

---

## Hinweise in eigener Sache...

- *Anmeldesystem ist freigeschaltet!*  
<http://www.complang.tuwien.ac.at/anmeldung>  
...Anmeldungen sind bis zum 23.10.2006 möglich!  
(Abmeldungen sind bis zum 31.10.2006 möglich, ebenfalls über das elektronische Anmeldesystem.)
- *Erstes Aufgabenblatt*  
Ausgabe: Montag, den 16.10.2006  
Abgabe: Montag, den 23.10.2006
- *Für Ihre automatische Benachrichtung per Email...*  
...über Ergebnisse zu den Übungsaufgaben: Richten Sie sich bitte eine Nachrichtenweiterleitung unter Ihrer fp-Kennung zu Ihrem bevorzugten elektronischen Postfach ein!
- *Partnerbörse*  
...am Ende der Vorlesung für Teilnehmer, die noch auf Partnersuche für die Gruppenbildung sind.

---

## Heutiges Thema

- *Teil 1: Einführung und Motivation*
  - Funktionale Programmierung: Warum? Warum mit Haskell?
  - Erste Schritte in Haskell, erste Schritte mit Hugs
- *Teil 2: Grundlagen*
  - Elementare Datentypen
  - Tupel, Listen und Funktionen

*Beachte:* ...einige Begriffe werden heute im Vorgriff angerissen und erst im Lauf der Vorlesung genau geklärt!

---

## Teil 1: Einführung und Motivation

- Funktionale Programmierung: Warum überhaupt? Warum mit Haskell?
- Erste Schritte in Haskell, erst Schritte mit Hugs

---

## Warum funktionale Programmierung?

Ein bunter Strauß an *Programmierparadigmen*, z.B.

- *imperativ*
  - prozedural (Pascal, Modula, C,...)
  - objektorientiert (Smalltalk, Oberon, C++, Java,...)
- *deklarativ*
  - funktional (Lisp, ML, Miranda, Haskell, Gofer,...)
  - logisch (Prolog und Varianten)
- *Mischformen*  
z.B. funktional/logisch, funktional/objektorientiert,...
- *visuell*  
*Stichwort:* Visual Programming Languages (VPLs),  
z.B. Forms/3, FAR,...  
Einstieg für mehr: [web.engr.oregonstate.edu/~burnett](http://web.engr.oregonstate.edu/~burnett)
- ...

---

## Ein Vergleich - prozedural vs. funktional

Gegeben eine Aufgabe *A*.

**Prozedural:** Typischer Lösungsablauf in folgenden Schritten:

1. Beschreibe eine(n) Lösung(sweg) *L* für *A*.
2. Gieße *L* unter expliziter Organisation der Speicherverwaltung in die Form einer Menge von Anweisungen für den Rechner.

---

## Zur Illustration ein einfaches Beispiel

Betrachte folgende Aufgabe:

... "bestimme die Werte aller Komponenten eines ganzzahligen Feldes, deren Werte kleiner oder gleich 10 sind."

Eine typische Lösung *prozedural*...

```
...
j := 1;
FOR i:=1 TO maxLength DO
  IF a[i] <= 10 THEN b[j] := a[i]; j := j+1 FI
OD;
```

Mögliches Problem bei großen Anwendungen:

...inadäquates Abstraktionsniveau  $\leadsto$  *Softwarekrise!*

---

## Softwarekrise

- Ähnlich wie objektorientierte Programmierung verspricht deklarative, insbesondere funktionale Programmierung ein angemesseneres Abstraktionsniveau zur Problemlösung zur Verfügung zu stellen
- ...und damit einen Beitrag zur Überwindung der vielzitierten Softwarekrise zu leisten

---

## Zum Vergleich...

...eine typische Lösung *funktional*:

```
...
a :: [Int]
b :: [Int]
...
b = [ n | n <- a, n <= 10 ]
```

Vergleiche diese Lösung mit:  $\{n \mid n \in a \wedge n \leq 10\}$

...und der Idee, etwas von der Eleganz der Mathematik in die Programmierung zu bringen!

**Essenz funktionaler Programmierung**

...statt des "wie" das "was" in den Vordergrund stellen!

Hilfsmittel im obigen Beispiel:

- *Listenkompensation* (engl. list comprehension!)  
...typisch und spezifisch für funktionale Sprachen!

## Noch nicht überzeugt?

Betrachte *Quicksort*, ein komplexeres Beispiel...

*Aufgabe*: Sortiere eine Liste  $L$  ganzer Zahlen aufsteigend.

*Lösung* (mittels Quicksort):

- *Teile*: Wähle ein Element  $l$  aus  $L$  und partitioniere  $L$  in zwei (möglicherweise leere) Teillisten  $L_1$  und  $L_2$  so, dass alle Elemente von  $L_1$  ( $L_2$ ) kleiner oder gleich (größer) dem Element  $l$  sind.
- *Herrsche*: Sortiere  $L_1$  und  $L_2$  mit Hilfe rekursiver Aufrufe von Quicksort.
- *Zusammenführen der Teilergebnisse*: Trivial, die Gesamtliste entsteht durch Konkatenation der sortierten Teillisten.

## Quicksort...

...eine typische *prozedurale* (Pseudocode-) Realisierung:

```
quickSort (L,low,high)
  if low < high
    then splitInd = partition(L,low,high)
         quickSort(L,low,splitInd-1)
         quickSort(L,splitInd+1,high) fi

partition (L,low,high)
  l = L[low]
  left = low
  for i=low+1 to high do
    if L[i] <= l then left = left+1
                swap(L[i],L[left]) fi od
  swap(L[low],L[left])
  return left
```

...mit dem initialen Aufruf `quickSort(L,1,length(L))`.

## Zum Vergleich...

...eine typische *funktionale* Realisierung von Quicksort, hier in Haskell:

```
quickSort :: [Int] -> [Int]

quickSort [] = []
quickSort (x:xs) =
  quickSort [ y | y<-xs, y<=x ] ++
  [x] ++ quickSort [ y | y<-xs, y>x ]
```

## Vorteile funktionaler Programmierung

- *Einfach(er) zu erlernen*  
...da weniger Grundkonzepte (insbesondere: keine Zuweisung, keine Schleifen, keine Sprünge)
- *Höhere Produktivität*  
...da Programme dramatisch kürzer als funktional vergleichbare imperative Programme (Faktor 5 bis 10)
- *Höhere Zuverlässigkeit*  
...da Korrektheitsüberlegungen/-beweise einfach(er) (math. Hintergrund, keine durchscheinende Maschine)

## Nachteile funktionaler Programmierung

- *Geringe(re) Effizienz*  
...aber: enorme Fortschritte, zudem Korrektheit vorrangig gegenüber Geschwindigkeit, weiters einfache(re) Parallelisierbarkeit
- *Gelegentlich unangemessen*, oft für inhärent zustandsbasierte Anwendungen oder zur GUI-Programmierung  
...aber: Anwendungseignung ist stets zu überprüfen; kein Spezifikum fkt. Programmierung

## Warum Haskell?

Ein Blick auf andere funktionale (Programmier-)sprachen...

- $\lambda$ -Kalkül (Ende der 30er-Jahre, Alonzo Church, Stephen Kleene)
- Lisp (frühe 60er-Jahre, John McCarthy)
- ML, SML (Mitte 70er-Jahre, Michael Gordon, Robin Milner)
- Hope (um 1980, Rod Burstall, David McQueen)
- Miranda (um 1980, David Turner)
- OPAL (Mitte der 80er-Jahre, Peter Pepper et al.)
- Haskell (Ende der 80er-Jahre, Paul Hudak, Philip Wadler et al.)
- Gofer (Anfang der 90er-Jahre, Mark Jones)
- ...

## Warum etwa nicht Haskell?

Haskell ist...

- eine fortgeschrittene moderne funktionale Sprache
  - starke Typisierung
  - lazy evaluation
  - pattern matching
  - Polymorphie
  - Generizität
  - Datenabstraktion (abstrakte Datentypen)
  - Modularisierung (Programmierung im Großen)
  - ...
- eine Sprache für "real world" Probleme (S.a. <http://homepages.inf.ed.ac.uk/wadler/realworld/index.html> (URL noch gültig?))
  - mächtige Bibliotheken
  - Schnittstellen z.B. zu C
  - ...

Nicht zuletzt: Wenn auch reich, ist Haskell eine "gute" Lehrsprache, auch dank Hugs!

## Fassen wir noch einmal zusammen...

Steckbrief: **Funktionale Programmierung**

Grundlage: Lambda-Kalkül

Abstraktion: Funktionen (höherer Ordnung)

Eigenschaft: referentielle Transparenz

Historische Bedeutung: Basis vieler Programmiersprachen

Anwendungsbereiche: Theoretische Informatik  
Artificial Intelligence  
experimentelle Software  
Programmierunterricht

Programmiersprachen: Lisp, ML, Miranda, Haskell,...

---

## sowie...

### Steckbrief: Haskell

benannt nach: Haskell B. Curry (1900-1982)  
<http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Curry.html>

Paradigma: rein funktionale Programmierung

Eigenschaften: lazy evaluation, pattern matching

Typsicherheit: stark typisiert, Typinferenz  
modernes polymorphes Typsystem

Syntax: komprimiert, intuitiv

Informationen: <http://haskell.org>  
<http://haskell.org/tutorial/>

Interpreter: Hugs (<http://haskell.org/hugs/>)

---

## Erste Schritte in Haskell

---

## Haskell-Programme...

...gibt es in zwei (notationellen) Varianten:

Als sog.

- (Gewöhnliches) Haskell-Skript

*Intuitiv* ...alles, was nicht als Kommentar notationell ausgezeichnet ist, wird als Programmtext betrachtet.

- Literate Haskell-Skript

*Intuitiv* ...alles, was nicht als Programmtext notationell ausgezeichnet ist, wird als Kommentar betrachtet.

---

## Zur Illustration: Ein Programm als...

...(gewöhnliches) Haskell-Skript:

```
{-#####  
  FirstScript.hs ...'ordinary scripts' erhalten  
  die Dateiendung .hs  
#####-}  
  
-- Die konstante Funktion sum17and4  
sum17and4 :: Int  
sum17and4 = 17+4  
  
-- Die Funktion square zur Quadrierung einer ganzen Zahl  
square :: Int -> Int  
square n = n*n  
  
-- Die Funktion double zur Verdopplung einer ganzen Zahl  
double :: Int -> Int  
double n = 2*n  
  
-- Die Funktion doubleSquare, eine Anwendung der vorherigen  
doubleSquare :: Int  
doubleSquare = double (square (4711 - sum17and4))
```

---

## Zum Vergleich das gleiche Programm...

...als literate Haskell-Skript:

```
#####  
  FirstLiterate.lhs ...'literate scripts' erhalten  
  die Dateiendung .lhs  
#####  
  
Die konstante Funktion sum17and4  
> sum17and4 :: Int  
> sum17and4 = 17+4  
  
Die Funktion square zur Quadrierung einer ganzen Zahl  
> square :: Int -> Int  
> square = n*n  
  
Die Funktion double zur Verdopplung einer ganzen Zahl  
> double :: Int -> Int  
> double = 2*n  
  
Die Funktion doubleSquare, eine Anwendung der vorherigen  
> doubleSquare :: Int  
> doubleSquare = double (square (4711 - sum17and4))
```

---

## Kommentare in Haskell-Programmen

Kommentare in...

- (gewöhnlichem) Haskell-Skript
  - *einzeilig*: ...bis zum Rest der Zeile nach --
  - *mehrzeilig*: ...alles zwischen {- und -}
- *literate Haskell-Skript*
  - Jede nicht durch > eingeleitete Zeile

*Konvention*: Dateiendung...

- .hs für gewöhnliche
- .lhs für literate

Haskell-Skripte.

---

## Erste Schritte mit Hugs

---

## Hugs: Der Haskell-Interpreter...

Aufruf von Hugs: `hugs <fileName>`

...und z.B. im Fall von FirstScript für <fileName> weiter mit:

```
Main> double (sum17and4)  
42
```

Wichtige Kommandos in Hugs:

:?	Liefert Liste der Hugs-Kommandos
:load <fileName>	Lädt die Haskell-Datei <fileName> (erkennbar an Endung .hs bzw. .lhs)
:reload	wiederholt letztes Ladekommando
:quit	Beendet den aktuellen Hugs-Lauf
:info name	Liefert Information über das mit name bezeichnete "Objekt"
:type exp	Liefert den Typ des Argumentausdrucks exp
:edit <fileName>.hs	Öffnet die Datei <fileName>.hs enthaltende Datei im voreingestellten Editor
:find name	Öffnet die Deklaration von name im voreingestellten Editor
!<com>	Ausführen des Unix- oder DOS-Kommandos <com>

...mehr dazu: <http://www.haskell.org/hugs/>

## Fehlermeldungen & Warnungen in Hugs

- Fehlermeldungen
  - Syntaxfehler  
Main> sum17and4 == 21) ...liefert  
ERROR: Syntax error in input (unexpected ')')
  - Typfehler  
Main> sum17and4 + False ...liefert  
ERROR: Bool is not an instance of class 'Num'
  - Programmfehler  
...später
  - Modulfehler  
...später
- Warnungen
  - Systemmeldungen  
...später

...mehr zu Fehlermeldungen:

<http://www.cs.kent.ac.uk/people/staff/sjt/craft2e/errors.html>

## Bequem...

Haskell stellt umfangreiche Bibliotheken (`Prelude.hs,...`) mit vielen vordefinierten Funktionen zur Verfügung, z.B. zum

- Umkehren von Zeichenreihen (`reverse`)
- Aufsummieren von Listenelementen (`sum`)
- Verschmelzen von Listen (`zip`)
- ...

## Exkurs: Mögliche Namenskonflikte

...soll eine Funktion gleichen (bereits vordefinierten) Namens deklariert werden, können Namenskonflikte durch *Verstecken* (engl. *hiding*) vordefinierter Namen vermieden werden.

Am Beispiel von `reverse`, `sum`, `zip`:

Ergänze...

```
import Prelude hiding (reverse,sum,zip)
```

...am Anfang des Haskell-Skripts im Anschluss an die Modul-Anweisung (so vorhanden), wodurch die vordefinierten Namen `reverse`, `sum` und `zip` verborgen werden.

(Mehr dazu später im Zusammenhang mit dem Modulkonzept von Haskell).

## Teil 2: Grundlagen

- Elementare Datentypen (`Bool`, `Int`, `Integer`, `Float`, `Char`)
- Tupel, Listen und Funktionen

## Elementare Datentypen

...werden in der Folge nach nachstehendem Muster angegeben:

- Name des Typs
- Typische Konstanten des Typs
- Typische Operatoren (und Relatoren, so vorhanden)

## (Ausgew.) Elementare Datentypen (1)

### Wahrheitswerte

Typ	Bool	Wahrheitswerte
Konstanten	<code>True :: Bool</code> <code>False :: Bool</code>	Symbol für "wahr" Symbol für "falsch"
Operatoren	<code>&amp;&amp; :: Bool -&gt; Bool -&gt; Bool</code> <code>   :: Bool -&gt; Bool -&gt; Bool</code> <code>not :: Bool -&gt; Bool</code>	logisches und logisches oder logische Negation

## Elementare Datentypen (2)

### Ganze Zahlen

Typ	Int	Ganze Zahlen(endl. Ausschn.)
Konstanten	<code>0 :: Int</code> <code>-42 :: Int</code> <code>2147483647 :: Int</code> ...	Symbol für "0" Symbol für "-42" Wert für "maxInt"
Operatoren	<code>+ :: Int -&gt; Int -&gt; Int</code> <code>* :: Int -&gt; Int -&gt; Int</code> <code>^ :: Int -&gt; Int -&gt; Int</code> <code>- :: Int -&gt; Int -&gt; Int</code> <code>- :: Int -&gt; Int</code> <code>div :: Int -&gt; Int -&gt; Int</code> <code>mod :: Int -&gt; Int -&gt; Int</code> <code>abs :: Int -&gt; Int</code> <code>negate :: Int -&gt; Int</code>	Addition Multiplikation Exponentiation Subtraktion (Infix) Vorzeichenwechsel (Prefix) Division Divisionsrest Absolutbetrag Vorzeichenwechsel

## Elementare Datentypen (3)

### Ganze Zahlen (fortgesetzt)

Relatoren	<code>&gt; :: Int -&gt; Int -&gt; Bool</code> <code>&gt;= :: Int -&gt; Int -&gt; Bool</code> <code>== :: Int -&gt; Int -&gt; Bool</code> <code>/= :: Int -&gt; Int -&gt; Bool</code> <code>&lt;= :: Int -&gt; Int -&gt; Bool</code> <code>&lt; :: Int -&gt; Int -&gt; Bool</code>	echt größer größer gleich gleich ungleich keiner gleich echt kleiner
-----------	--	---

...die Relatoren `==` und `/=` sind auf Werte aller Elementar- und vieler weiterer Typen anwendbar, beispielsweise auch auf Wahrheitswerte (Stichwort: *Überladen* (engl. *Overloading*))!

...mehr dazu später.

## Elementare Datentypen (4)

### Ganze Zahlen (Variante)

Typ	Integer	Ganze Zahlen
Konstanten	0 :: Integer	Symbol für "0"
	-42 :: Integer	Symbol für "-42"
	21474836473853883234 :: Integer	"Große" Zahl
Operatoren	...	...

...wie Int, jedoch ohne a priori Beschränkung für eine maximal darstellbare Zahl.

## Elementare Datentypen (5)

### Gleitkommazahlen

Typ	Float	Gleitkommazahlen (endlicher Ausschnitt)
Konstanten	0.123 :: Float	Symbol für "0,123"
	-42.4711 :: Float	Symbol für "-42,4711"
	123.6e-2 :: Float	123,6×10 <sup>-2</sup>
Operatoren	+ :: Float -> Float -> Float	Addition
	* :: Float -> Float -> Float	Multiplikation
	sqrt :: Float -> Float	(pos.) Quadratwurzel
	sin :: Float -> Float	sinus
Relatoren	== :: Float -> Float -> Bool	gleich
	/= :: Float -> Float -> Bool	ungleich

## Elementare Datentypen (6)

### Zeichen

Typ	Char	Zeichen (Literal)
Konstanten	'a' :: Char	Symbol für "a"
	'Z' :: Char	Symbol für "Z"
	'\t' :: Char	Tabulator
	'\n' :: Char	Neue Zeile
	'\\' :: Char	Symbol für "backslash"
	'\'' :: Char	Hochkomma
	'\"' :: Char	Anführungszeichen
Operatoren	ord :: Char -> Int	Konversionsfunktion
	chr :: Int -> Char	Konversionsfunktion

## Zusammengesetzte Datentypen und Funktionen...

- Tupel
- Listen
  - *Spezialfall*: Zeichenreihen
- Funktionen

## Tupel

Tupel ...fassen eine festgelegte Zahl von Werten möglicherweise verschiedener Typen zusammen.

↪ Tupel sind *heterogen*!

### Beispiele:

- ...Modellierung von Studentendaten

```
("Max Mustermann", "e0123456@student.tuwien.ac.at", 534) :: (String, String, Int)
```
- ...Modellierung von Bibliotheksdaten

```
("PeytonJones", "Implementing Funct. Lang.", 1987, True) :: (String, String, Int, Bool)
```

## Tupel...

- Allgemeines Muster

```
(v1, v2, ..., vk) :: (T1, T2, ..., Tk)
```

mit  $v_1, \dots, v_k$  Bezeichnungen von Werten und  $T_1, \dots, T_k$  Bezeichnungen von Typen mit

```
v1 :: T1, v2 :: T2, ..., vk :: Tk
```

*Lies*:  $v_i$  ist vom Typ  $T_i$
- Standardkonstruktor

```
(. , . , ... , .)
```

## Spezialfall: Paare ("Zweitupel")

- Beispiele

```
type Point = (Float, Float)

(0.0, 0.0) :: Point
(3.14, 17.4) :: Point
```

- Standardselektoren (für Paare)

```
fst (x, y) = x
snd (x, y) = y
```

- Anwendung der Standardselektoren

```
fst (0.0, 0.0) = 0.0
snd (3.14, 17.4) = 17.4
```

## Hilfreich...

### Typsynonyme

```
type Student = (String, String, Int)
type Buch = (String, String, Int, Bool)
```

...erhöhen die Transparenz in Programmen.

*Beachte*: Typsynonyme definieren *keine* neuen Typen, sondern einen Namen für einen schon existierenden Typ (später mehr dazu).

## Tupel...

Selbstdefinierte Selektorfunktionen...

```
type Student = (String, String, Int)
```

```
name :: Student -> String
email :: Student -> String
kennzahl :: Student -> Int
```

```
name (n,e,k) = n
email (n,e,k) = e
kennzahl (n,e,k) = k
```

...mittels *Mustererkennung* (engl. *pattern matching*)  
(später mehr dazu).

## Selbstdefinierte Selektorfunktionen...

Ein weiteres Beispiel...

```
autor :: Buch -> String
kurzTitel :: Buch -> String
erscheinungsJahr :: Buch -> Int
ausgeliehen :: Buch -> Bool
```

```
autor (a,t,j,b) = a
kurzTitel (a,t,j,b) = t
erscheinungsJahr (a,t,j,b) = j
ausgeliehen (a,t,j,b) = b
```

```
autEntlehnt (a,t,j,b) = (autor (a,t,j,b), ausgeliehen (a,t,j,b))
```

...auch hier mittels Mustererkennung

## Listen

*Listen* ...fassen eine beliebige/unbestimmte Zahl von Werten gleichen Typs zusammen.

↪ Listen sind *homogen!*

*Einfache Beispiele:*

- Listen ganzer Zahlen  
[2,5,12,42] :: [ Int ]
- Listen von Wahrheitswerten  
[True,False,True] :: [ Bool ]
- Listen von Gleitkommazahlen  
[3.14,5.0,12.21] :: [ Float ]
- Leere Liste  
[]
- ...

## Listen

*Beispiele komplexerer Listen:*

- Listen von Listen  
[[2,4,23,2,5],[3,4],[],[56,7,6,]] :: [[Int]]
- Listen von Paaren  
[(3.14,42.0),(56.1,51.3)] :: [(Float,Float)]
- ...
- *Ausblick:* Listen von Funktionen  
[fac, abs, negate] :: [Integer -> Integer]

## Vordefinierte Funktionen auf Listen

Die Funktion `length` mit einigen Aufrufen:

```
length :: [a] -> Integer
length [] = 0
length (x:xs) = 1 + length xs
```

```
length [1, 2, 3] => 3
length ['a','b','c'] => 3
length [[1],[2],[3]] => 3
```

Die Funktionen `head` und `tail` mit einigen Aufrufen:

```
head :: [a] -> a
head (x:xs) = x
```

```
tail :: [a] -> [a]
tail (x:xs) = xs
```

```
head [[1],[2],[3]] => [1]
tail [[1],[2],[3]] => [[2],[3]]
```

## Zeichenreihen

...in Haskell als spezielle Listen realisiert:

Typ	String type String = [Char]	Zeichenreihen Deklaration (als Liste von Zeichen)
Konstanten	"Haskell" :: String " ...	Zeichenr. für "Haskell" Leere Zeichenreihe
Operatoren	++ :: String -> String -> String	Konkatenation
Relatoren	== :: String -> String -> Bool /= :: String -> String -> Bool	gleich ungleich

## Spezielle Notationen für Listen

- Spezialfälle (i.w. für Listen über Zahlen und Zeichen)  
...[2 .. 6] kurz für [2,3,4,5,6]  
...[11,9 .. 2] kurz für [11,9,7,5,3]  
...['a','d' .. 'j'] kurz für ['a','d','g','j']  
...[0.0,0.3 .. 1.0] kurz für [0.0,0.3,0.6,0.9]
- *Listenkomprehension*  
...ein erstes Beispiel:  
[3\*n | n <- list] kurz für [3,6,9,12], wobei hier `list` vom Wert [1,2,3,4] vorausgesetzt ist.  
↪ Listenkomprehension ist ein sehr elegantes und ausdruckskräftiges Sprachkonstrukt!

## Weitere Beispiele zu Zeichenreihen

```
['h','e','l','l','o'] == "hello"
"hello" ++ " world" == "hello world"
```

Es gilt:

```
[1,2,3] == 1:2:3:[]
```

## Funktionen in Haskell

...am Beispiel der Fakultätsfunktion:

Zur Erinnerung:

$$n! = \begin{cases} 1 & \text{falls } n = 0 \\ n * (n - 1)! & \text{sonst} \end{cases}$$

...und eine mögliche Realisierung in Haskell:

```
fac :: Integer -> Integer
fac n = if n == 0 then 1 else n * fac(n-1)
```

*Beachte:* ...Haskell stellt eine Reihe, oft eleganterer, notati-  
oneller Varianten zur Verfügung!

## Fkt. in Haskell: Notat. Varianten (1)

...am Beispiel der Fakultätsfunktion.

```
fac :: Integer -> Integer
```

### (1) In Form "bedingter Gleichungen"

```
fac n
| n == 0    = 1
| otherwise = n * fac (n - 1)
```

~> *Hinweis:* Variante (1) ist "der" Regelfall in Haskell!

## Fkt. in Haskell: Notat. Varianten (1)

### (2) $\lambda$ -artig

```
fac = \n -> (if n == 0 then 1 else (n * fac (n - 1)))
```

- Reminiszenz an den funktionaler Programmierung zugrun-  
deliegenden  $\lambda$ -Kalkül ( $\lambda xy. (x + y)$ )
- In Haskell:  $\lambda xy \rightarrow x + y$  sog. *anonyme* Funktion. Praktisch,  
wenn der Name keine Rolle spielt und man sich deshalb  
bei Verwendung anonymer Funktionen keinen zu überlegen  
braucht.

### (3) Gleichungsorientiert

```
fac n = if n == 0 then 1 else (n * fac (n - 1))
```

## Fkt. in Haskell: Notat. Varianten (2)

...am Beispiel weiterer Funktionen.

```
kVA :: Float -> (Float, Float)
-- Berechnung von Volumen (V) und Fläche (A)
-- einer Kugel (K).
Zur Erinnerung:  $V = \frac{4}{3} \pi r^3$   $A = 4 \pi r^2$ 
```

Mittels *lokaler Deklarationen*...

### (4a) where-Konstrukt

```
kVA r =
  ((4/3) * myPi * rcube r, 4 * myPi * square r)
  where
    myPi    = 3.14
    rcube x = x * square x
    square x = x * x
```

## Fkt. in Haskell: Notat. Varianten (3)

bzw...

### (4b) let-Konstrukt

```
kVA r =
  let
    myPi    = 3.14
    rcube x = x * square x
    square x = x * x
  in
    ((4/3) * myPi * rcube r, 4 * myPi * square r)
```

## Fkt.in Haskell: Notat. Varianten (4)

In einer Zeile...

### (5a) ...mittels ";

```
kVA r =
  ((4/3) * myPi * rcube r, 4 * myPi * square r)
  where
    myPi = 3.14; rcube x = x * square x; square x = x * x
```

### (5b) ...mittels "in"

```
kVA r =
  let myPi = 3.14; rcube x = x * square x; square x = x * x
  in
    ((4/3) * myPi * rcube r, 4 * myPi * square r)
```

## Fkt. in Haskell: Notat. Varianten (5)

Spezialfall: *binäre (zweistellige)* Funktionen...

```
imax :: Integer -> Integer -> Integer
```

```
imax p q
| p >= q    = p
| otherwise = q
```

```
tripleMax :: Integer -> Integer -> Integer -> Integer
```

```
tripleMax p q r
| (imax p q == p) && (p 'imax' r == p) = p
| ...
| otherwise                          = r
```

...imax in tripleMax als *Präfix-* und als *Infixoperator* verwandt

## Fkt. in Haskell: Notat. Varianten (6)

*Musterbasiert*...

```
fib :: Integer -> Integer
```

```
fib 0 = 1
fib 1 = 1
fib n = fib(n-2) + fib(n-1)
```

```
capVowels :: Char -> Char
```

```
capVowels 'a' = 'A'
capVowels 'e' = 'E'
capVowels 'i' = 'I'
capVowels 'o' = 'O'
capVowels 'u' = 'U'
capVowels c  = c
```

---

## Fkt. in Haskell: Notat. Varianten (7)

Mittels *case*-Ausdrucks...

```
capVowels :: Char -> Char      deCapVowels :: Char -> Char
capVowels letter              deCapVowels letter
= case letter of              = case letter of
  'a'  -> 'A'                  'A'    -> 'a'
  'e'  -> 'E'                  'E'    -> 'e'
  'i'  -> 'I'                  'I'    -> 'i'
  'o'  -> 'O'                  'O'    -> 'o'
  'u'  -> 'U'                  'U'    -> 'u'
  letter -> letter             otherwise -> letter
```

---

## Fkt. in Haskell: Notat. Varianten (8)

Mittels *Muster* und "*wild cards*"...

```
add :: Integer -> Integer -> Integer
add n 0 = n
add 0 n = n
add m n = m+n

mult :: Integer -> Integer -> Integer
mult _ 0 = 0
mult 0 _ = 0
mult m n = m*n
```

---

## Muster können (u.a.) sein...

- *Werte* (z.B. 0, 'c', True)  
...ein Argument "passt" auf das Muster, wenn es vom entsprechenden Wert ist.
  - *Variablen* (z.B. n)  
...jedes Argument passt.
  - *Wild card* "\_"  
...jedes Argument passt (sinnvoll für nicht zum Ergebnis beitragende Argumente)
  - ...
- mehr über Muster und musterbasierte Funktionsdefinitionen später...

---

## Literaturhinweis

...auf den Haskell-Sprachreport:

- *Haskell 98: Language and Libraries. The Revised Report.* Simon Peyton Jones (Hrsg.), Cambridge University Press, 2003.

---

## Zum ersten Aufgabenblatt...

- ...erhältlich ausschließlich im Web unter folgender URL  
[http://www.complang.tuwien.ac.at/knoop/fp185161\\_ws0607.html](http://www.complang.tuwien.ac.at/knoop/fp185161_ws0607.html)  
...ab Montag, den 16.10.2006
- Abgabe: Mo, den 23.10.2006, 12:00 Uhr
- Zweitabgabe: Mo, den 30.10.2006, 12:00 Uhr

**Vorschau:**  
Ausgabe des...

- zweiten Aufgabenblatts: Mo, den 23.10.2006  
...Abgabetermine: Mo, 30.10.2006, und Mo, 07.11.2006
- dritten Aufgabenblatts: Mo, den 30.10.2006  
...Abgabetermine: Mo, 07.11.2006, und Mo, 14.11.2006

---

## Vorschau auf die nächsten Vorlesungstermine...

- **Ersatztermin:** Di, 17.10.2006, 13 Uhr s.t. - 14 Uhr, Treitl-Hörsaal
- Do, 19.10.2006, Vorlesung von 16:30 Uhr bis 18:00 Uhr im Radinger-Hörsaal
- **Ersatztermin:** Di, 24.10.2006, 13 Uhr s.t. - 14 Uhr, Treitl-Hörsaal
- Do, 26.10.2006: *Nationalfeiertag, keine Vorlesung!*