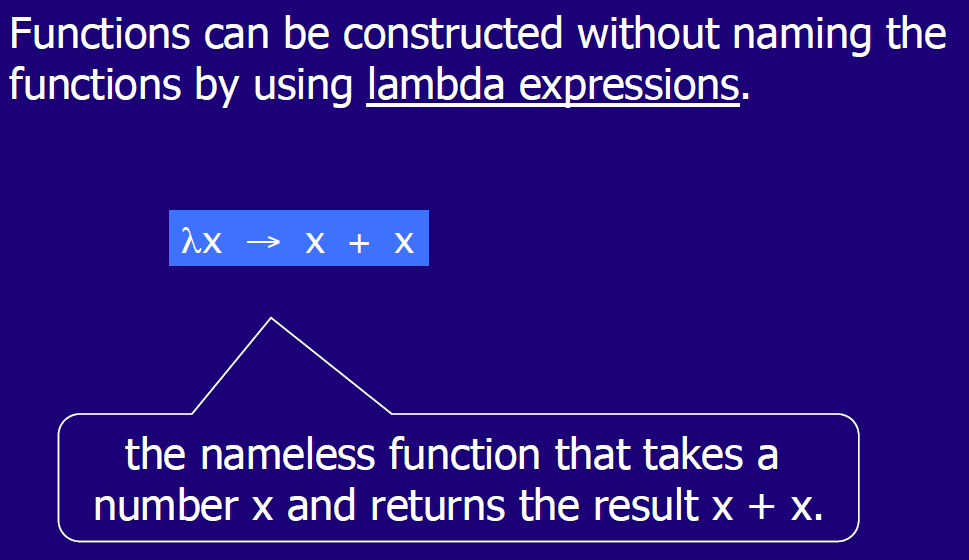


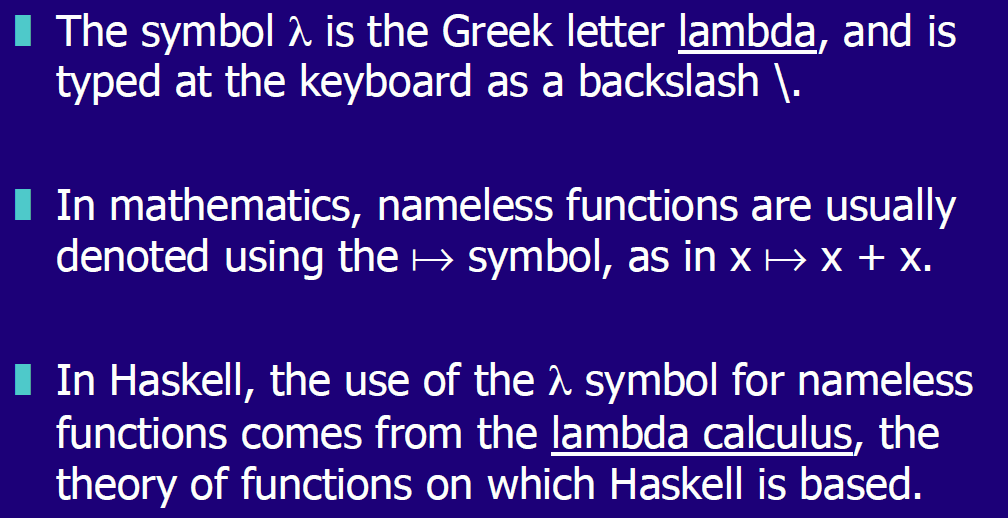




# Woche 6

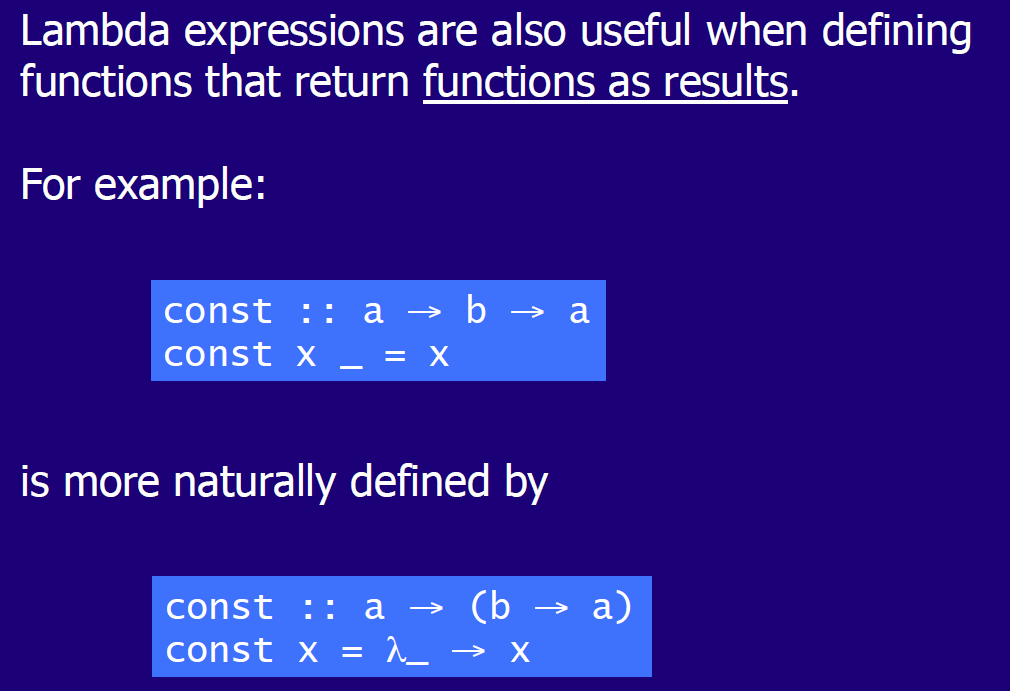
## Lambda Expressions

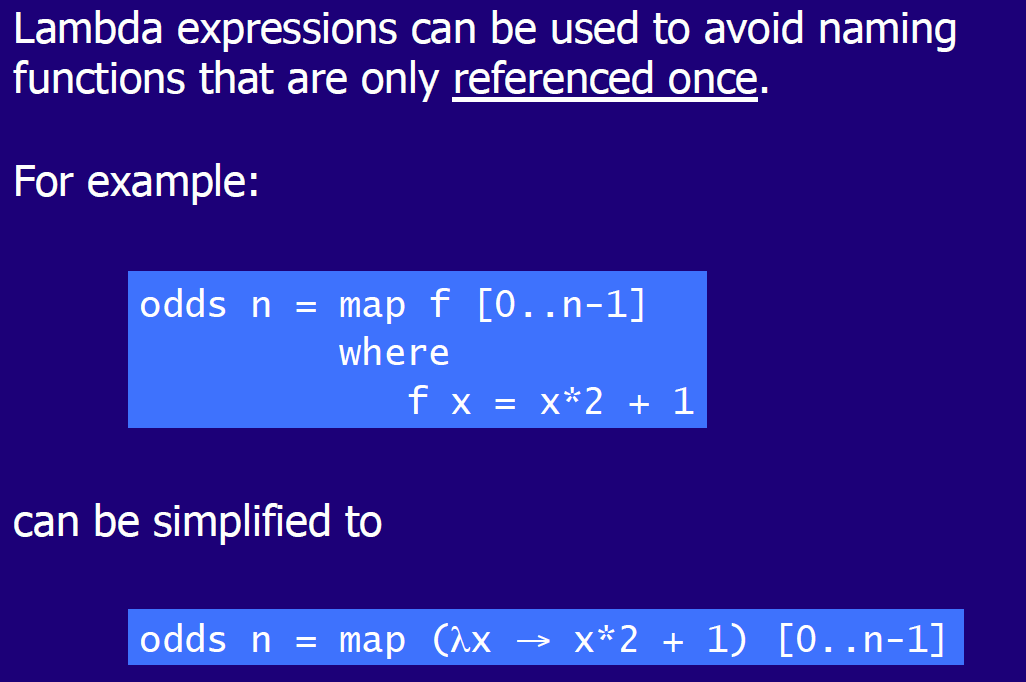




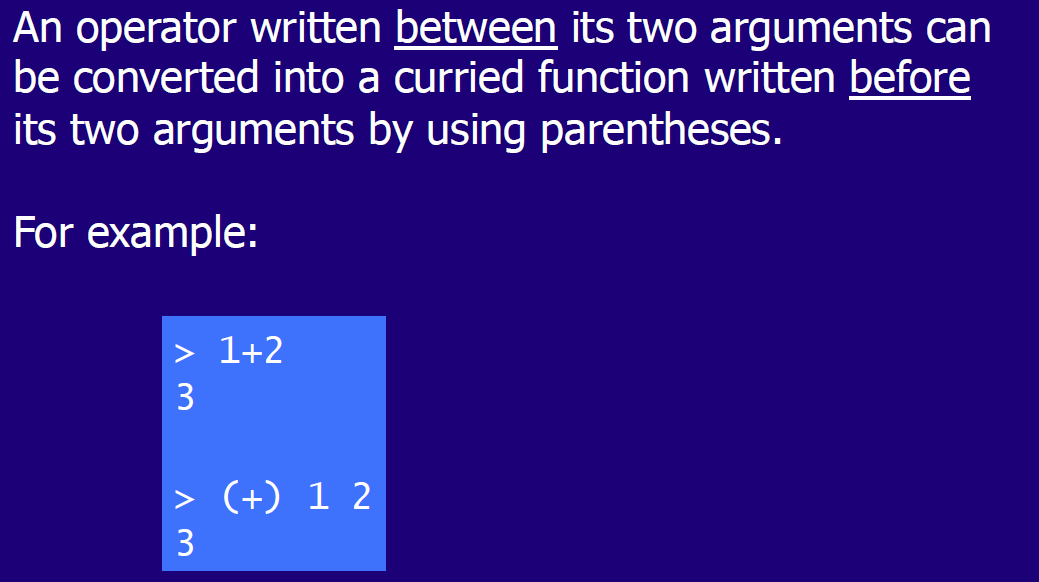
## Why are Lambdas useful

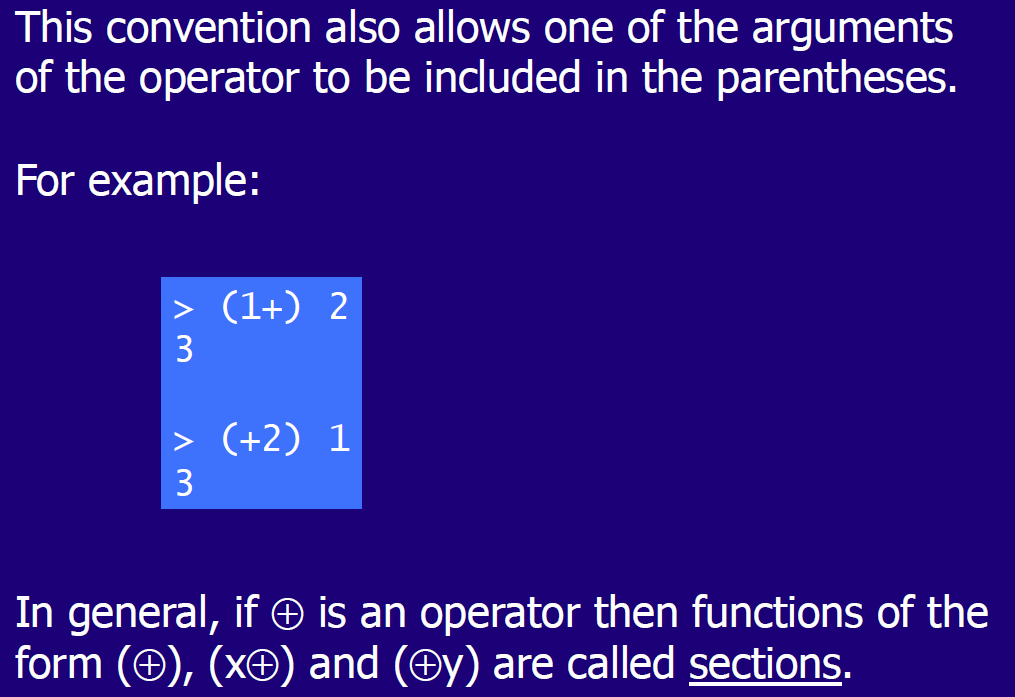




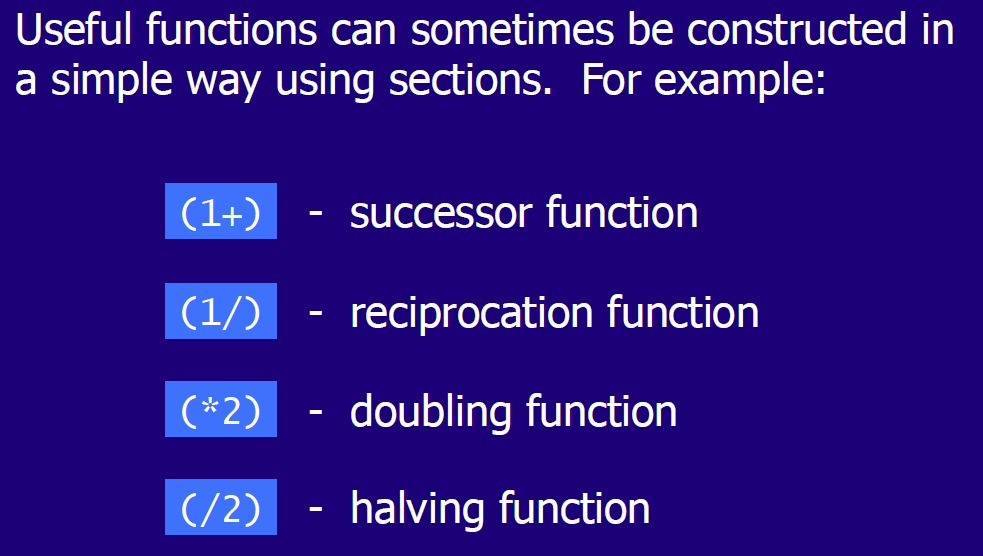


## Operator Sections

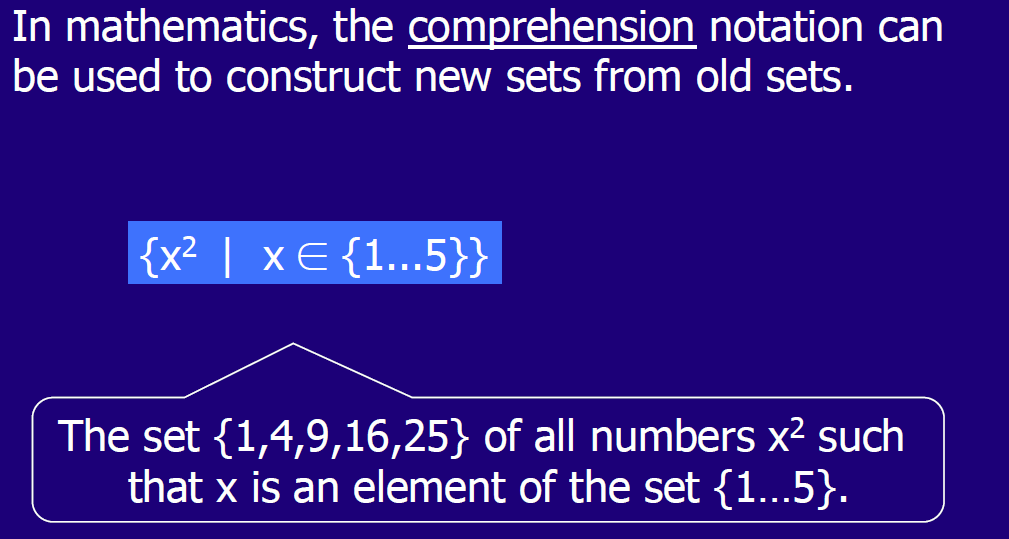




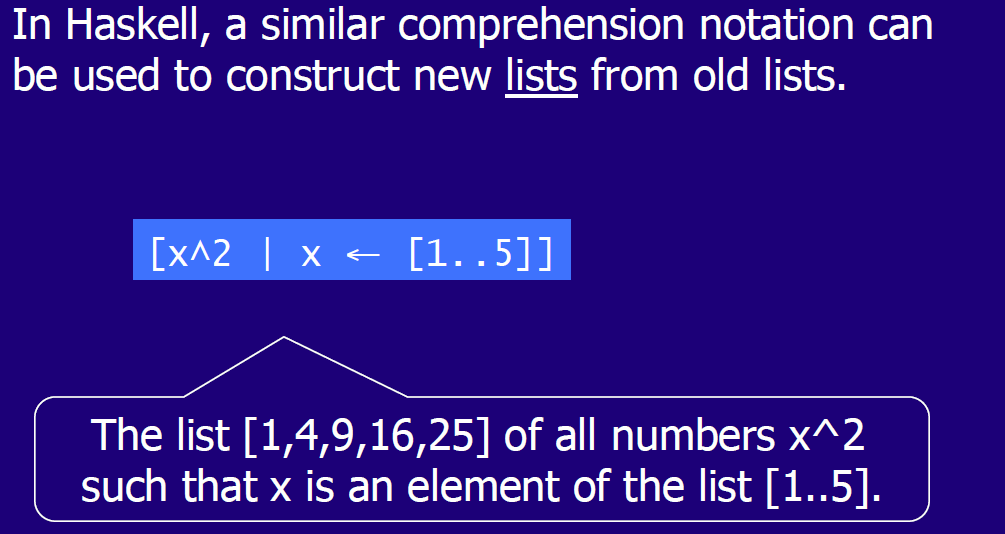
## Why are Sections useful

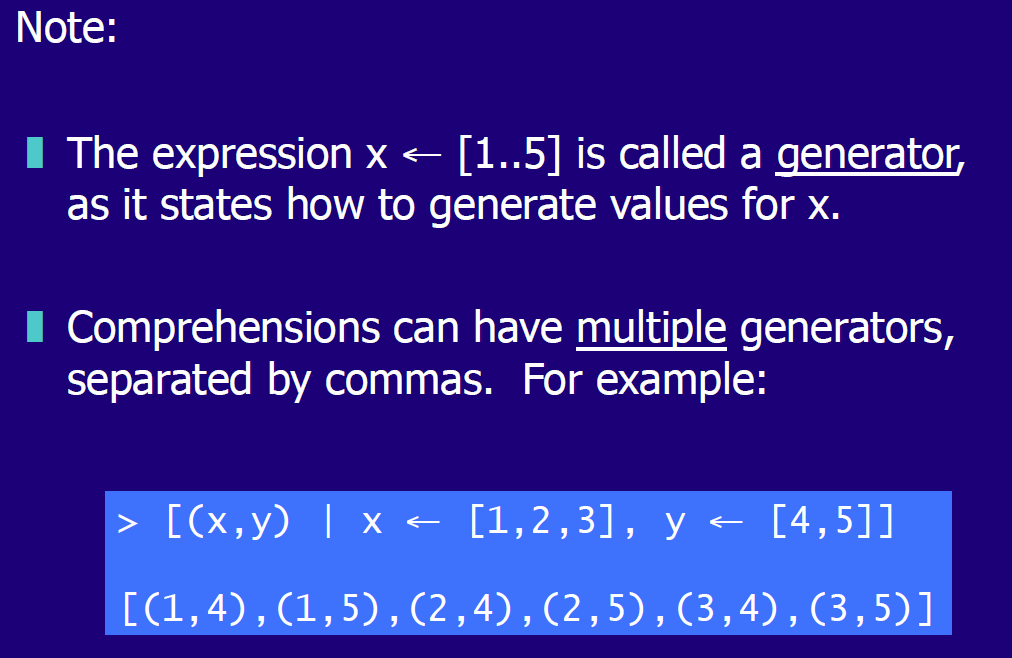


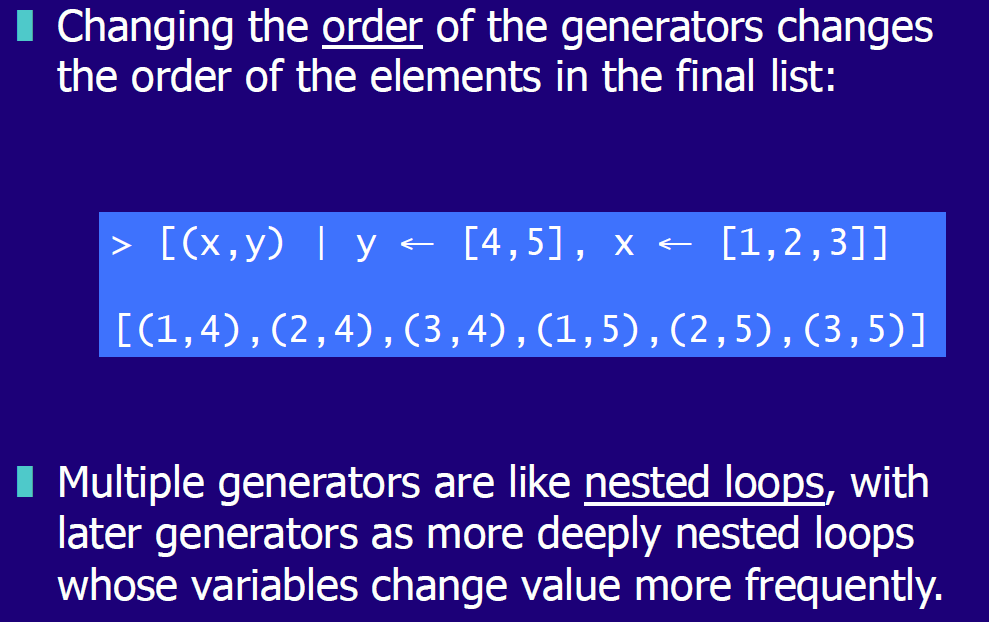
## Set Comprehensions

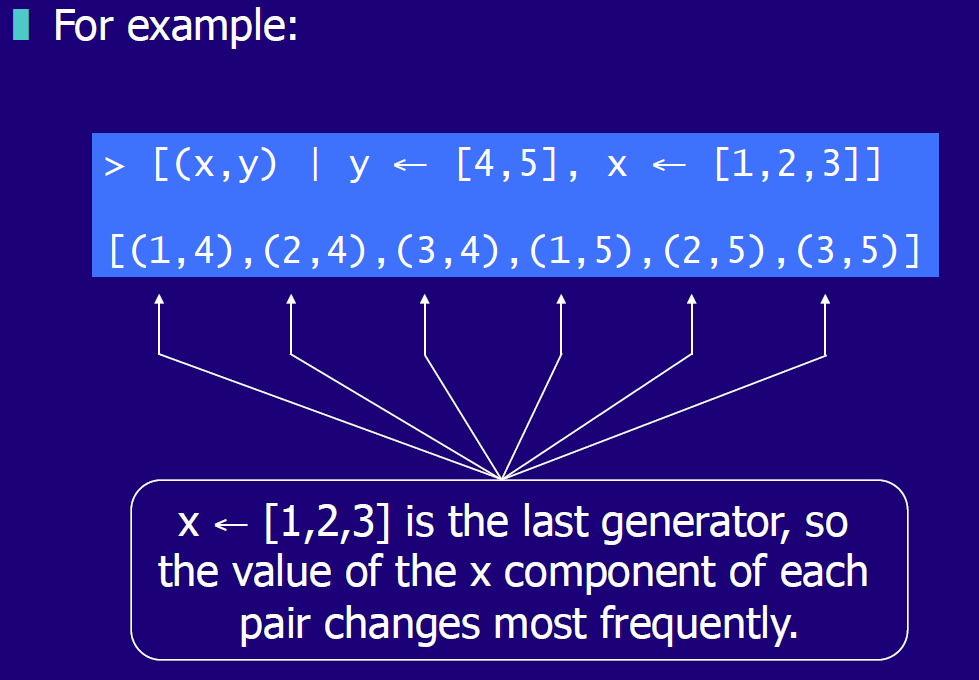


## List Comprehensions

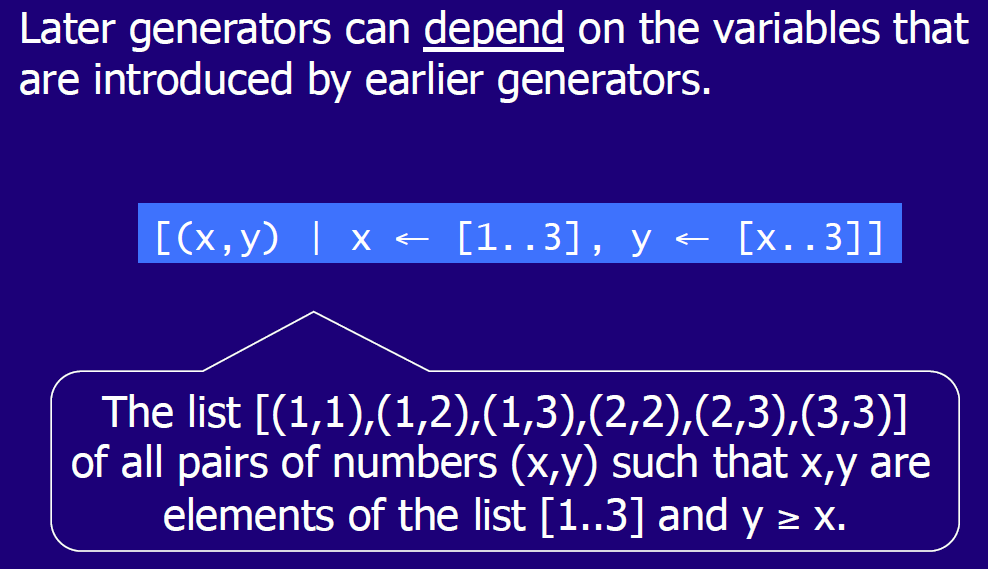


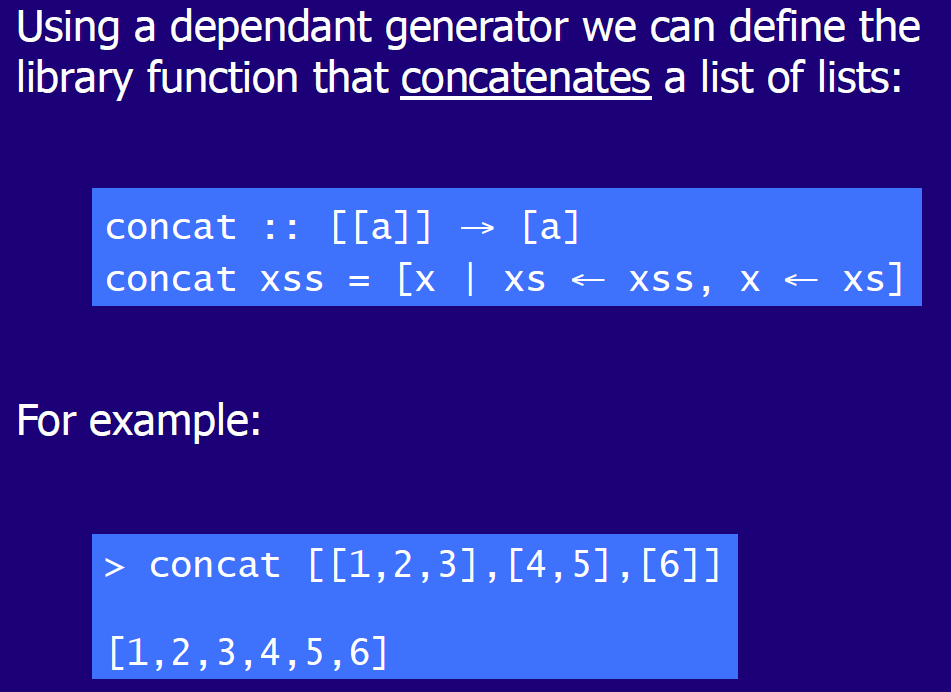




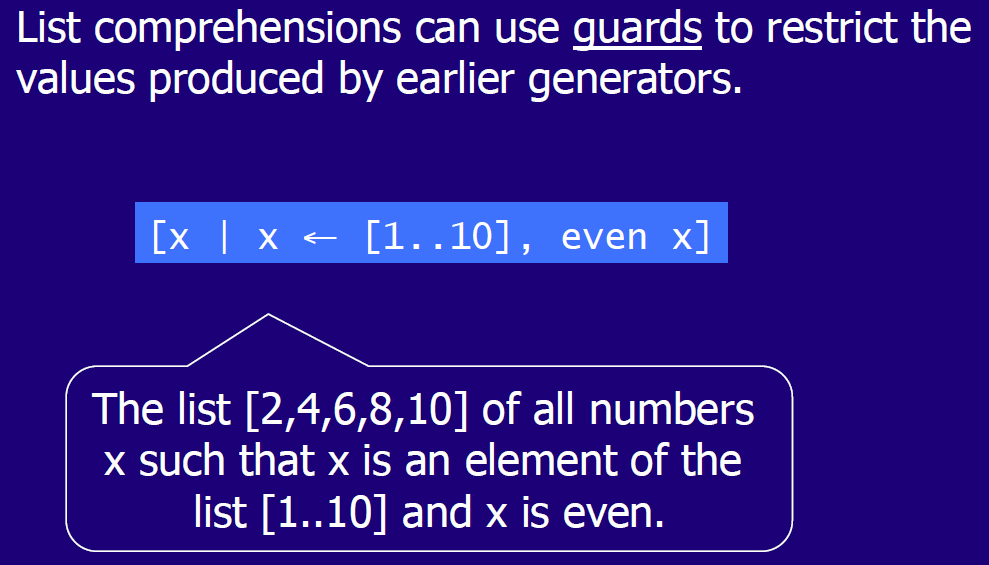


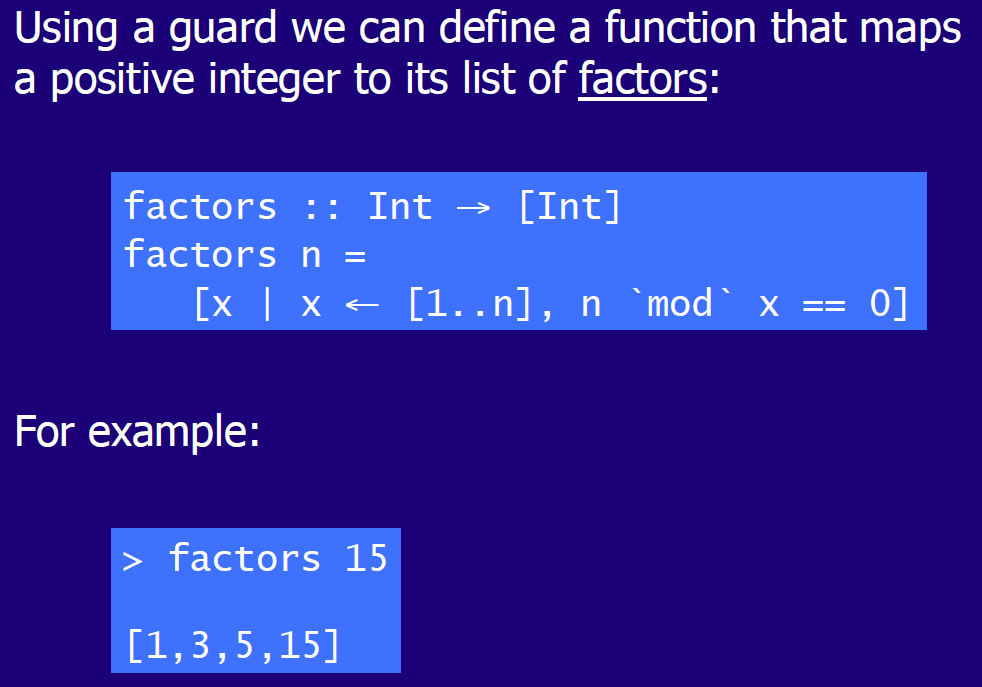
## Dependent Generators

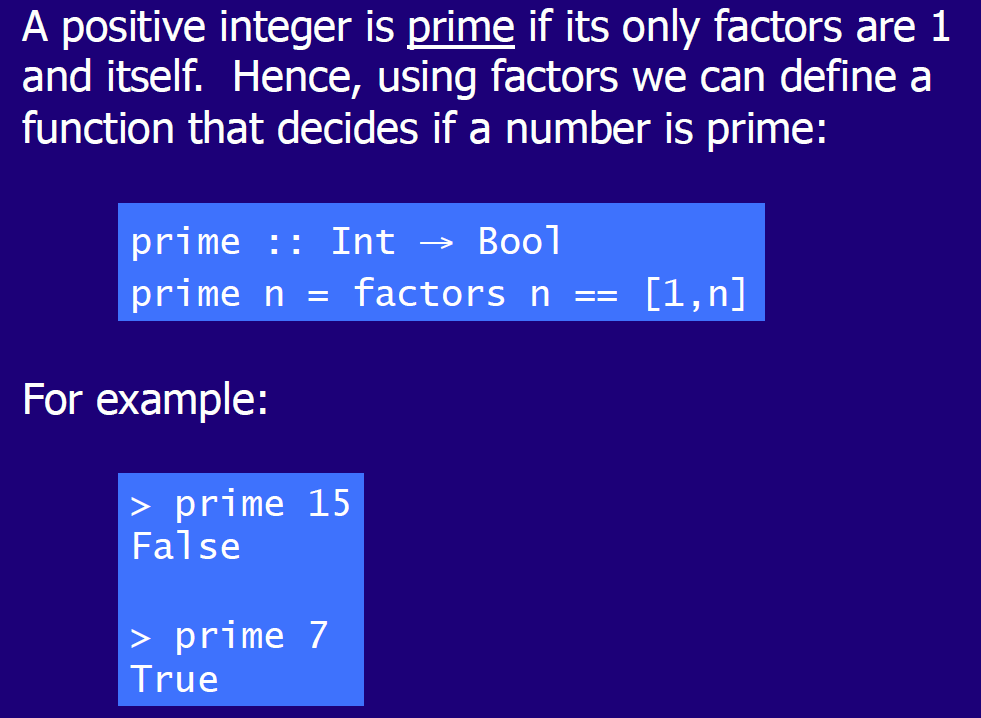


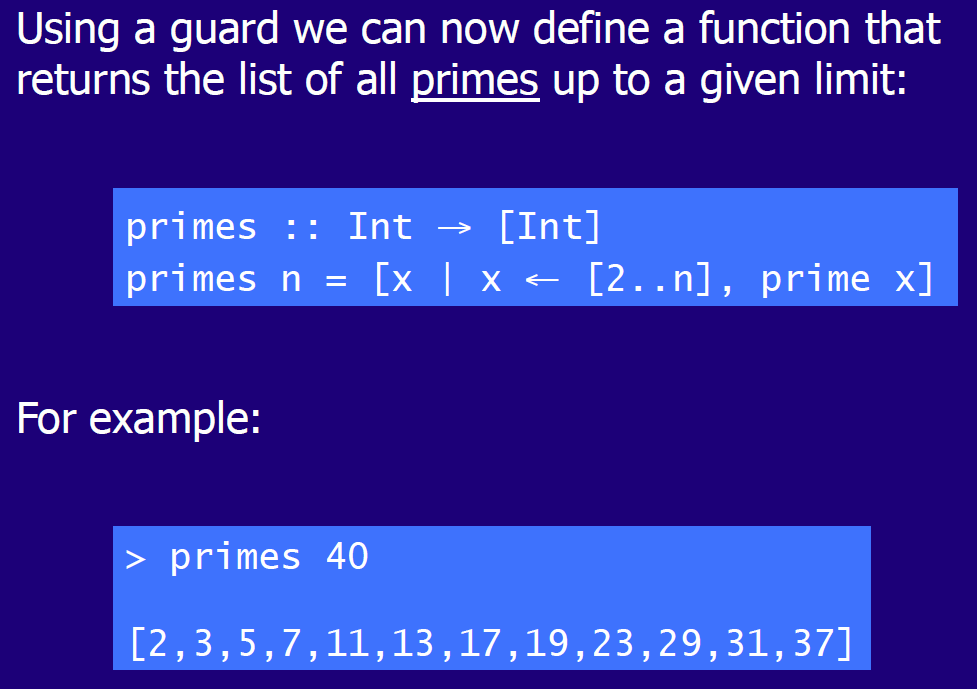


## Guards



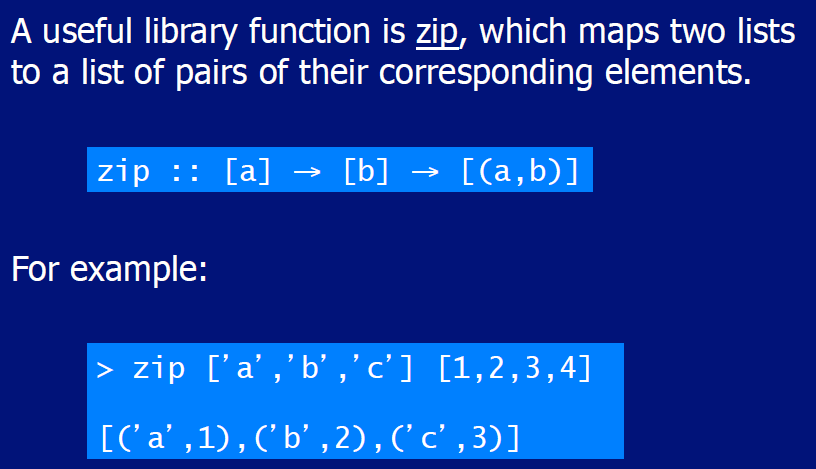


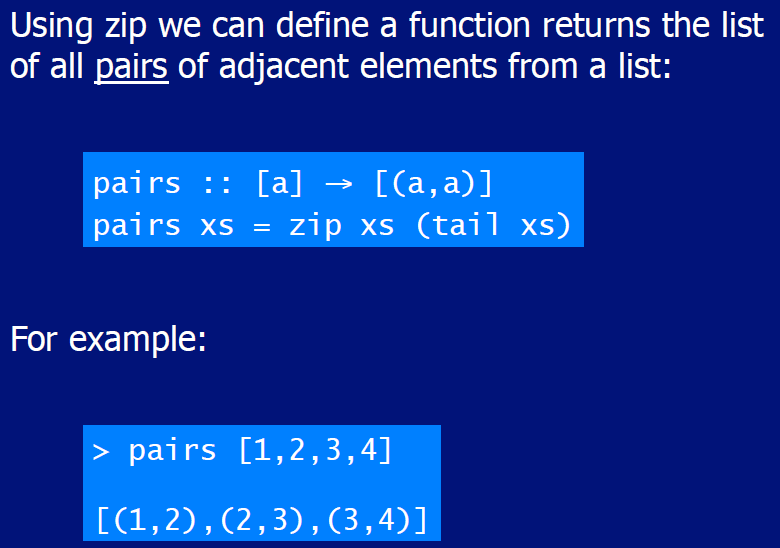


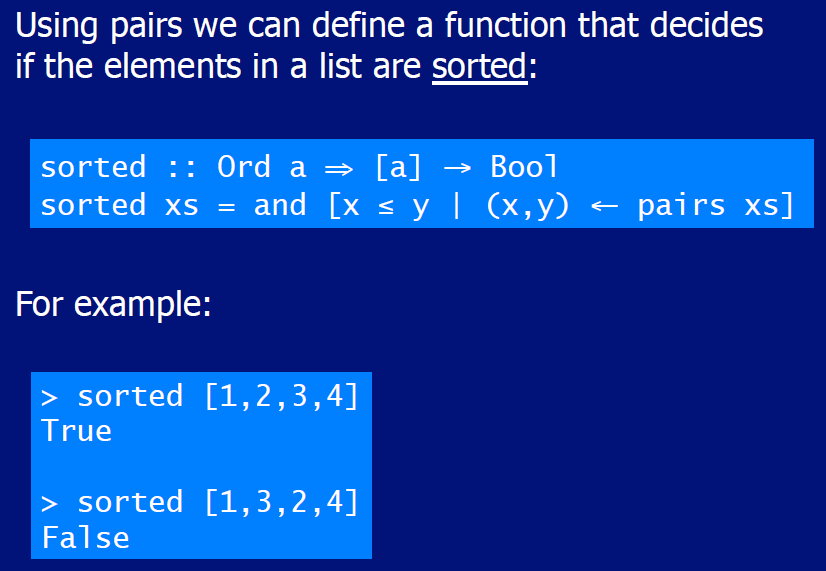


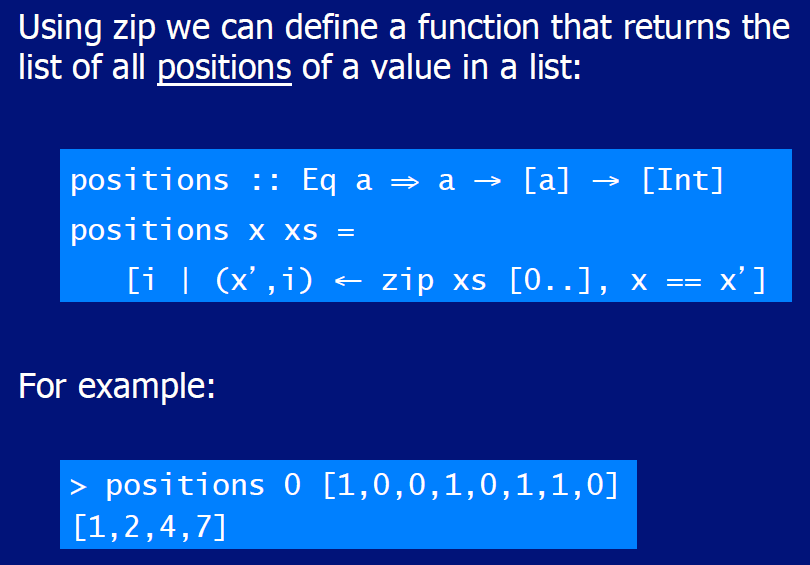
# Woche 7

## Zip Function

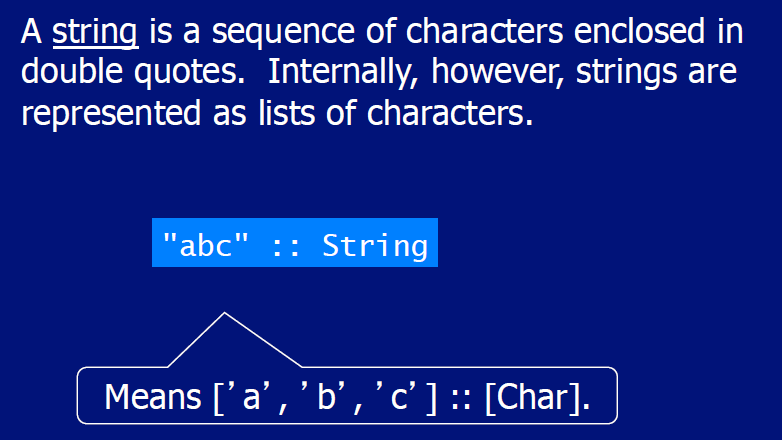


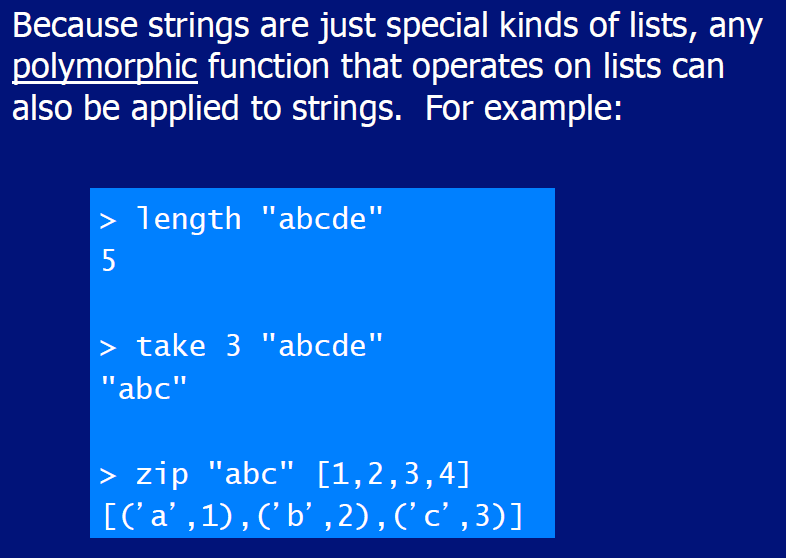


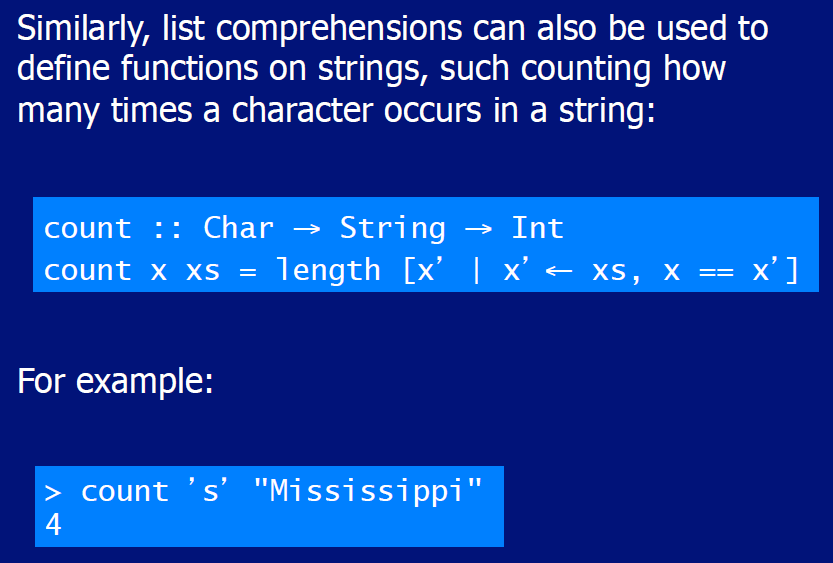




## String Comparison





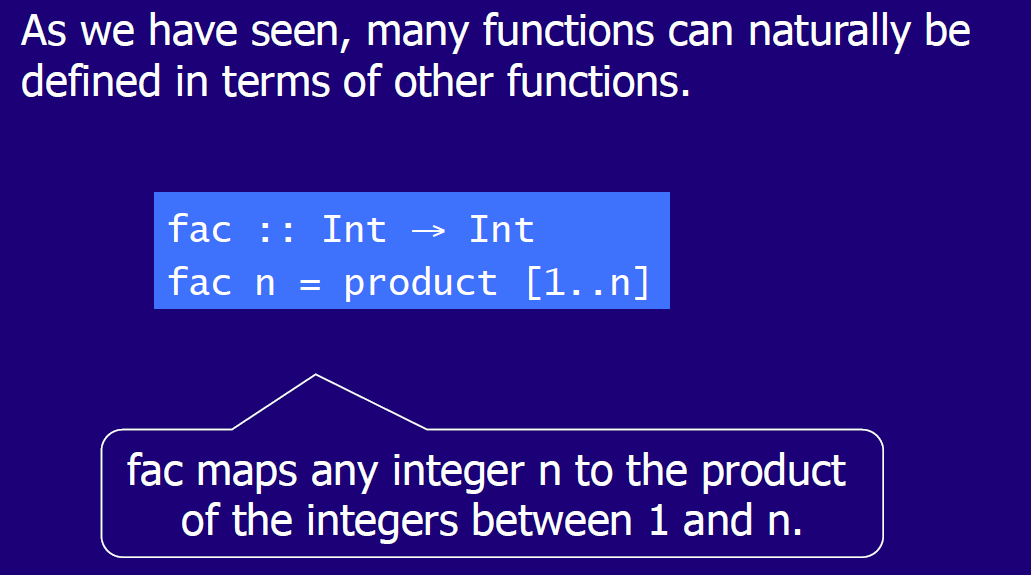


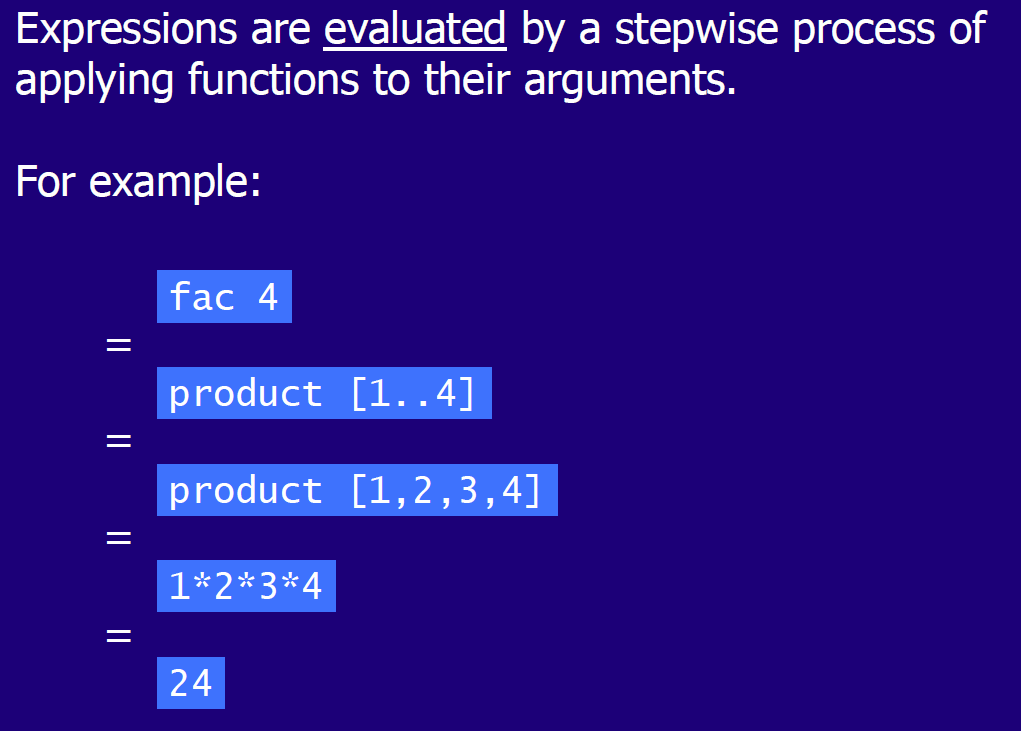
# Woche 8

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

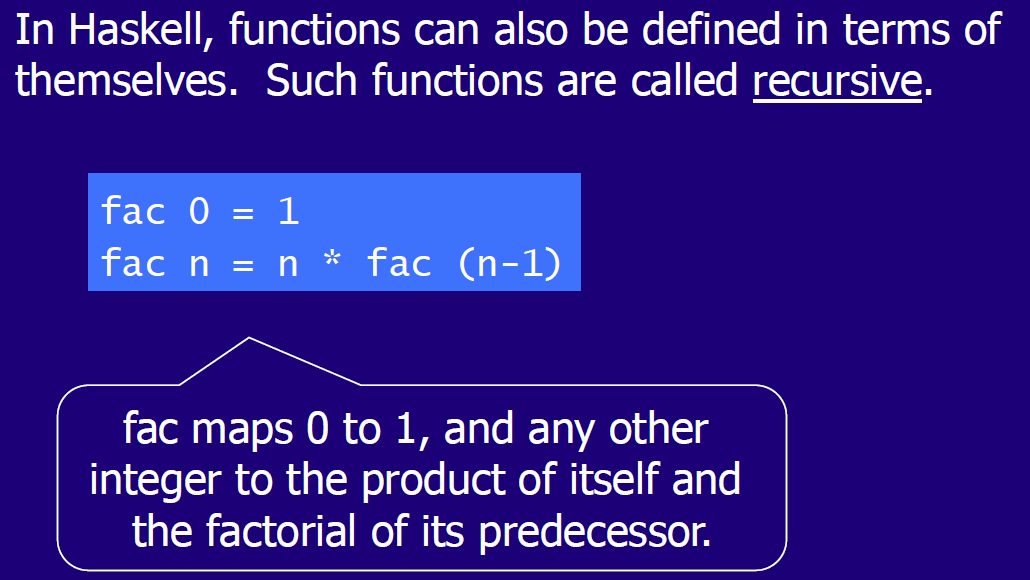
# Woche 9 & 10

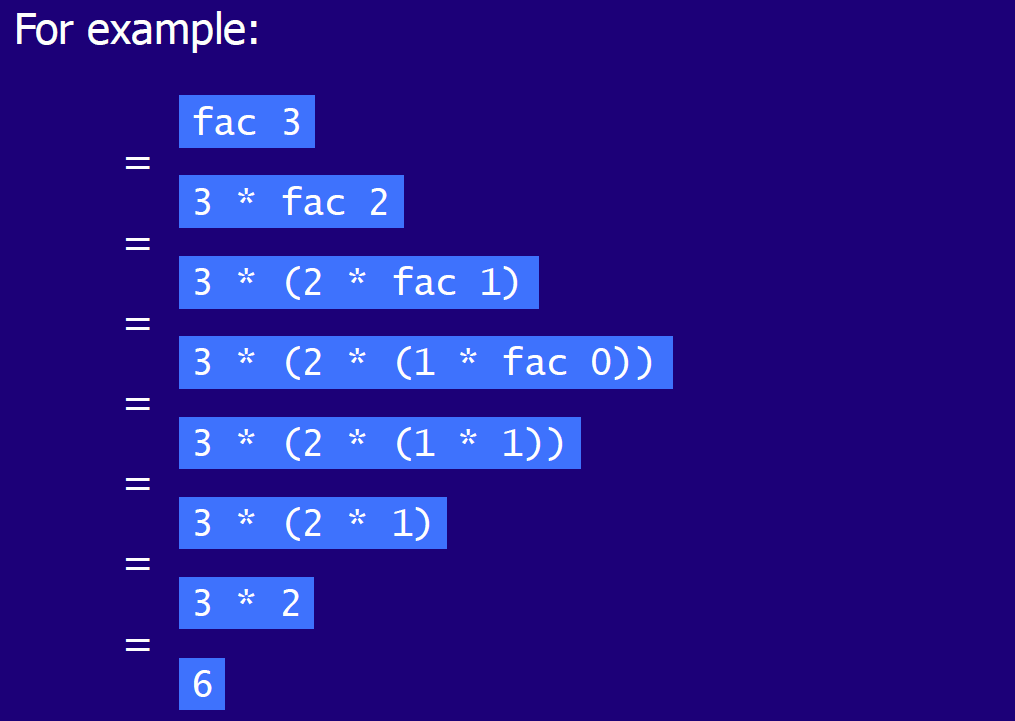
## Introduction to Recursion

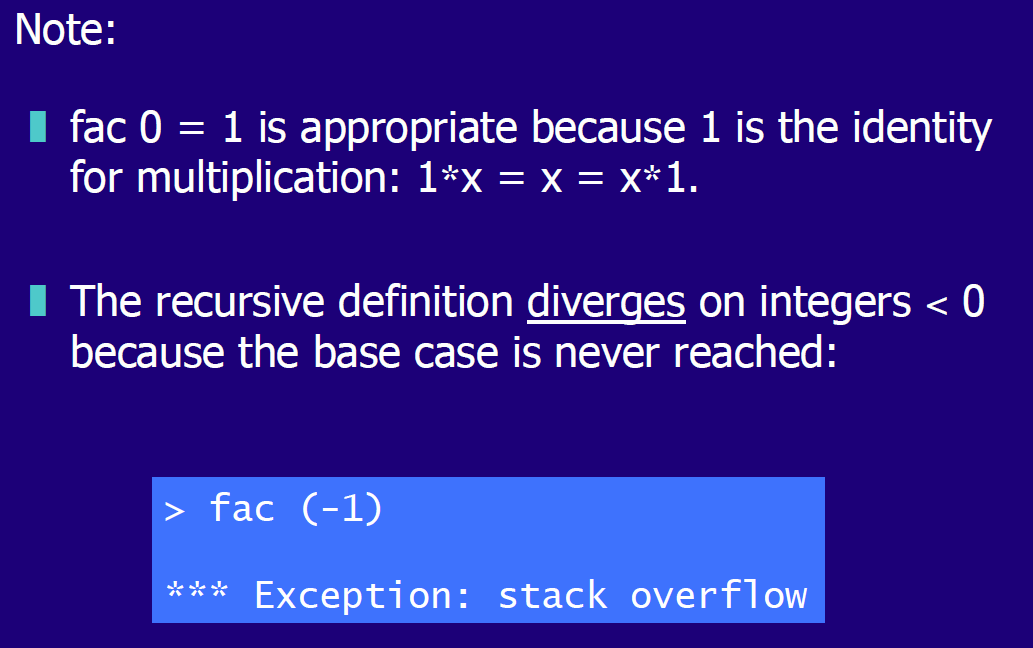




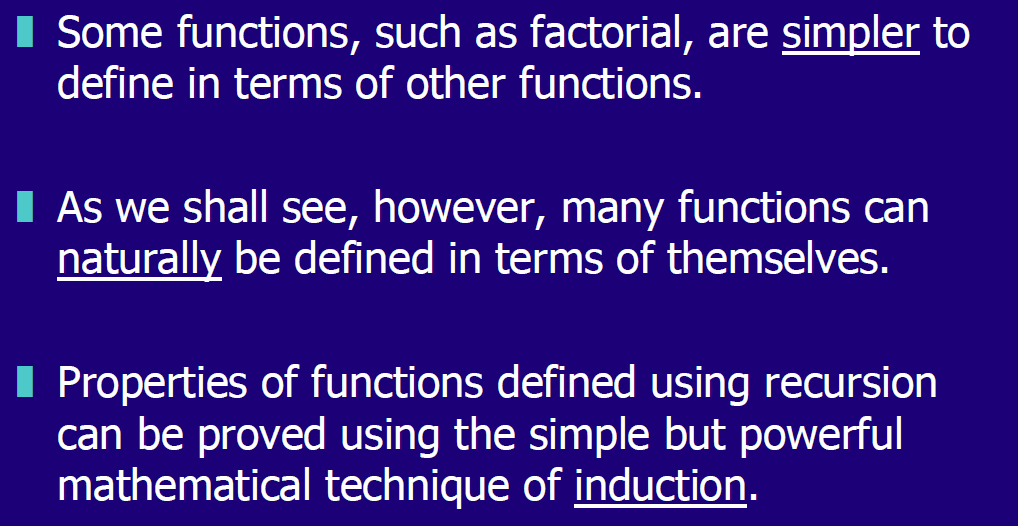
## Recursive Functions



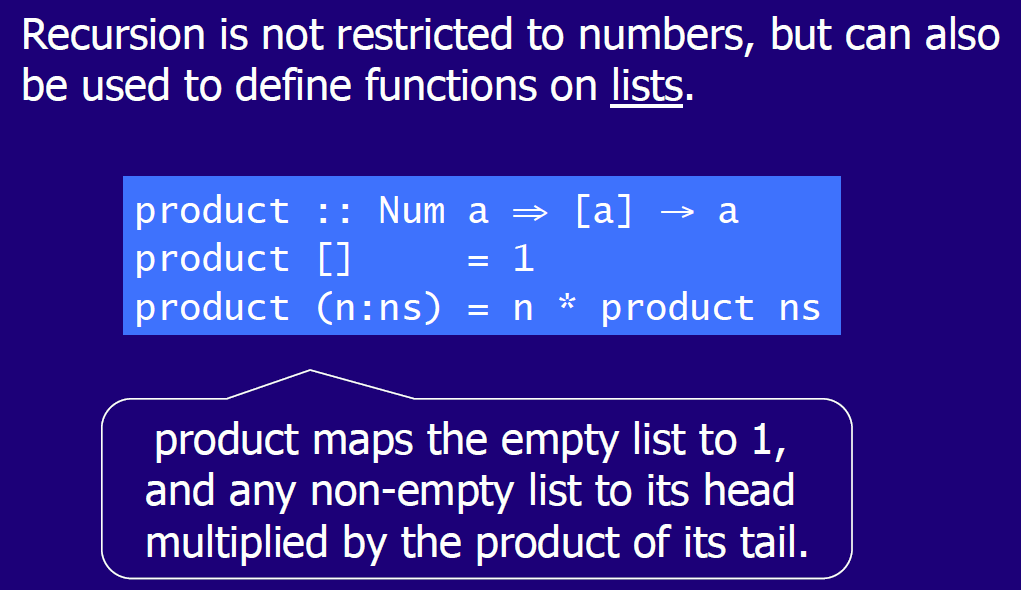


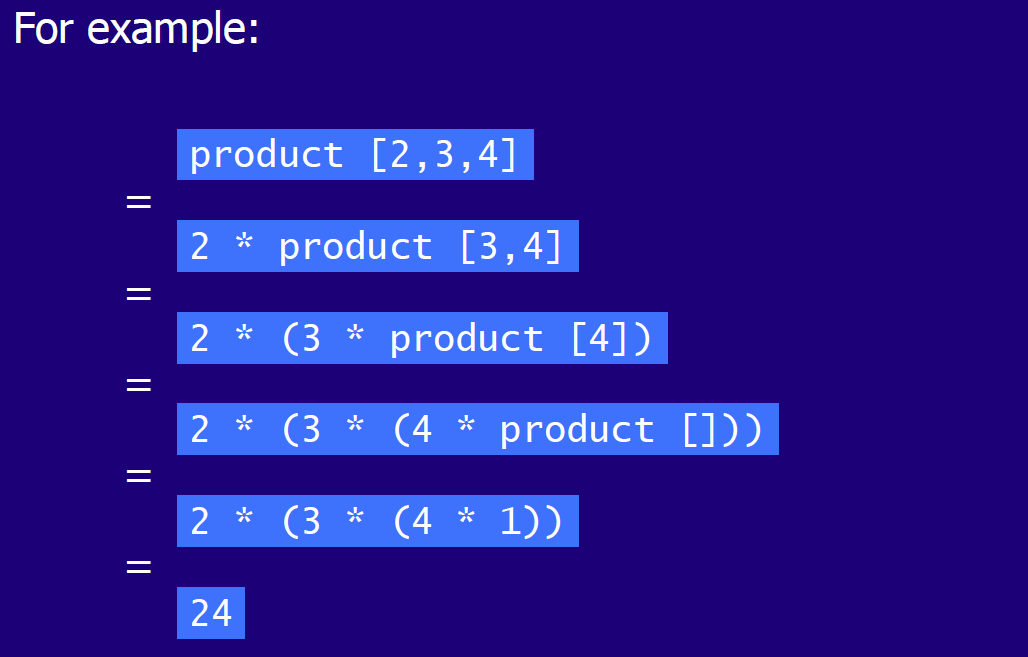


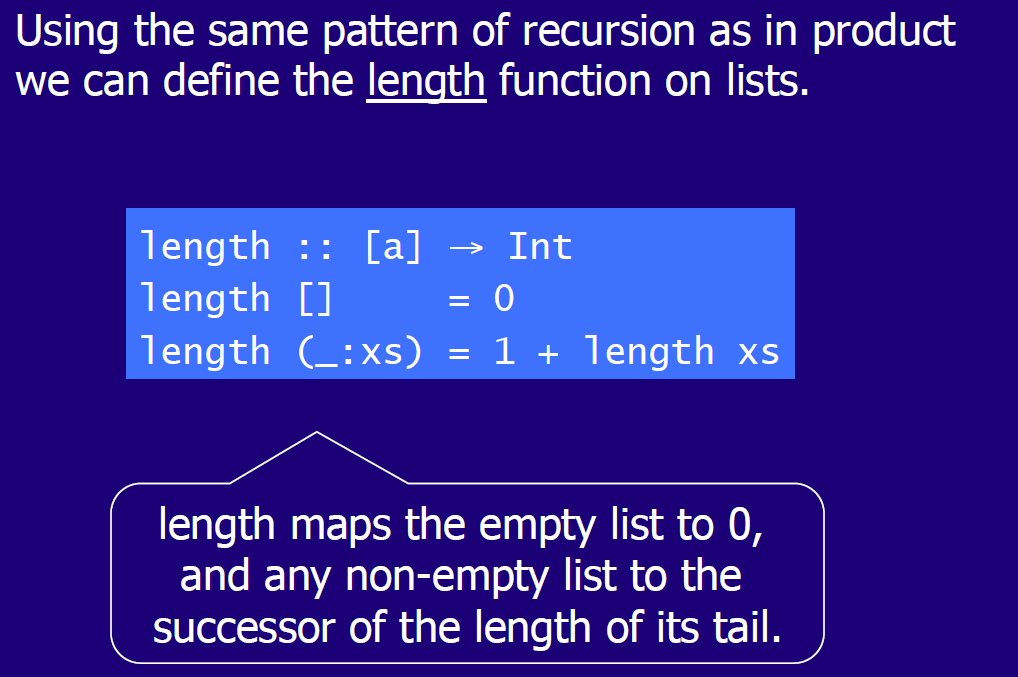
## Why is Recursion useful

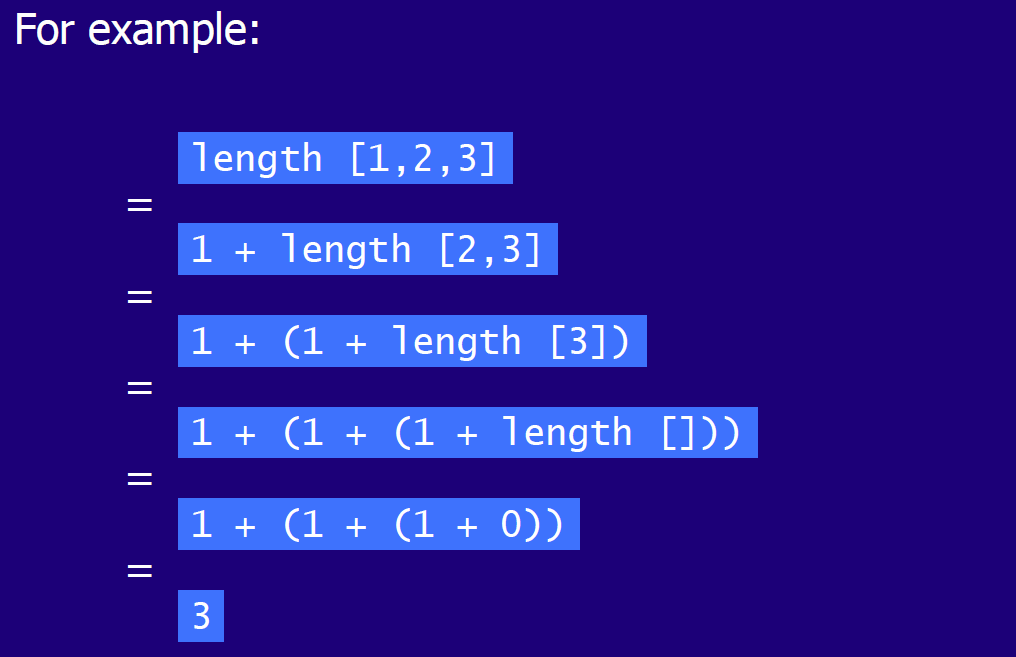


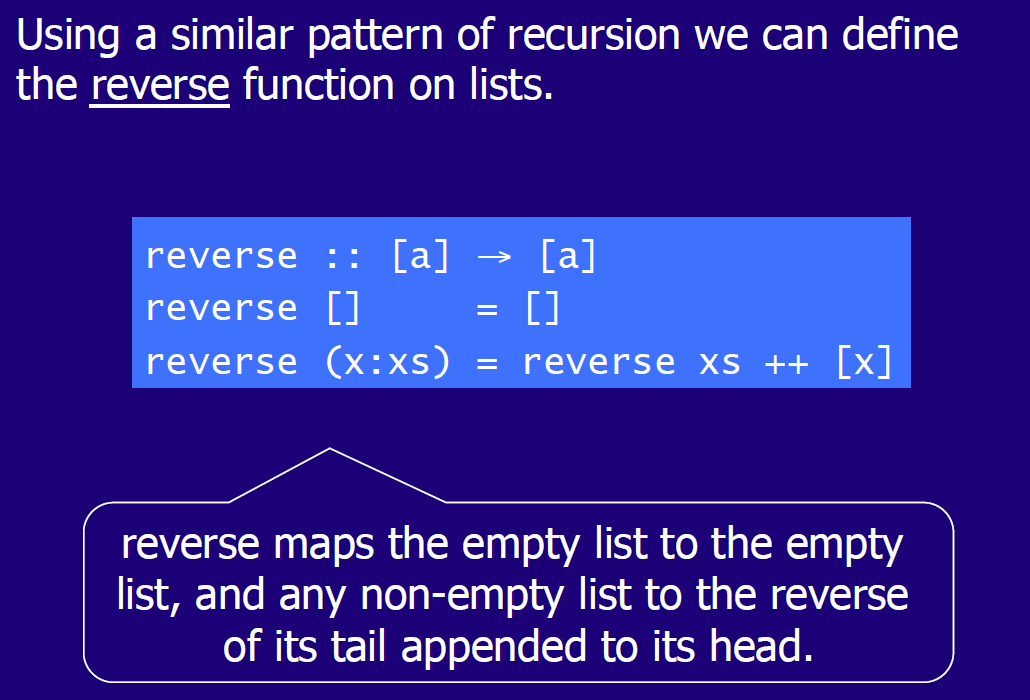
## Recursion on Lists

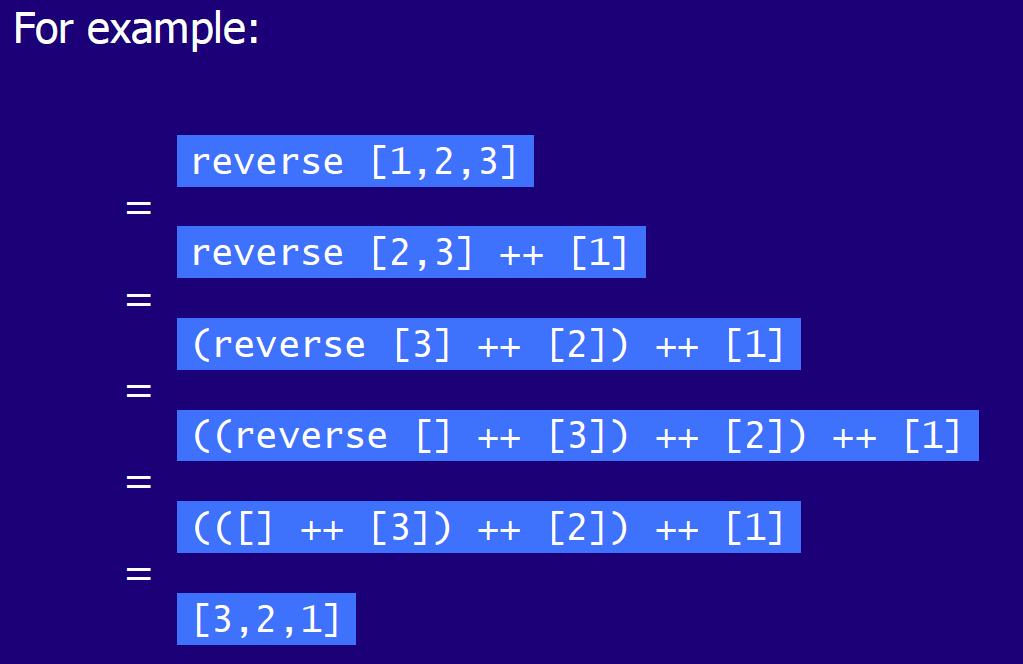




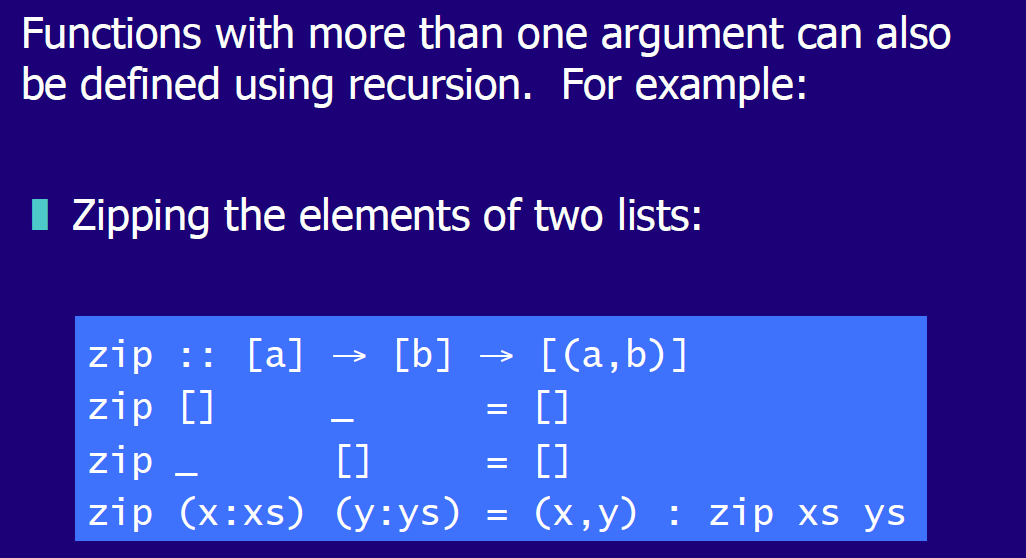


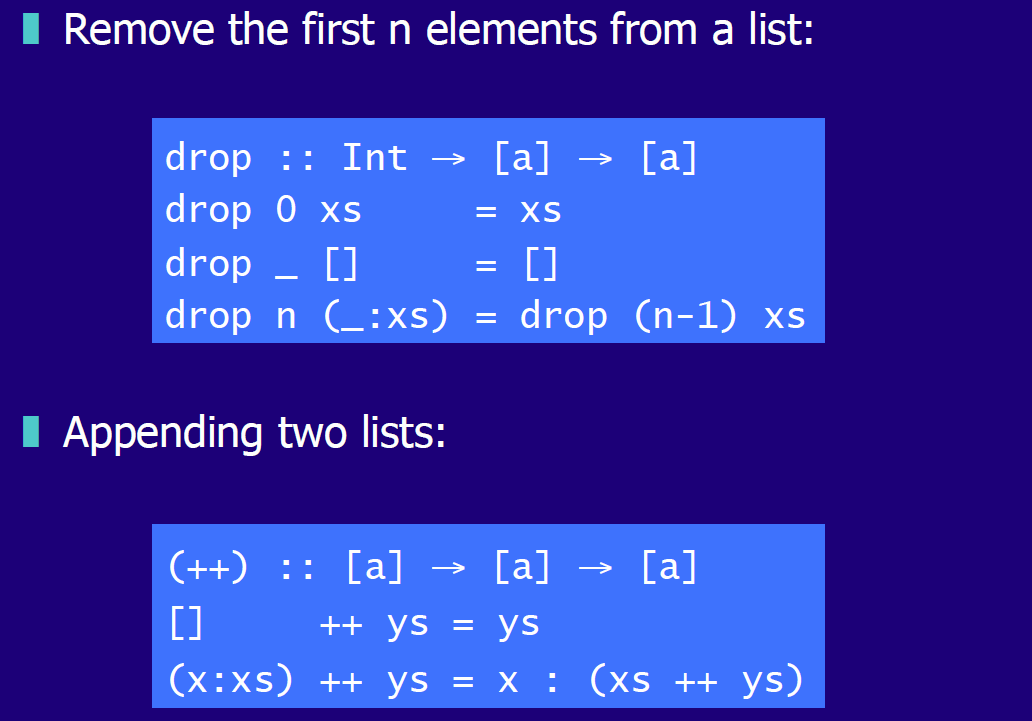




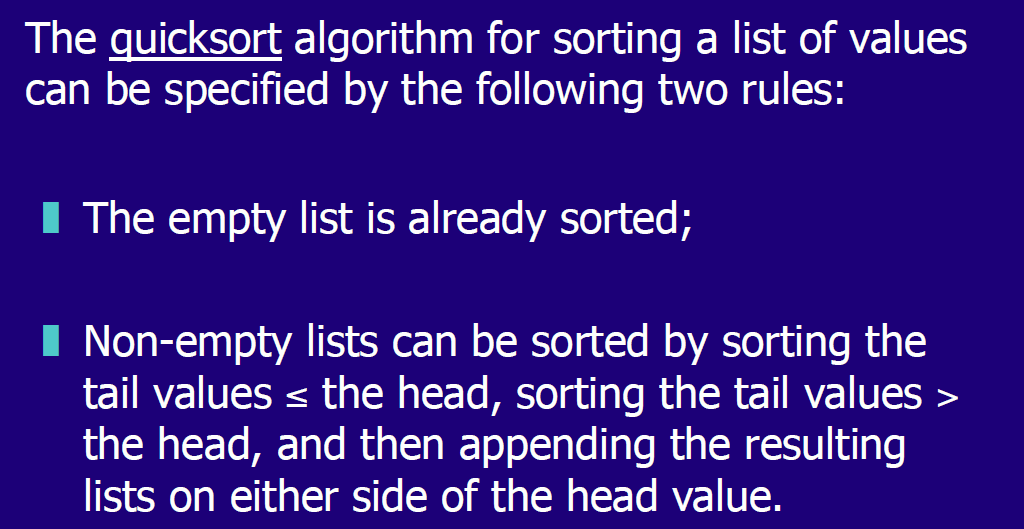


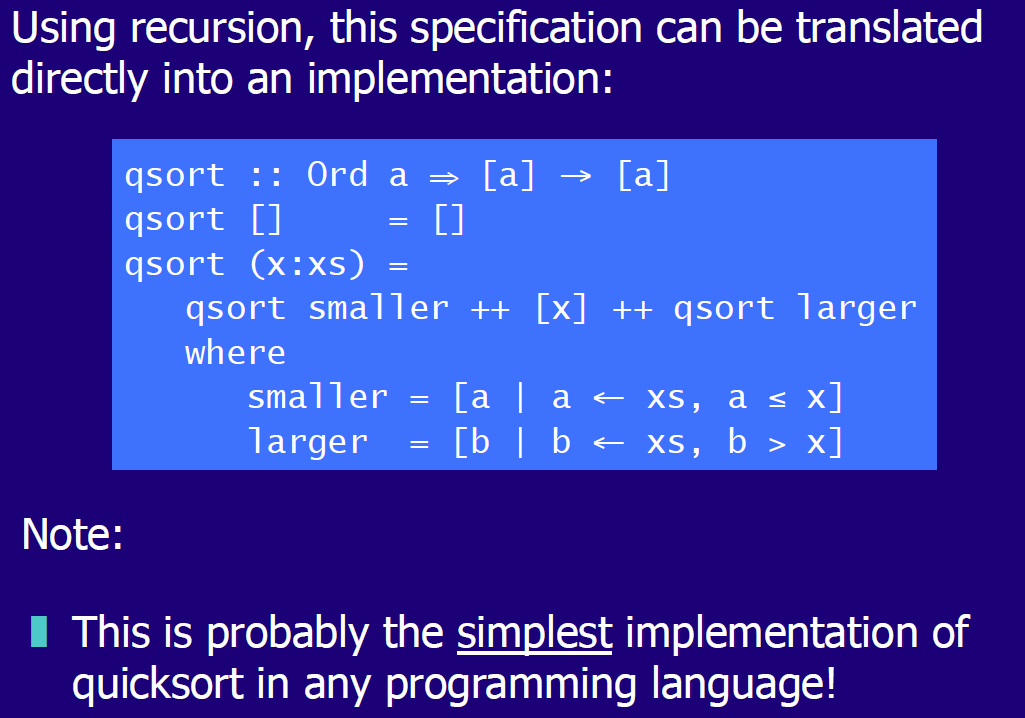
## Multiple Arguments

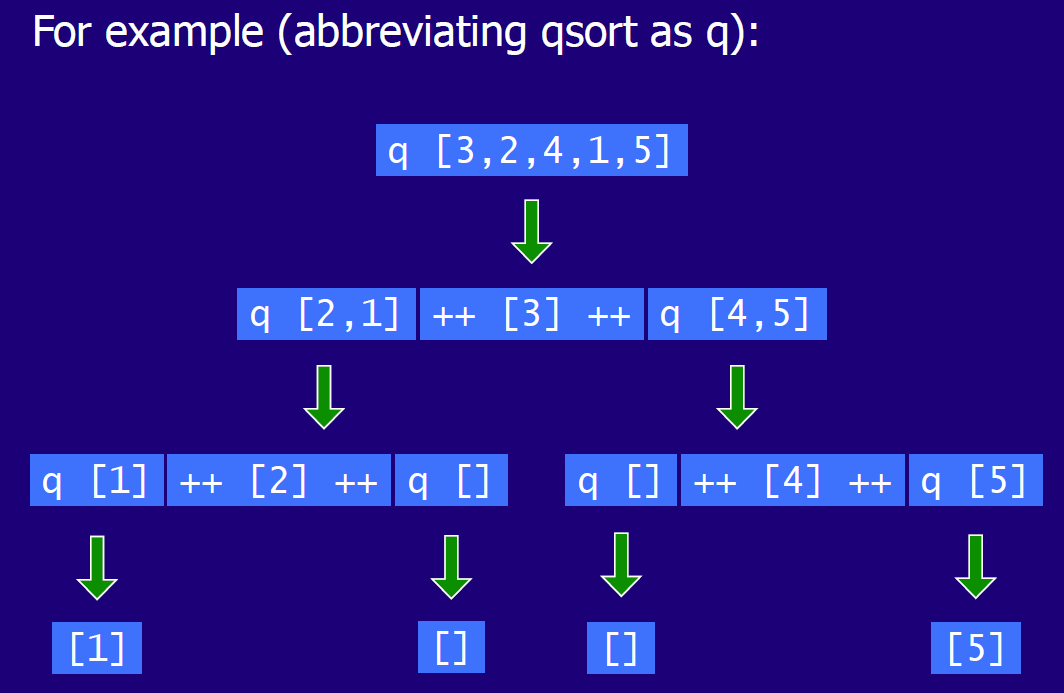




## Quicksort





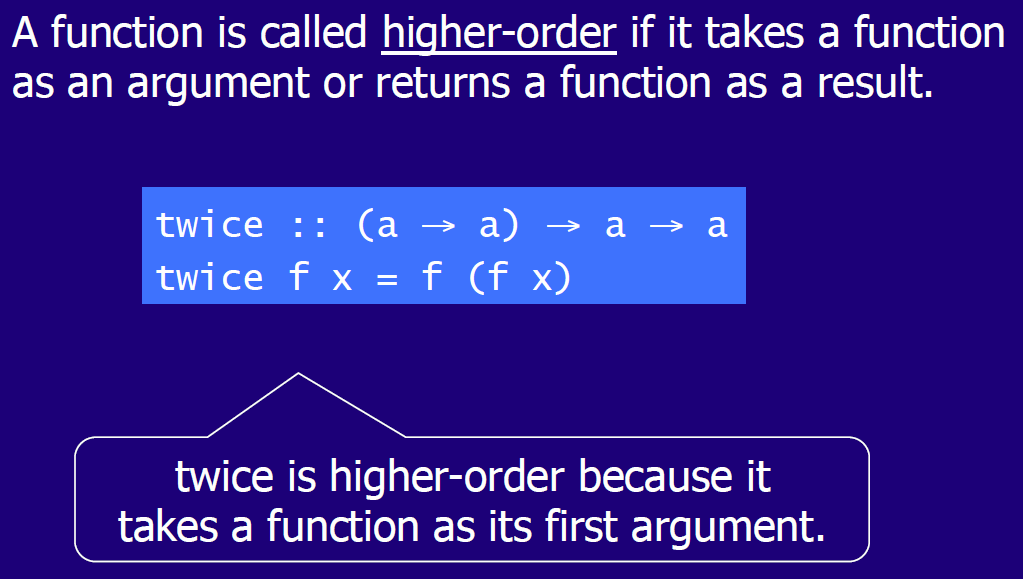


# Woche 11

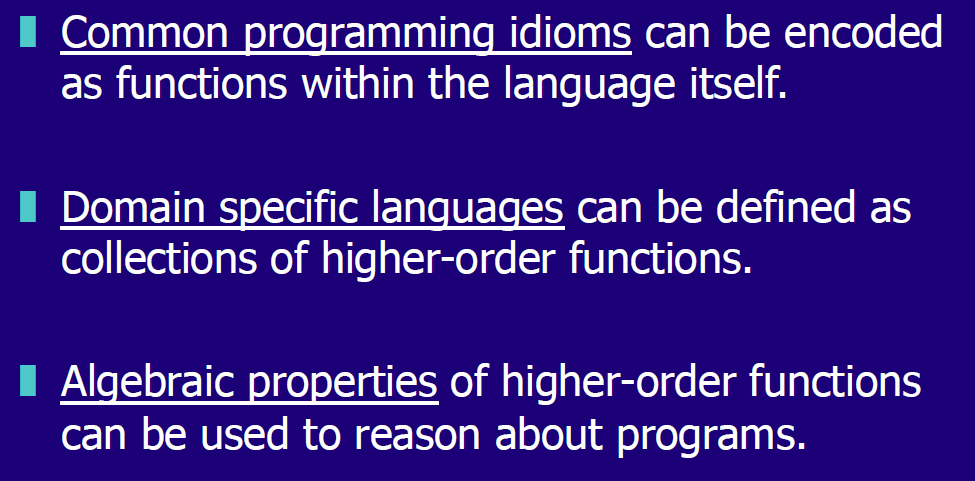
In Woche 11 fand aufgrund der Projektwoche kein Unterricht statt.

# Woche 12 & 13 & 14

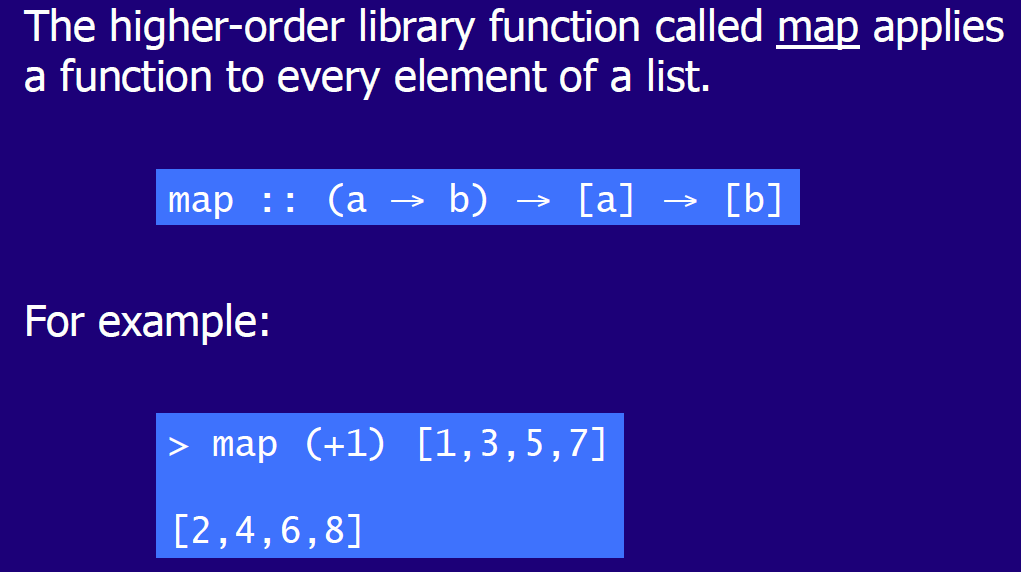
## Introduction to Higher Order Functions



## Why are they useful

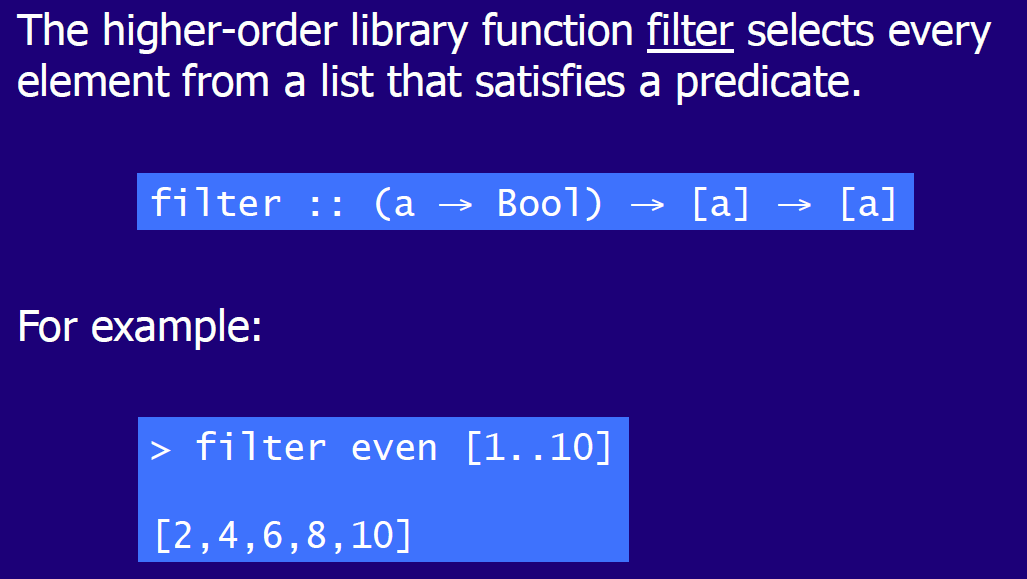


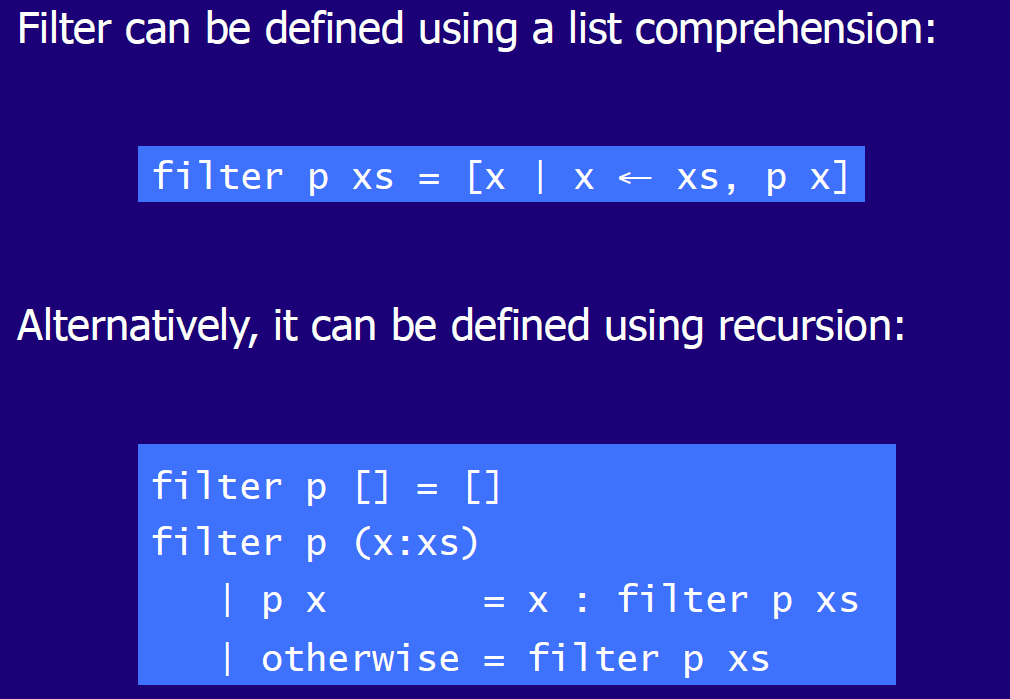
## Map Function



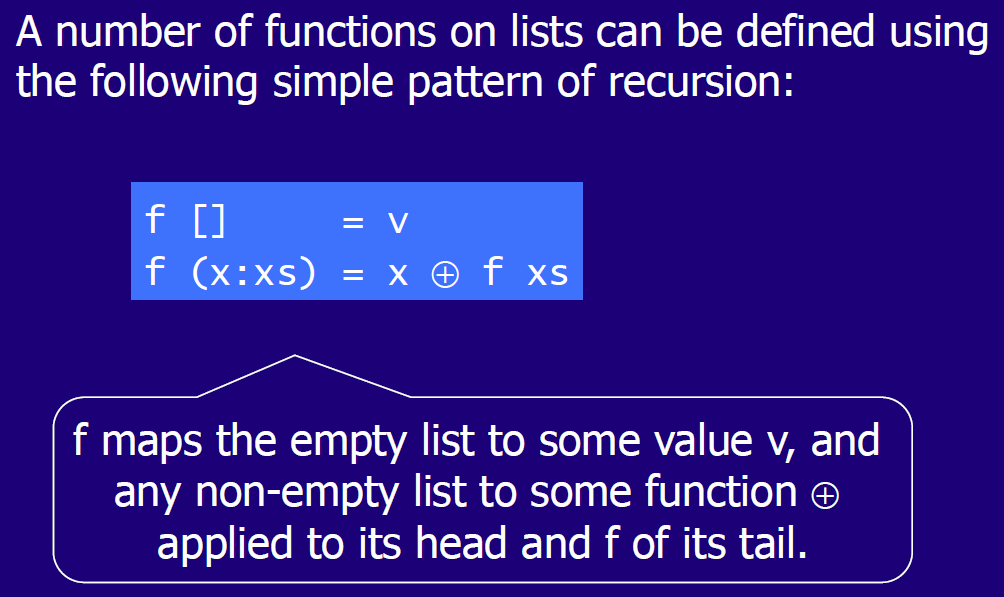


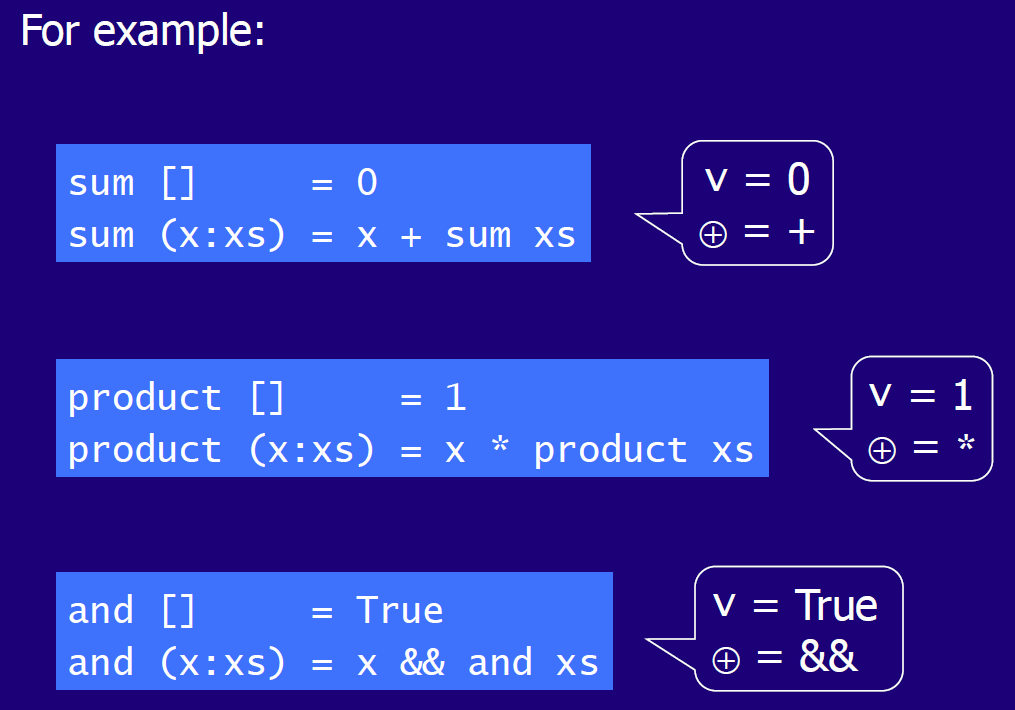
## Filter Function

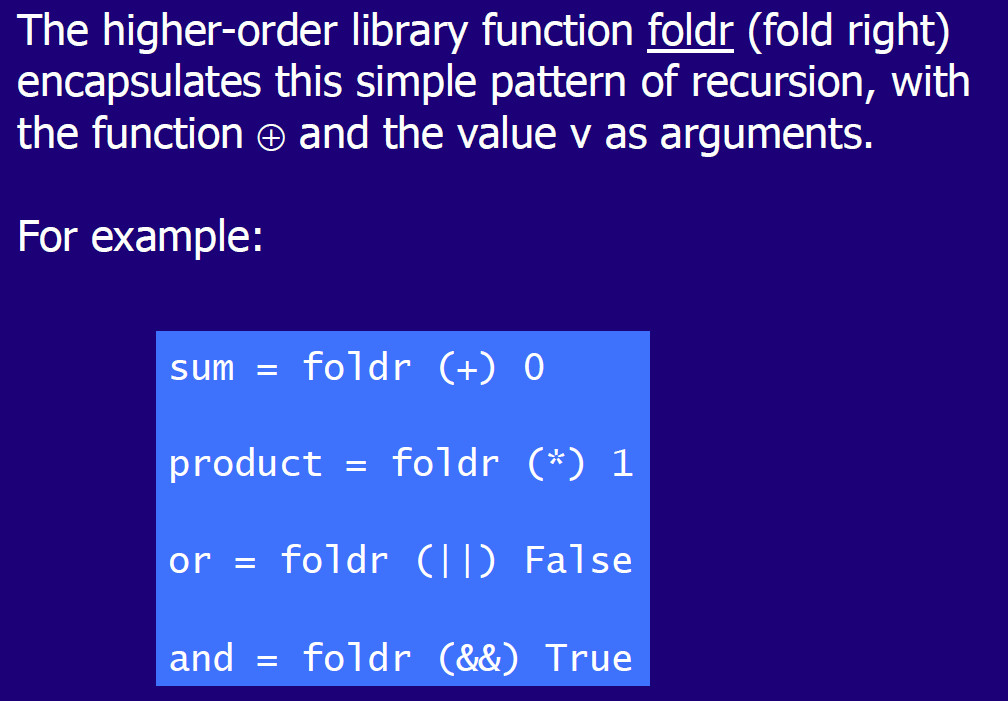


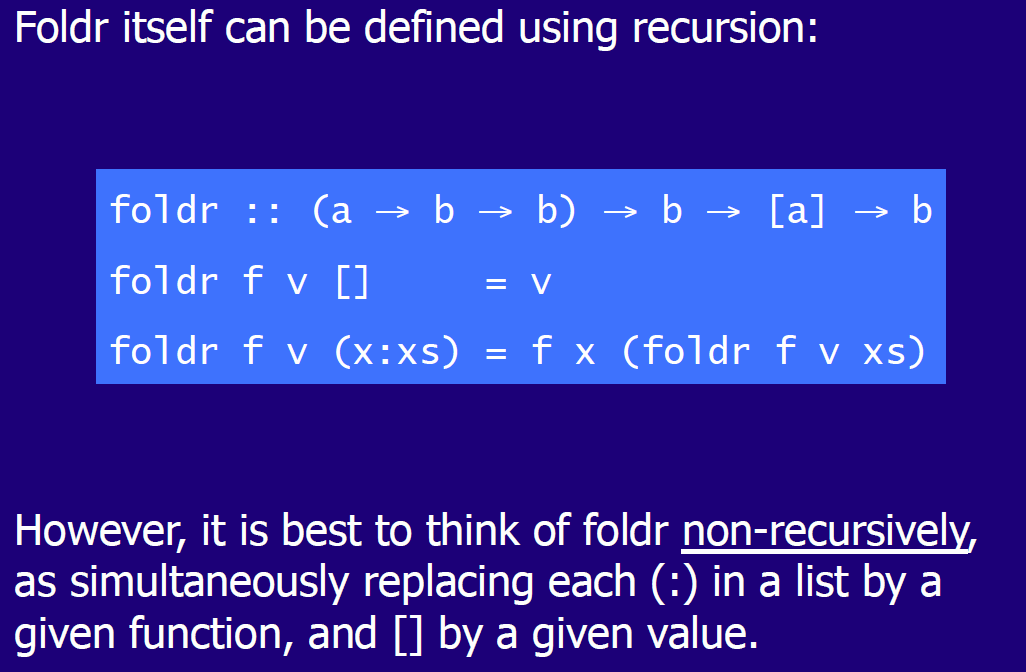


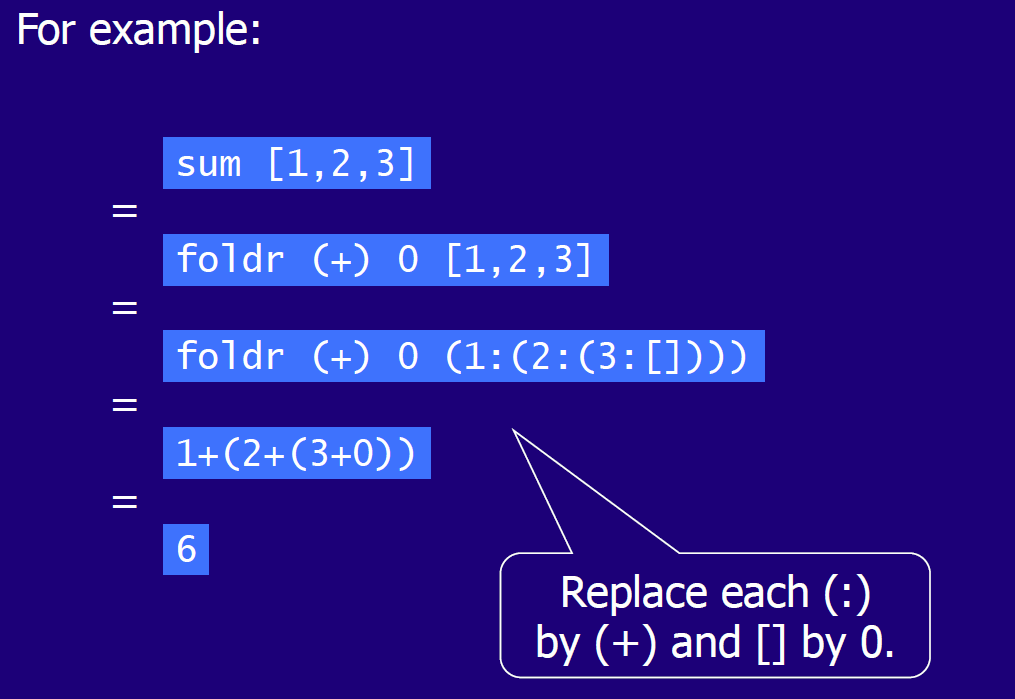
## Foldr Function

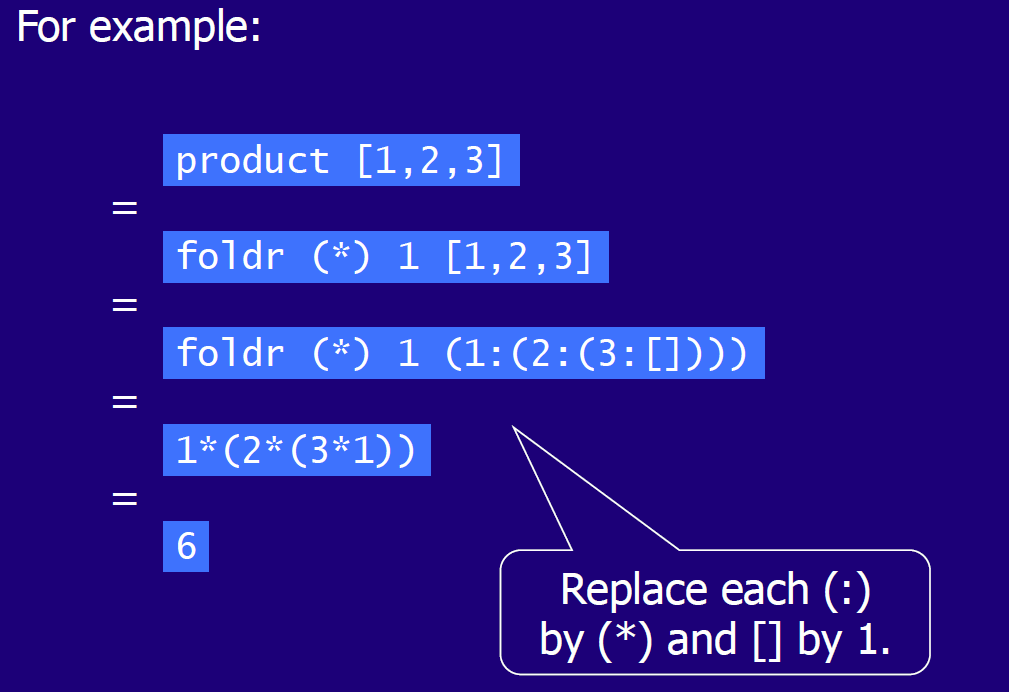




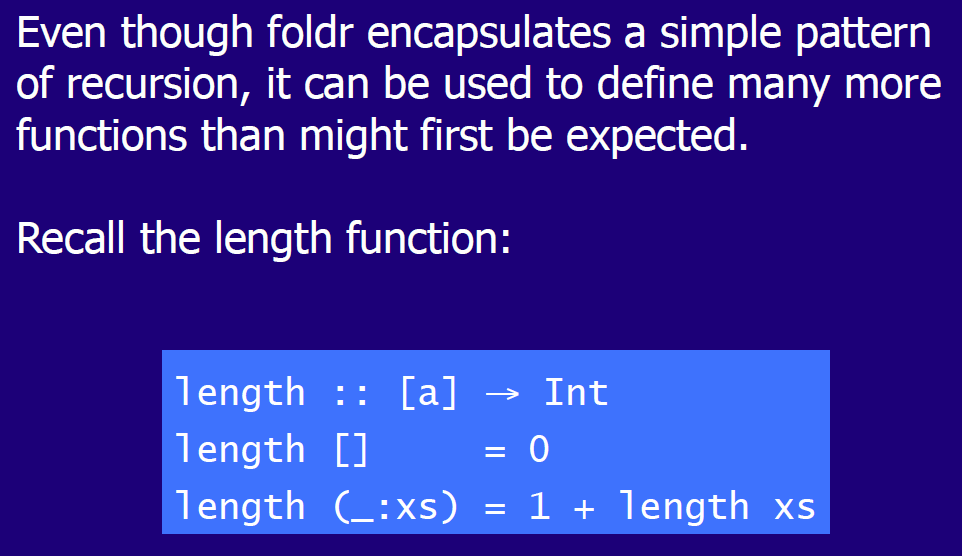


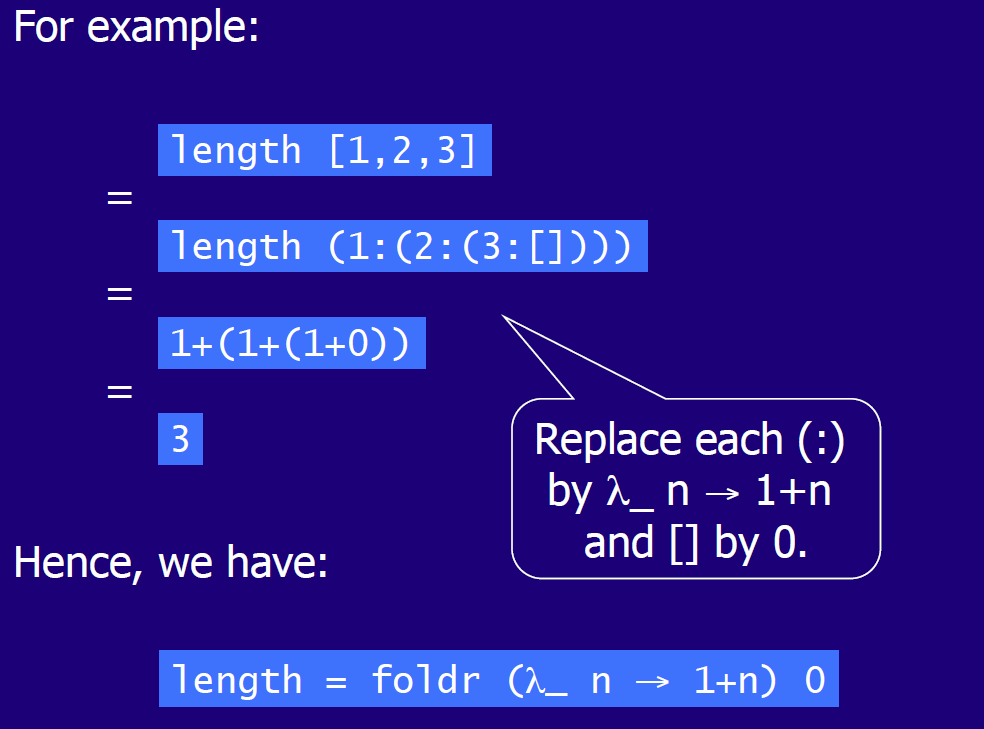


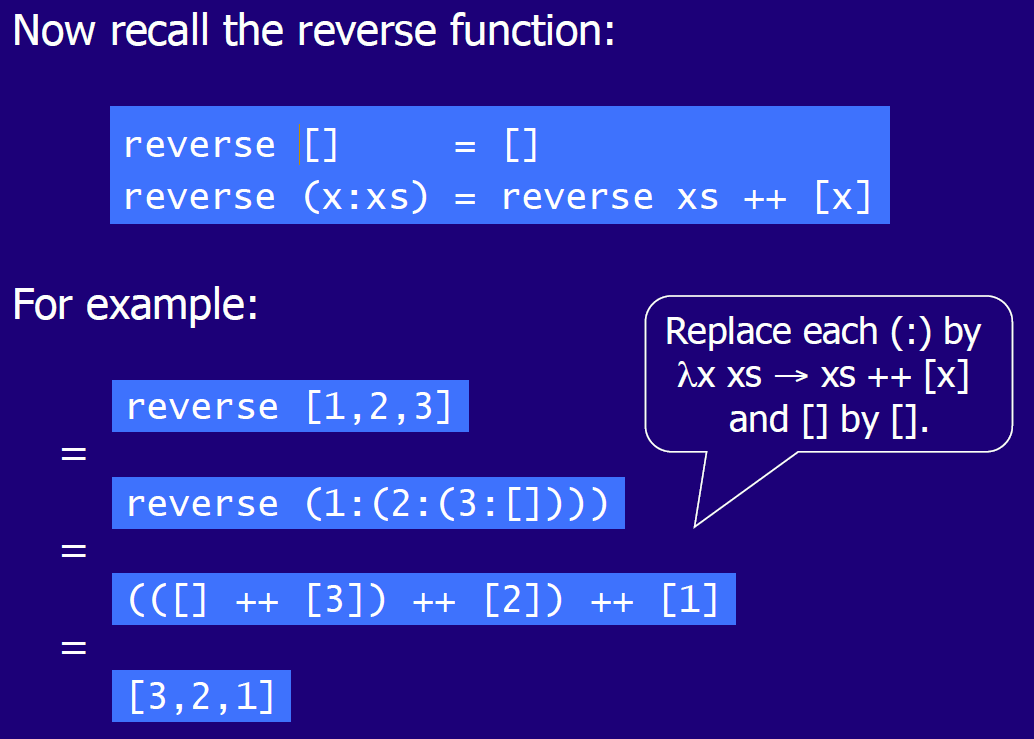


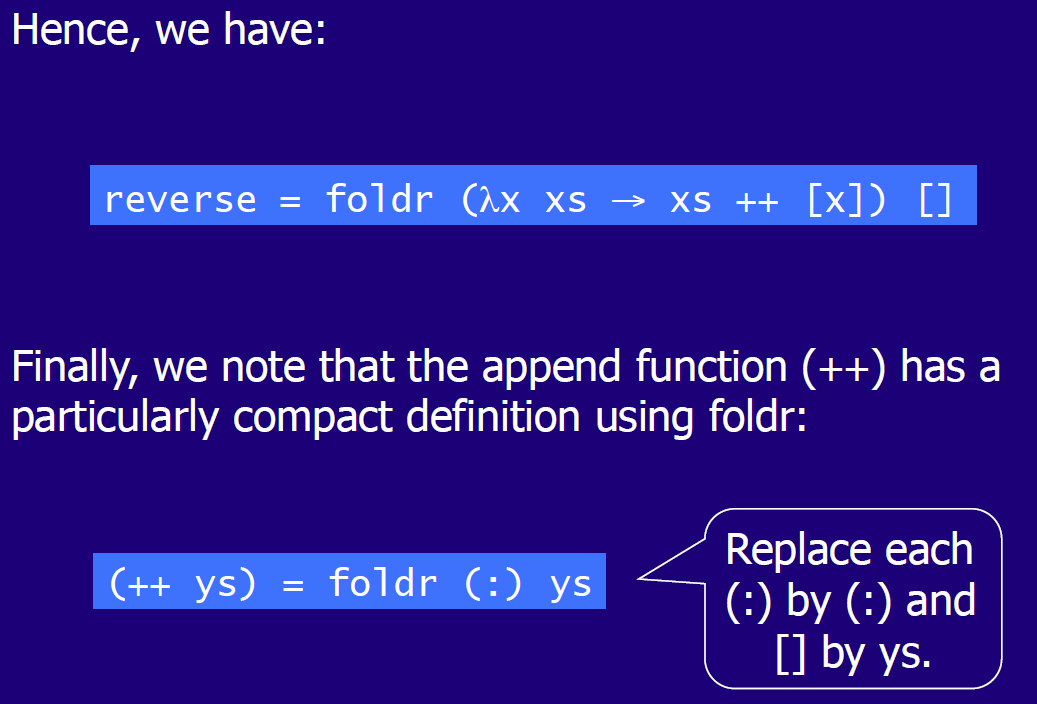


## Other Foldr Examples

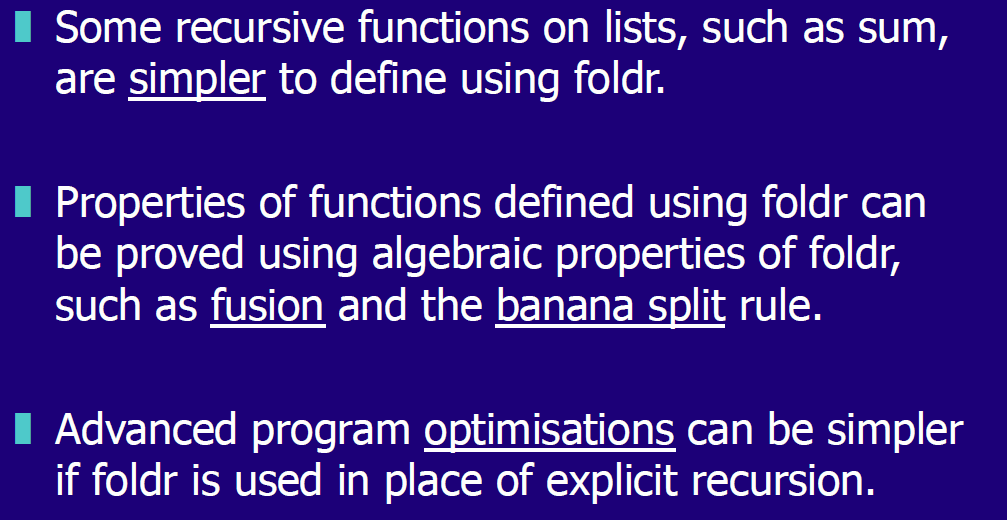




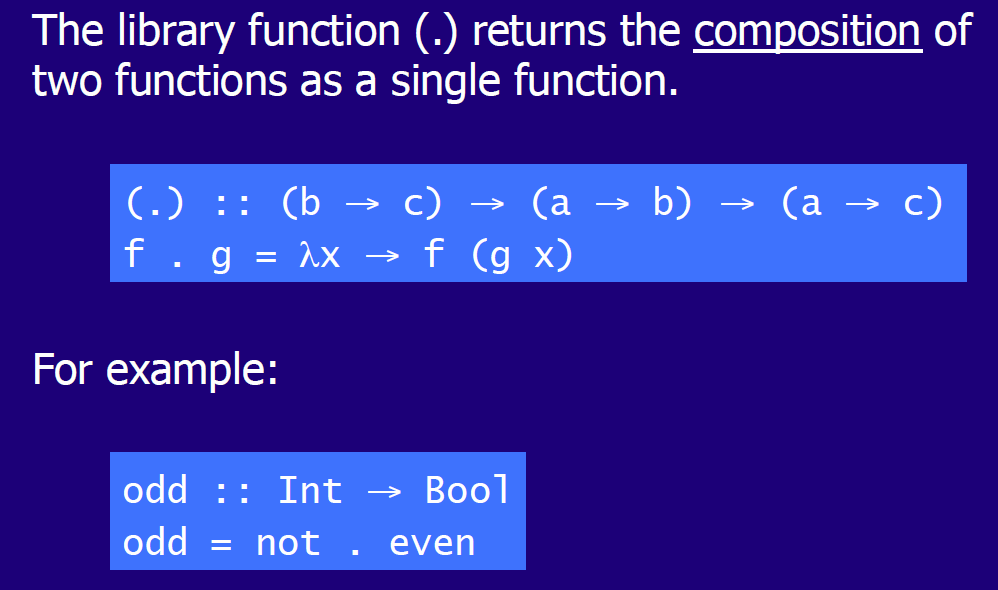


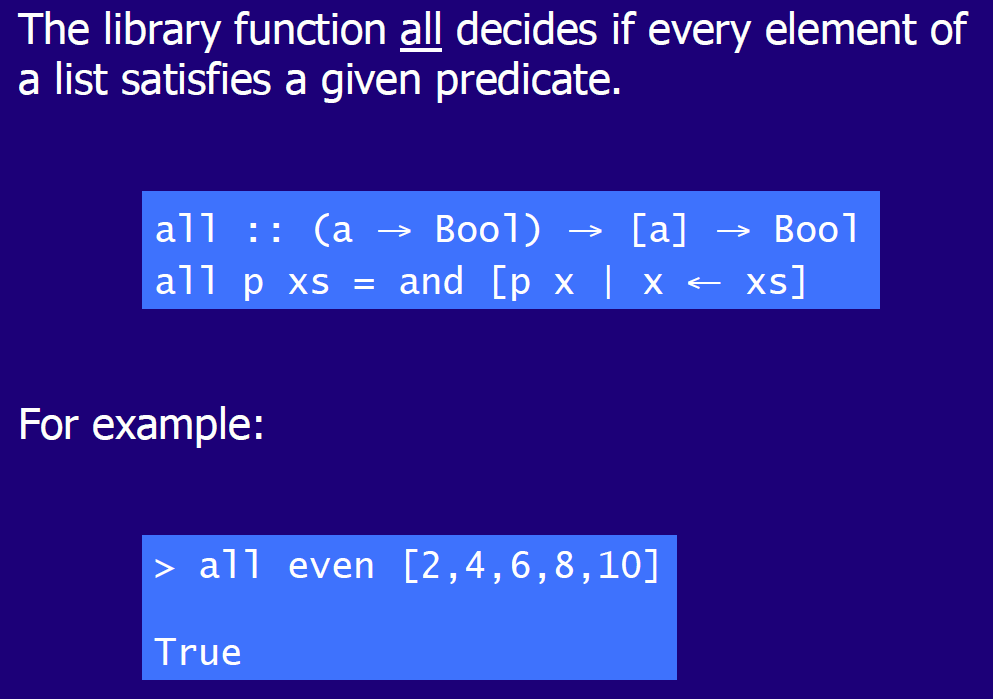


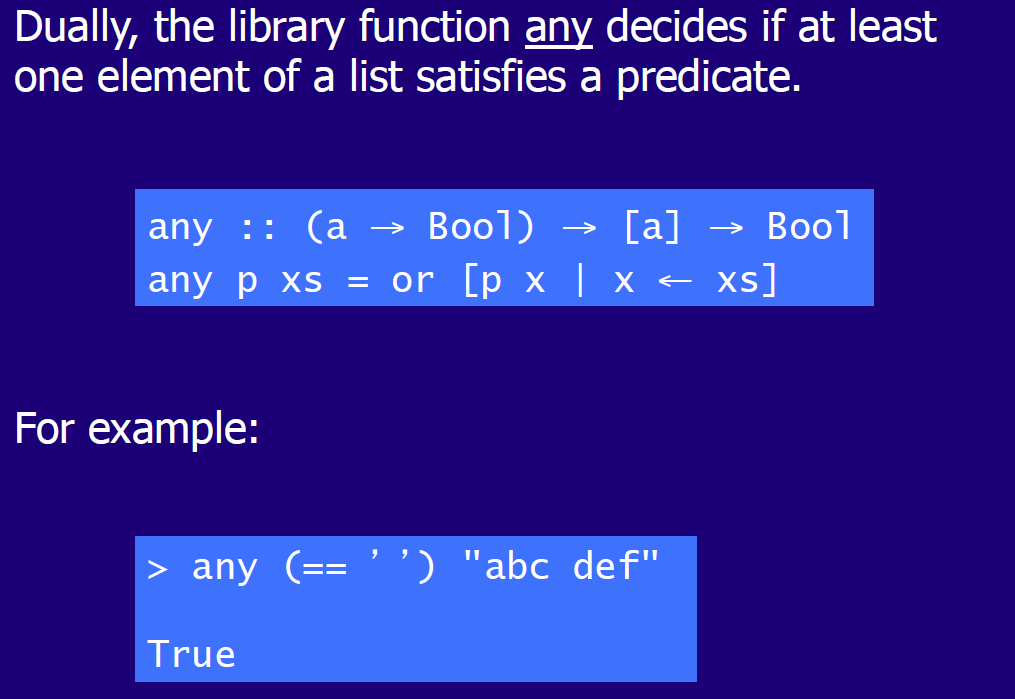
## Why is Foldr useful

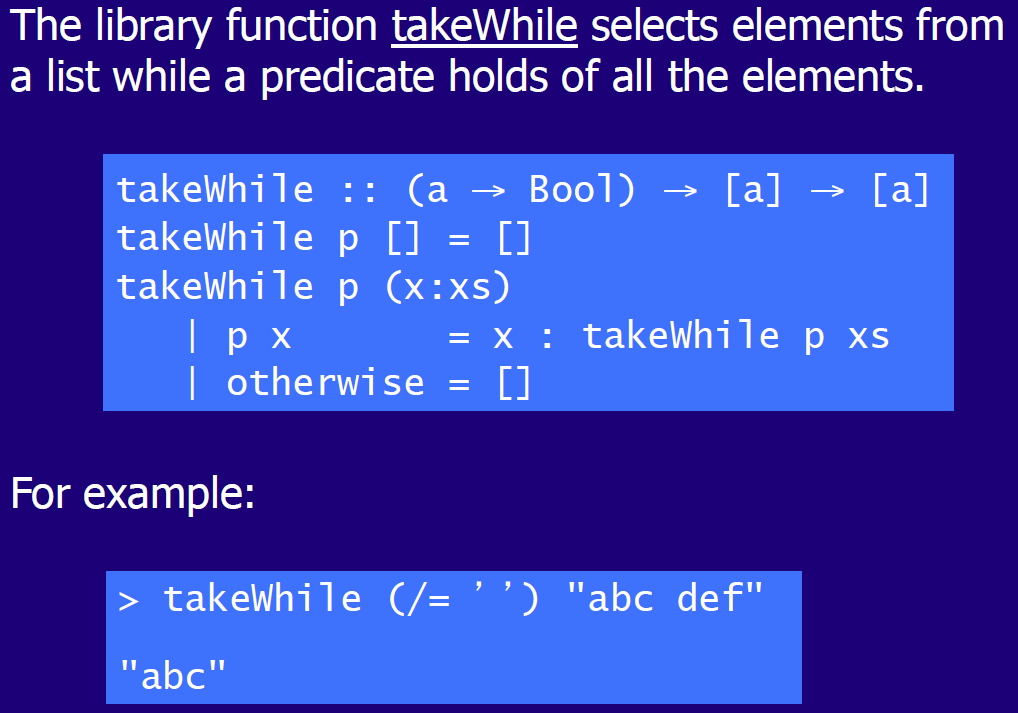


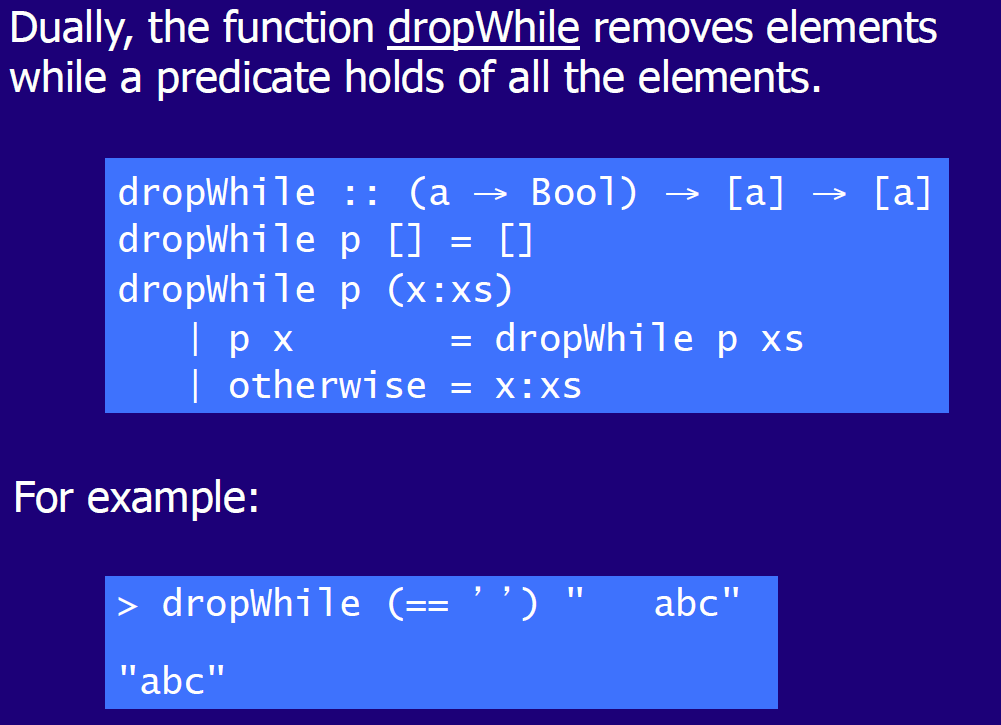
## Other Library Functions





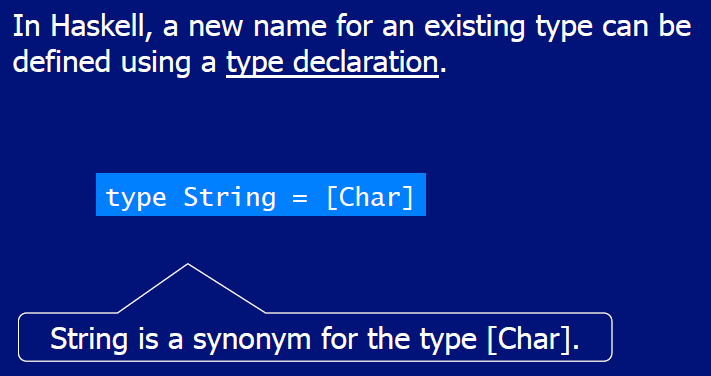


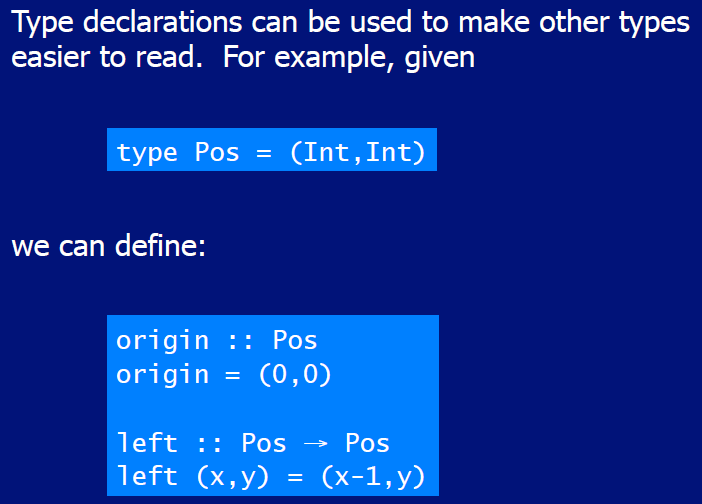


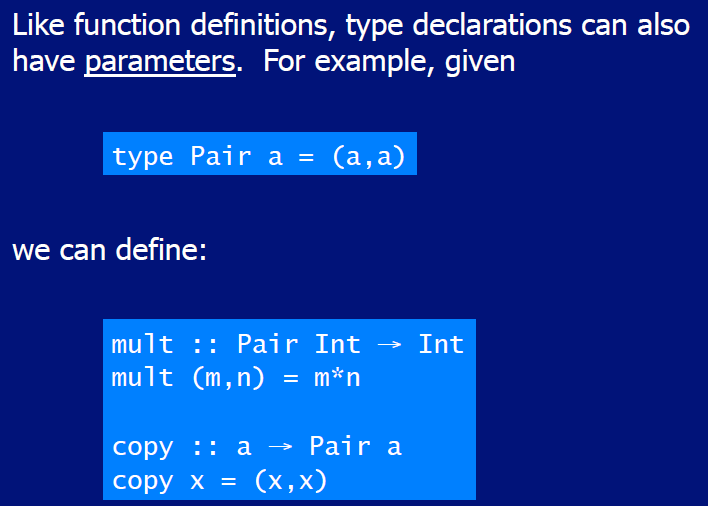


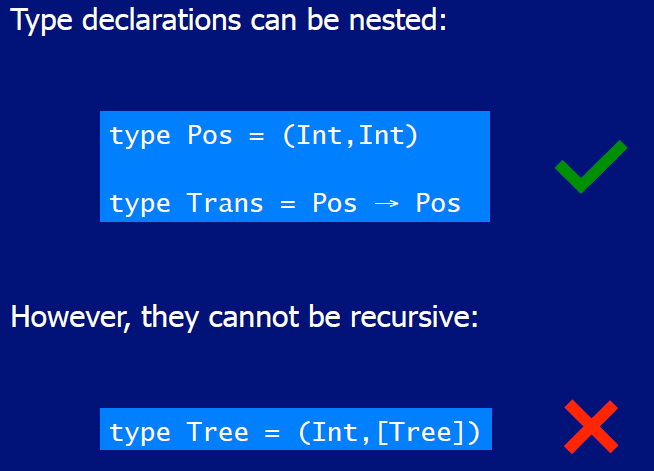
# Woche 15

## Type Declarations

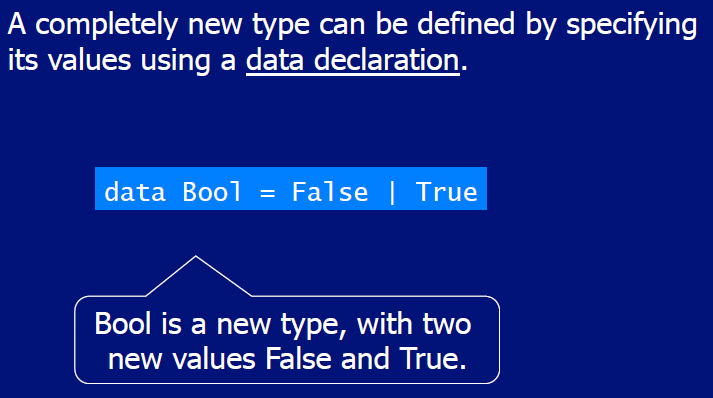


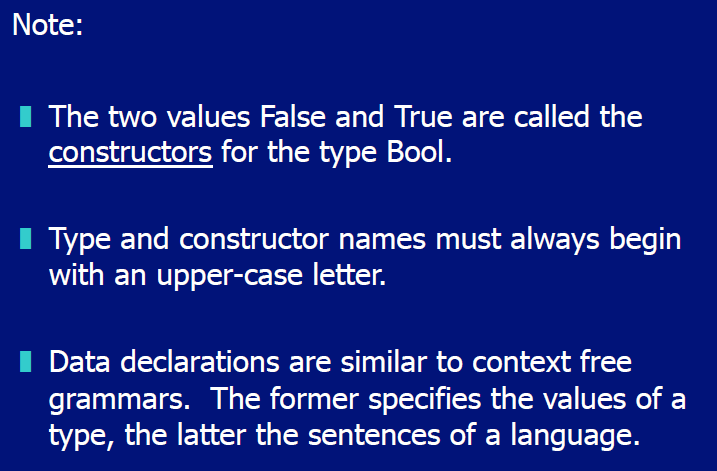




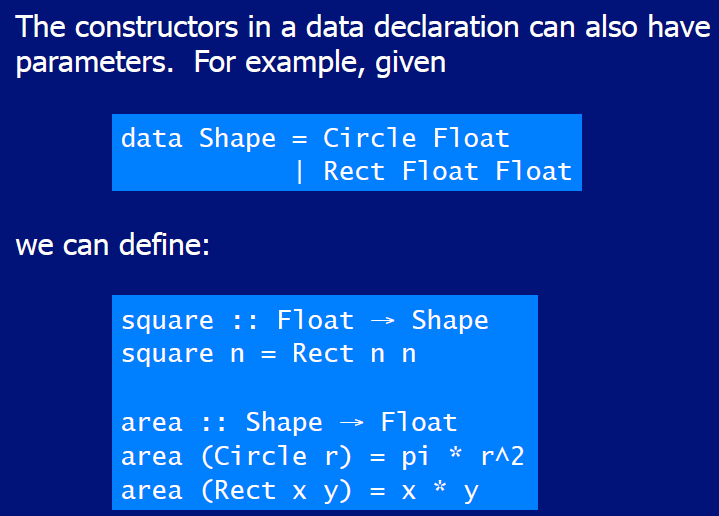


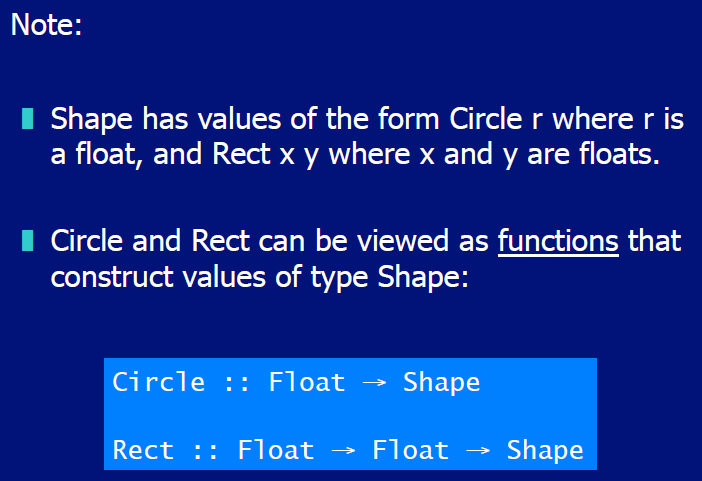
## Data Declarations

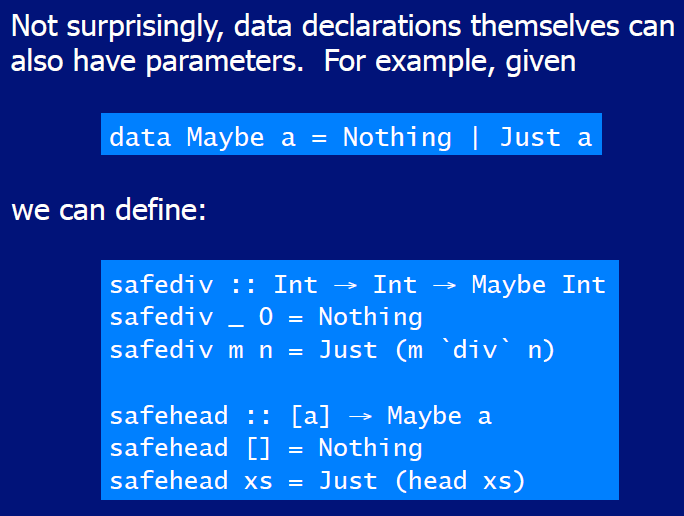




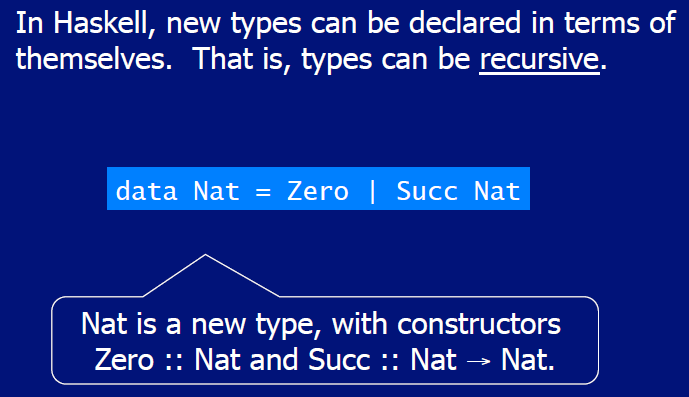


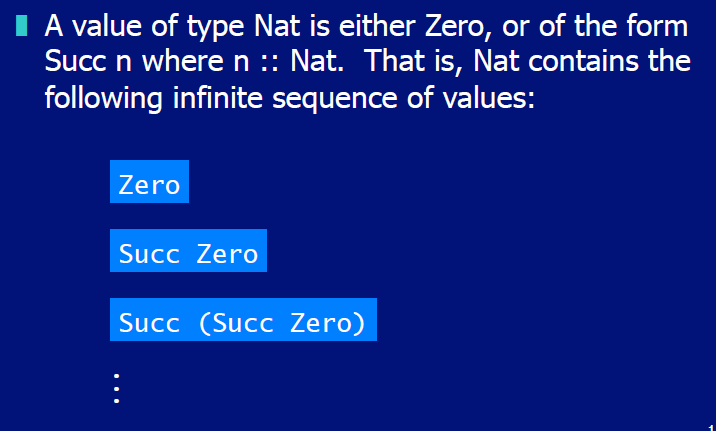


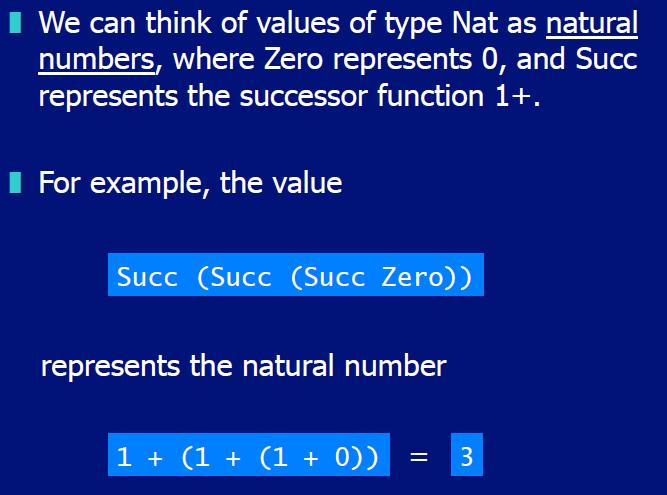


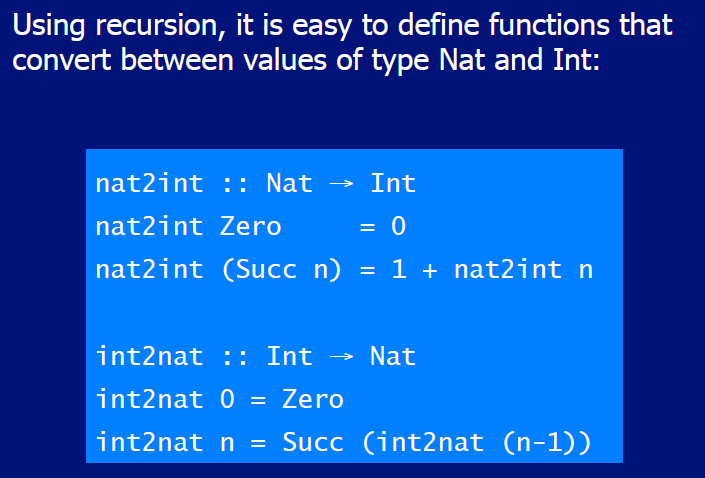


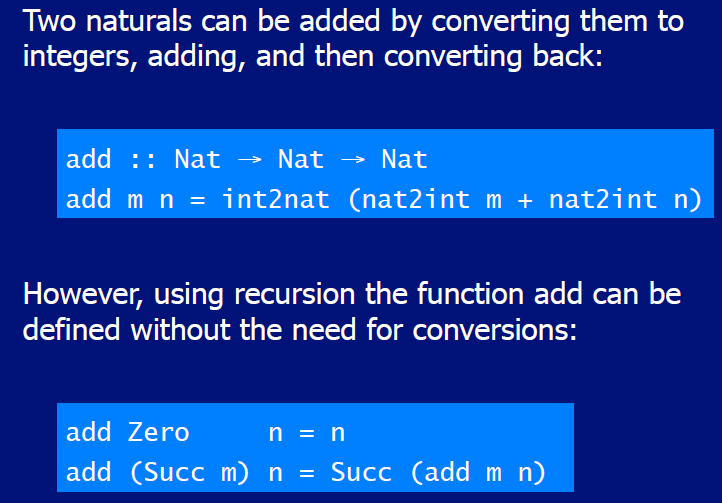
## Recursive Types

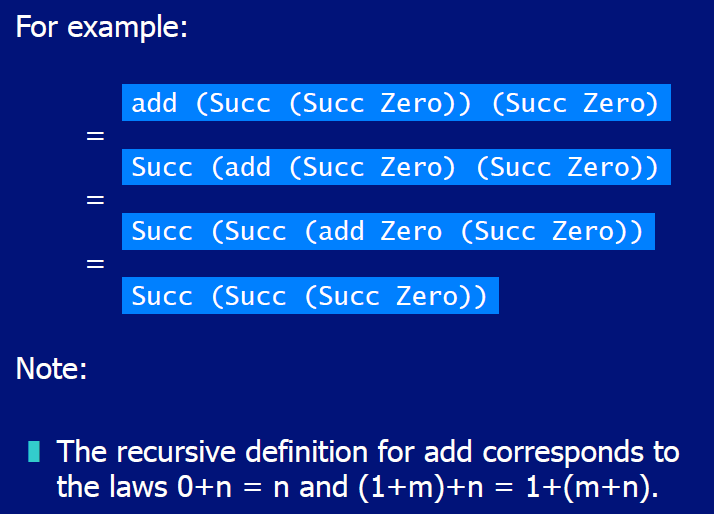




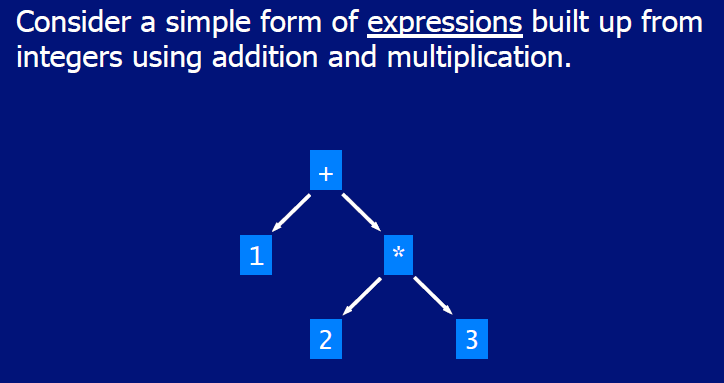


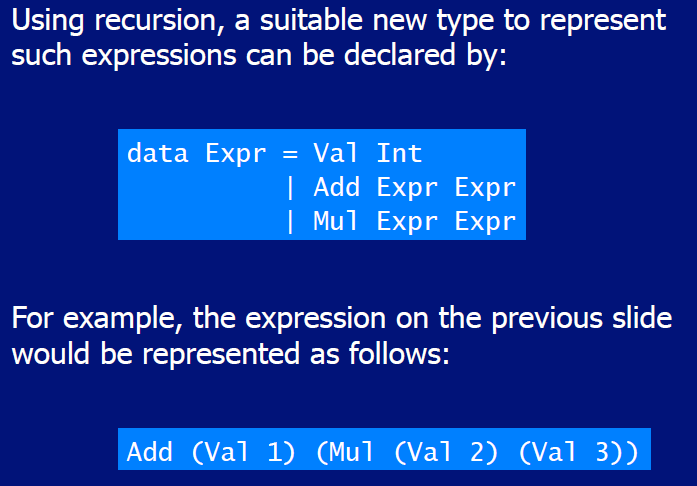


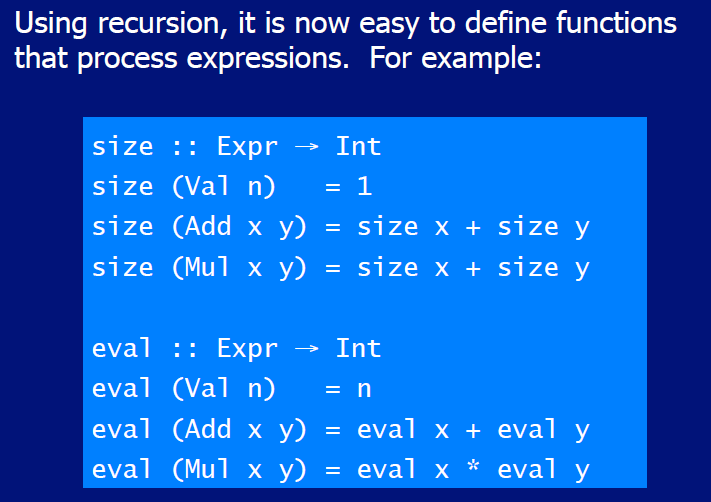




## Arithmetic Expressions

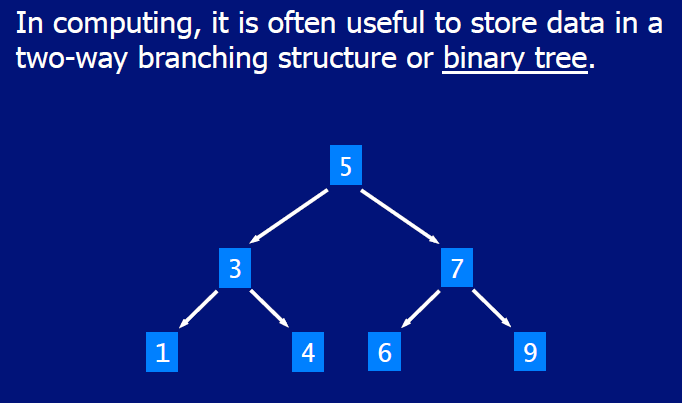


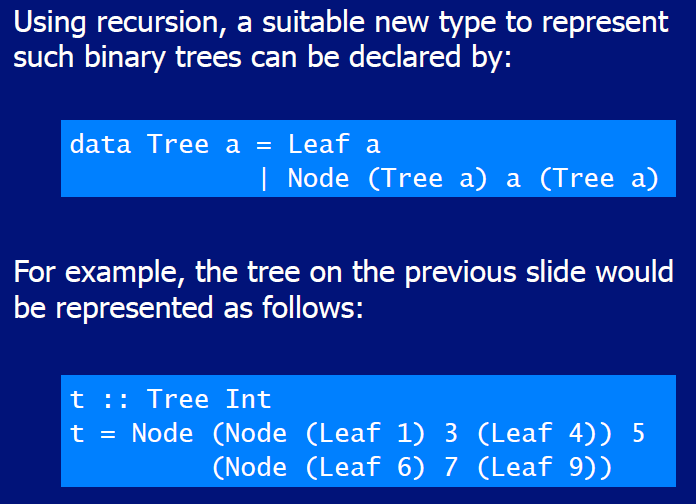


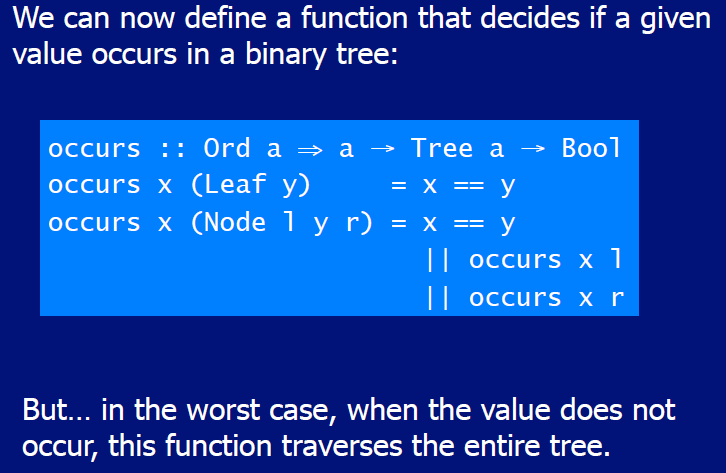


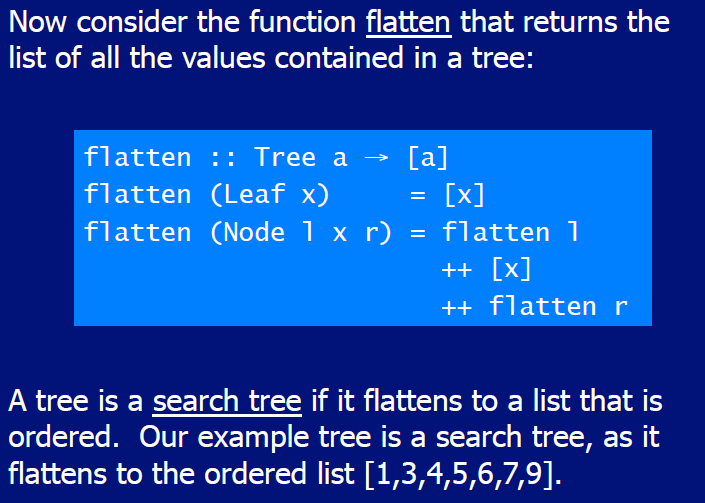


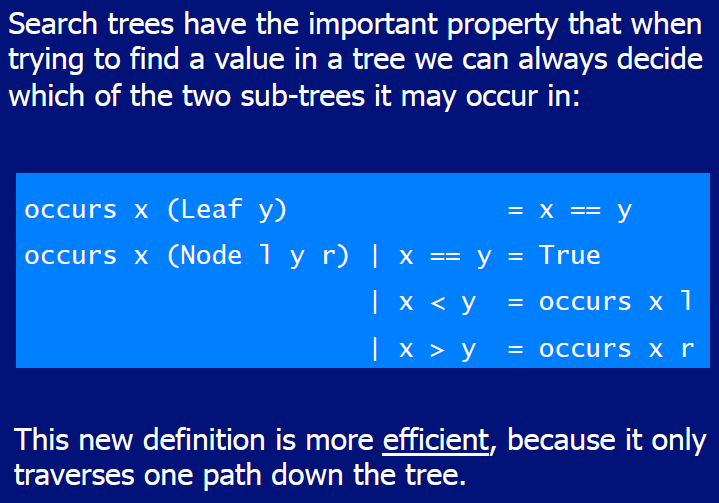
## Binary Trees











# Woche 16

In Woche 16 fand die zweite Prüfung statt und somit fiel der Unterricht aus.