Funktionale Programmierung

LVA 185.A03, VU 2.0, ECTS 3.0
WS 2020/2021

Vortrag I
Orientierung, Einordnung
06.10.2020

Jens Knoop

Technische Universität Wien
Information Systems Engineering
Compilers and Languages

Vortrag I
Präludium
Teil I
Kap. 1
Hinweis
Aufgabe

Vortrag I

Orientierung, Einordnung

...zum selbstgeleiteten, eigenständigen Weiterlernen.

Teil I: Einführung

- Kapitel 1: Motivation

Vortrag I
Präludium
Teil I
Kap. 1
Hinweis
Aufgabe

Präludium

Funktionen, funktionales Programmieren

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Funktionen: Bekannt aus der Mathematik

...sinus-, cosinus-, Wurzel-, Exponential-, Logarithmusfkt.,...

Die Fakultätsfunktion:

$$! : \mathbb{N} \to \mathbb{N}$$

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

Die Fibonacci-Funktion:

$$fib : \mathbb{N} \to \mathbb{N}$$

$$fib(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ fib(n-2) + fib(n-1) & \text{falls } n > 1 \end{cases}$$

Funktionale Programme

...bestehen ausschließlich aus Funktionen!

Hauptprogramm eines funktionalen Programms:

- eine Funktion, die Hauptfunktion.

Programmeingabe, Programmausgabe:

- Argument und Resultat der Hauptfunktion.

Programmaufbau:

- die Hauptfunktion stützt sich ab auf andere Funktionen,
- die sich auf weitere Funktionen abstützen,
- eine Abstützung, die bei einfachsten, sog. primitiven Funktionen, schließlich zum Abschluss kommt.

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Funktionen: Schon bekannt aus imperativen, objektorientierten Programmiersprachen (1)

...zum Beispiel aus Java, Beispiel einer aktuell einflussreichen objektorientierten Sprache (Mitte der 1990er Jahre):

```
int fac (int n) {                                    (Fakultätsfkt.)
 if (n  <  0) return -1;
 if (n == 0) return 1;
 return (n * fac (n-1));
}

int fib (int n) {                                    (Fibonacci-Fkt.)
 if (n  <  0) return -1;
 if (n == 0) return 0;
 if (n == 1) return 1;
 return (fib (n-1) + fib (n-2));
}
```

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Funktionen: Schon bekannt aus imperativen, objektorientierten Programmiersprachen (2)

..zum Beispiel aus Pascal, Beispiel einer einflussreichen klassischen imperativen Sprache (Mitte der 1970er Jahre):

```
FUNCTION fac (n: integer): integer;
BEGIN
 IF n=0 THEN fac := 1 ELSE fac := n*fac(n-1)
END;

FUNCTION fib (n: integer): integer;
BEGIN
 IF n=0
  THEN fib := 0
  ELSE IF n=1
         THEN fib := 1
         ELSE fib := fib(n-1) + fib(n-2)
END;
```

Funktionen, bekannt bereits aus imperativen, objektorientierten Programmiersprachen (3)

...zum Beispiel aus Fortran, Beispiel einer der allerersten höheren Programmiersprachen überhaupt (Mitte der 1950er Jahre):

(Fortran 77)

```
      INTEGER FUNCTION FAC(N)
      FAC = 1
      DO 100 I = 2, N
         FAC = I * FAC
100   CONTINUE
      RETURN
      END
```

(Fortran 95)

```
recursive function fib(n) result(m)
integer, intent(in) :: n
integer :: m
if (n == 0) then m = 0
else if (n == 1) then m = 1
     else m = fib(n-1) + fib(n-2)
end if
end function fib
```

Gleiches Ziel, unterschiedliche Syntax

...in der Mathematik: $! : \mathbb{N} \to \mathbb{N}$

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

...in Java:
```
int fac (int n) {
if (n  <  0) return -1;
if (n == 0) return 1;
return (n * fac (n-1));
```

...in Pascal:
```
FUNCTION fac (n: integer): integer;
BEGIN
 IF n=0 THEN fac := 1 ELSE fac := n*fac(n-1)
END;
```

...in Fortran 77:
```
      INTEGER FUNCTION FAC(N)
      FAC = 1
      DO 100 I = 2, N
         FAC = I * FAC
  100 CONTINUE
      RETURN
      END
```

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Gleiches Recht auf eigene Syntax auch f. fkt. Spr.

...die Fakultäts-Funktion in Scheme, ein Lisp-Dialekt;
Lisp (späte 1950er Jahre):

```
(define fac
  (lambda (n)
   (if (= n 1)
       1
       (* (fac (- n 1))
          n))))
```

Lisp, Akronym für:

- List Processing (Language)

Unter der Hand aber auch für:

- Lots of Irritating/Insiduous Silly/Superfluous Parentheses
  Lost In Stupid Parentheses

Gleiches Recht auf eigene Syntax auch f. fkt. Spr.

...in (Standard) ML (Mitte der 1980er Jahre); in 3 Varianten:

Grundform:

```
val rec fac : int->int =
    fn 0 => 1 | n:int => n * fac (n-1)
```

Syntaktisch 'gezuckert':

```
fun fac (n:int):int =
  case n
    of 0 => 1
     | n => n * fac (n-1)

fun fac 0 = 1
  | fac (n:int) = n * fac (n-1)
```

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Gleiches Recht auf eigene Syntax auch f. fkt. Spr.

...in Opal (Mitte der 1980er Jahre):

```
fun fac: nat -> nat
DEF fac == \\n.
     IF  n=0  THEN 1
     IF n>=1 THEN n * fac(n-1) FI
```

(*vgl. math. Schreibw.* : ! : IN $\rightarrow$ IN)

...in Miranda (Mitte der 1980erJahre):

```
fac :: num -> num
fac n = 1                  , if n = 0
      = n * fac (n-1), if n > 0
```

Vgl. wieder mit math. Schreibweise:

$$! : \mathbb{N} \rightarrow \mathbb{N}$$

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

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Gleiches Recht auf eigene Syntax auch f. fkt. Spr.

...in Haskell (späte 1980er Jahre); auch hier gleich mehrere Varianten:

```
fac :: Int -> Int                    (mit bedingtem Ausdruck)
fac n = if n == 0 then 1 else n * fac (n-1)
```

```
fac :: Int -> Int                  (mit bewachten Gleichungen)
fac n
  | n == 0 = 1
  | True   = n * fac (n-1)
```

```
fac :: Int -> Int                                  (mit Mustern)
fac 0 = 1
fac n = n * fac (n-1)
```

```
fac :: Int -> Int                (mit anonymer λ-Abstraktion)
fac = \n -> if n == 0 then 1 else n * fac (n-1)
```

```
fac :: Int -> Int                  (mit vordef. Listen-Fkt.)
fac n = prod [1..n]
```

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

All das zeigt: Funktionen

...sind – bei großer syntaktischer Variabilität – allgegenwärtig in Programmiersprachen und Programmierung, gleich ob funktional oder nicht, und das von den allerersten Anfängen an!

Nihil novi sub sole?
Nichts Neues unter der Sonne?

Vulgata, lat. Neuübersetzung der Bibel aus dem 4. Jhdt. von Hieronymus (um 347 - 419 oder 420)
Liber Ecclesiastes 1,10

Much Ado about Nothing?
Viel Lärm um nichts?

William Shakespeare (um 1564 - 1616)
Titel einer (sehr beliebten) Shakespeare-Komödie

Das wäre ein Kurz- und vor allem Fehlschluss!

...der **Clou** funktionaler Programmierung sind nicht Funktionen an sich, sondern das

- 'Rechnen mit Funktionen'!

'Rechnen mit Funktionen' heißt:

- Mit Funktionen zu rechnen wie mit elementaren Werten!
- Argument und Resultat von Funktionen sind Funktionen!

Ein Beispiel aus der Mathematik:

- Die Funktionskomposition: $f \circ g$

  Argumente der Funktion $\cdot \circ \cdot$: Zwei Funktionen f und g.

  Resultat von $\cdot \circ \cdot$: Die Komposition h von f und g, auch eine Funktion: $h(x) = (f \circ g)(x) = f(g(x))$.

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Weitere Beispiele aus der Mathematik

- Differenzieren: $\frac{d}{dx} f(x)$

  Argument der Funktion $\frac{d}{dx}\cdot$: Eine Funktion f.

  Resultat von $\frac{d}{dx}\cdot$: Die Ableitung f' von f, auch eine Funktion.

- Integrieren 1: $\int_a^t f(x)\,dx$ mit a fest, t variabel.

  Argument der Funktion $\int_a^t \cdot\, dx$: Eine Funktion f.

  Resultat von $\int_a^t \cdot\, dx$: Das Integral F von f, auch eine Funktion.

  (mit $F' = f$ und $F(a) = 0$)

- Integrieren 2: $\int f(x)\,dx$

  Argument der Funktion $\int \cdot\, dx$: Eine Funktion f.

  Resultat von $\int \cdot\, dx$: Das unbestimmte Integral von f, die Menge $M = \{F \mid F' = f\}$ der Stammfunktionen von f.

Funktionale Programmierung

...vs. imperative, objektorientierte Programmierung

Funktionales Programmieren ist

- das Rechnen mit Funktionen!
  (Funktionen als Argument und Resultat von Funktionen!)

Imperatives, objektorientiertes Programmieren ist

- (was immer es ist) das **nicht!**

Aus diesem Unterschied ergeben sich neue, weitreichende

- Möglichkeiten für die Programmierung!

Um diese Möglichkeiten geht es in dieser LVA!

...einen weiteren wichtigen Unterschied, den wir bereits jetzt voll verstehen können (und nicht erst im Lauf des Semesters), betrachten wir als nächstes.

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Funktionen: Imperativ/oo vs. funktional (1)

Vergleiche die funktionale und imperative Implementierung der Fakultätsfunktion mithilfe einer Fallunterscheidung:

Imperativ (Pascal):

```
FUNCTION fac (n: integer): integer;
BEGIN
  IF n=0 THEN fac := 1 ELSE fac := n*fac(n-1)
END;
```

Funktional (Haskell):

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

...imperative und funktionale Fallunterscheidung sind nur äußerlich sehr ähnlich, **konzeptuell aber sehr verschieden!**

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Funktionen: Imperativ/oo vs. funktional (2)

Die Fallunterscheidung 'if-then-else' im Vergleich:

Imperativ:

```
FUNCTION fac (n: integer): integer;
BEGIN IF n=0 THEN fac := 1 ELSE fac := n*fac(n-1) END;
```

Ausdruck *Ausdruck* *Ausdruck*

Anweisung *Anweisung*

Anweisung

Funktional:

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

Ausdruck *Ausdruck* *Ausdruck*

Ausdruck

Funktionen: Imperativ vs. funktional (3)

Die Fallunterscheidung 'if-then-else':

- Imperativ: Die Fallunterscheidung ist eine **Anweisung**. Ihre Bedeutung (Semantik) ist eine **Zustandstransformation**, eine Belegung von Variablen mit (neuen) Werten.
- Funktional: Die Fallunterscheidung ist ein **Ausdruck**. Ihre Bedeutung (Semantik) ist ein **Wert**. Anweisungen gibt es nicht!

Ergo:

– 'if-then-else' imperativ $\neq$ 'if-then-else' funktional

Dieser Unterschied in Konzept u. Bedeutung ist fundamental:

– Imperativ/objektorientiert: Ausdrücke und Anweisungen.
– Funktional: Ausschließlich Ausdrücke, keine Anweisungen!

Bestandsaufnahme:

Was kennen Sie? Was wird neu?

...danach geht's richtig los!

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Das kennen Sie: Imperative Programmierung

...gekennzeichnet durch:

1. Unterscheidung von Ausdrücken und Anweisungen.
2. Ausdrücke liefern Werte; Anweisungen bewirken Zustandsänderungen (Seiteneffekte).
3. Programmausführung ist die Abarbeitung von Anweisungen (dabei müssen auch Ausdrücke ausgewertet werden).
4. Explizite Kontrollflussspezifikation mittels spezieller Anweisungen (sequentielle Komposition, Fallunterscheidung, Schleifen,...)
5. Variablen sind Namen für Speicherplätze: ihr Wert sind die dort gespeicherten Werte; sie können im Verlauf der Programmausführung (beliebig oft) geändert werden.
6. Bedeutung des Programms ist die Beziehung zwischen Anfangs- und Endzuständen, die bewirkte Zustandsänderung.

Das wird neu: Funktionale Programmierung

...gekennzeichnet durch:

1. Keine Anweisungen! Ausschließlich Ausdrücke!
2. Ausdrücke liefern Werte. Anweisungen fehlen; deshalb: keine Zustandsänderungen, keine Seiteneffekte.
3. Programmausführung ist Auswertung von Ausdrücken.
4. Keine Kontrollflussspezifikation! Allein Datenabhängigkeiten steuern die Auswertung(sreihenfolge).
5. Variablen sind Namen für Ausdrücke: ihr Wert ist der Wert des Ausdrucks, den sie bezeichnen; ein späteres Ändern, Überschreiben oder Neubelegen ist nicht möglich.
6. Bedeutung des Programms ist die Beziehung zwischen Aufrufargumenten von Ausdrücken und ihrem Wert.

Das kennen Sie: Imperative Programmierung

...ist befehlsbasiertes Programmieren:

1. Programm: Menge von Befehlen (oder Instruktionen, Anweisungen) strukturiert durch ein Regelwerk von Kontrollflussanweisungen (*if-then-else*, *while-do*, *for-do*,...)), das ihre Ausführungsabfolge festlegt und steuert.
2. Bedeutung von Befehlen und Programmen: Zustandsänderungen.
3. Vorlegbare Frage an ein Programm: In welchem Zustand terminiert Programm π_{imp} angesetzt auf einen Anfangszustand, oder: Was sind die Werte der Variablen von π_{imp} nach seiner Terminierung?

Das wird neu: Funktionale Programmierung

...ist gleichungsbasiertes, ergebnisorientiertes Programmieren:

1. Programm: Menge von Vereinbarungen von Wertegleichheiten von Ausdrücken.
2. Bedeutung von Ausdrücken und Programmen: Wertberechnungen.
3. Vorlegbare Frage an ein Programm: Welchen Wert hat ein Ausdruck *ausd* für konkrete Werte seiner Operanden, wenn er mit den im Programm π_{fkt} definierten Wertegleichheiten von Ausdrücken ausgewertet wird?

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Ihre Herausforderung

...konzeptuell und praktisch den Übergang von

- befehlsorientierter

zu

- gleichungs- und ergebnisorientierter

Denk- und Handlungsweise zu meistern!

...für viele ein **veritabler Kulturschock!**

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Wie immer...

Omne initium difficile.
Aller Anfang ist schwer.

lat., sprichwörtl.

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

...aber auch:

Suavis est laborum praeteritorum memoria.
Süss ist die Erinnerung an vergangene Mühen.

Cicero (106 - 43 v.Chr.)
röm. Staatsmann und Schriftsteller

All's well, that ends well.
Ende gut, alles gut.

William Shakespeare (um 1564 - 1616)
Titel einer (weniger populären) Shakespeare-Komödie

Vortrag I
Präludium
Bestandsaufn
Herausforderu
Teil I
Kap. 1
Hinweis
Aufgabe

Teil I

Einführung

Vortrag I
Präludium
Teil I
Kap. 1
Hinweis
Aufgabe

Kapitel 1

Motivation

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Das leere Haskell-Programm

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Das leere Haskell-Programm: Mehr als nichts!

...bereits das leere Haskell-Programm bietet Taschenrechner-funktionalität:

```
>ghci leeresHaskellProgramm.hs
Prelude>2+3
5
Prelude>abs (5-12)
7
Prelude>sqrt 121
11.0
Prelude>abs (-5) * 6 + 3 <= 2^3 * (4 + round 3.14)
True
Prelude>cos 0
1.0
Prelude>[-2..3]
[-2,-1,0,1,2,3]
Prelude>[n | n <- [-6..8], mod n 2 == 0]
[-6,-4,-2,0,2,4,6,8]
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Überblick

Funktionale Programmierung, funktionale Programmierung in Haskell

1.1 Ein Beispiel sagt (oft) mehr als 1000 Worte

1.2 Warum funktionale Programmierung? Warum mit Haskell?

1.3 Nützliche Werkzeuge für Haskell: Hugs, GHC, GHCi, Hoogle, Hayoo, Leksah

1.4 Literaturverzeichnis, Leseempfehlungen

Kapitel 1.1

Ein Beispiel sagt (oft) mehr als 1000 Worte

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Nichts ist schwerer zu befolgen
als ein gutes Beispiel.

Mark Twain (1835-1910)
amerik. Schriftsteller

...deshalb haben wir gleich:

Kapitel 1.1.1

Zehn Beispiele

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Zehn Beispiele

Verba docent, exempla trahunt.
Worte belehren, Beispiele reißen mit.

lat., sprichwörtl.

1. *Hello, World!*
2. !: Die Fakultätsfunktion
3. Euklidischer Algorithmus (größter gemeinsamer Teiler)
4. Gerade/ungerade-Test für ganze Zahlen
5. Längenberechnung von Listen
6. Umkehren von Zeichenreihen
7. Transformieren von Listen
8. Addieren von Zahlen
9. Binomialkoeffizientenberechnung
10. Das Sieb des Eratosthenes (Primzahlberechnung)

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Probieren Sie die Beispiele aus!

Programmieren ist wie schwimmen.
Man kann jahrelang zusehen,
ohne es zu lernen.

unbekannt

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

...in 6 einfachen Schritten!

1. Öffnen Sie eine neue Datei, z.B. `haskell_appetizer.hs`
2. Schreiben Sie die Funktionen der Beispiele in diese Datei und speichern Sie sie:

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

binom :: Integer -> (Integer -> Integer)
binom n k = div (fac n) (fac k * fac (n-k))

...
```

   Manche Beispielfunktionen sind in Haskell schon vordefiniert (`length`, `map`,...); die brauchen Sie nicht hineinzuschreiben.

3. Laden Sie die Datei in einem Kommandofenster in einen Haskell-Interpretierer (den Sie vorher installiert haben müssen), z.B. GHCi oder Hugs:

```
>ghci haskell_appetizer.hs
>hugs haskell_appetizer.hs
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

...ist es getan!

4. Der Interpretierer meldet sich mit einer Eingabeaufforderung, z.B.:

```
Main>
```

5. Geben Sie Ausdrücke ein und lassen Sie sie auswerten:

```
Main>fac 5
120
Main>binom 45 6
8145060
Main>3 + fac (2+3)
123
Main>map fac [0..5]
[1,1,2,6,24,120]
Main>length [fac,fac,fac]
3
```

6. Beenden Sie schließlich den Interpretiererlauf und kehren Sie wieder auf Betriebssystemebene zurück:

```
Main>:quit
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

1) Hello, World!

```
main = putStrLn "Hello, World!"
```

...ein Beispiel für ein Programm mit Ein-/Ausgabeoperation.

Die Deklaration von `putStrLn`, nicht gänzlich selbsterklärend:

```
putStrLn :: String -> IO ()
putStrLn "Hello, World!"
```

Allerdings: Die Java-Entsprechung

```
class HelloWorld {
  public static void main (String[] args) {
    System.out.println("Hello, World!"); } }
```

...bedarf auch einer weiter ausholenden Erläuterung.

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

2) !: Die Fakultätsfunktion (i)

$$! : \mathbb{N} \to \mathbb{N}$$

$$\forall\, n \in \mathbb{N}.\ n! = \begin{cases} 1 & \text{falls } n = 0 \\ n * (n-1)! & \text{sonst} \end{cases}$$

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

...ein Beispiel für eine rekursive Funktionsdefinition.

Aufrufe:

```
fac 0 ->> 1    fac 3 ->> 6      fac 6  ->> 720
fac 1 ->> 1    fac 5 ->> 120    fac 10 ->> 3.628.800
```

Lies: "Die Auswertung des Ausdrucks/Aufrufs `fac 5` liefert den Wert `120`; der Ausdruck/Aufruf `fac 5` hat den Wert `120`."

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

2) !: Die Fakultätsfunktion (ii)

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

Funktionale Programmierung mag es kurz und knackig, prägnant und konzis, ohne kryptisch zu sein. Auch Haskell hat hierfür ein Angebot.

Alternative Schreibweise:

```
fac :: Integer -> Integer
fac n
  | n == 0    = 1                    (| für (oder) wenn)
  | otherwise = n * fac (n - 1)      (otherwise ->> True)

fac :: Integer -> Integer       (Diese Variante nur zur
fac n                                 Illustration von |)
  | n == 0 || n == 1 = 1          ((||) logisches oder)
  | n == 2           = 2
  | otherwise        = n * fac (n - 1)
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

2) !: Die Fakultätsfunktion (iii)

Eine zweite weitere Schreibweise, musterbasiert:

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

Weitere alternative Implementierungen:

```
fac :: Integer -> Integer
fac n = foldl (*) 1 [1..n]      ([1..3] ->> [1,2,3],
                                 [1..0] ->> [],
                 foldl (*) 1 [1..0] ->> foldl (*) 1 [] ->> 1)
```

```
fac :: Integer -> Integer
fac n = product [1..n]          (product = foldl (*) 1)
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

2) !: Die Fakultätsfunktion (iv)

Zwei einfache Formen der Fehlerbehandlung:

```
fac′ :: Integer -> Integer
fac′ n
 | n == 0     = 1
 | n > 0      = n * fac′ (n - 1)
 | otherwise = error "Arg. unzulässig"  (sog. Panikmodus)

fac″ :: Integer -> Integer
fac″ n
 | n == 0     = 1
 | n > 0      = n * fac″ (n - 1)
 | otherwise = n                  (sog. Auffangwertrückgabe)
```

Aufrufe:

```
fac′ 5     ->> 120                    fac″ 5     ->> 120
fac′ 0     ->> 1                      fac″ 0     ->> 1
fac′ (-5) ->> "Arg. unzulässig"       fac″ (-5) ->> -5
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

3) Euklidischer Algorithmus (3. Jhdt. v. Chr.)

...zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen $m, n \in \mathbb{N}_0$, $m \geq 0$, $n > 0$:

```
ggt :: Int -> Int -> Int  (Ganzz.-Typ, beschränkt)
ggt m n
  | n == 0 = m
  | n > 0  = ggt n (mod m n)

mod :: Int -> Int -> Int
mod m n
  | m < n  = m
  | m >= n = mod (m-n) n
```

...ein Beispiel für ein hierarchisches System von Funktionen.

Aufrufe:

```
ggt 25 15 ->> 5   ggt 48 60 ->> 12   mod 8 3 ->> 2
ggt 28 60 ->> 4   ggt 60 40 ->> 20   mod 9 3 ->> 0
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

4) Gerade/ungerade-Test für ganze Zahlen

```
isEven :: Integer -> Bool
isEven n
  | n == 0 = True
  | n > 0  = isOdd (n-1)
  | n < 0  = isOdd (n+1)

isOdd :: Integer -> Bool
isOdd n
  | n == 0 = False
  | n > 0  = isEven (n-1)
  | n < 0  = isEven (n+1)
```

...ein Beispiel für ein System wechselweise (oder indirekt) rekursiver Funktionen.

Aufrufe:

```
isEven 6    ->> True        isEven 9    ->> False
isOdd 6     ->> False       isOdd 9     ->> True
isOdd (-5)  ->> True        isEven (-8) ->> True
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

5) Längenberechnung von Listen

```
data [a] = []              (Informell: Datentypspez.
         | (a:[a])          für Listen, sprachintern
                                vordefiniert)

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

...ein Bsp. für eine parametrisch polymorphe Fkt. auf Listen.

Aufrufe:

```
length [2,4,6,8,10,12] ->> 6
length [(2,4),(6,8),(10,12)] ->> 3
length [(2,6),(8,10,12)] ->> 2
length ["Fkt.","Prog.","macht","Spass"] ->> 4
length [‘a‘,‘b‘,‘c‘] ->> 3
length [isOdd,isEven,isEven,isOdd,isOdd] ->> 5
length [] ->> 0
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

6) Umkehren von Zeichenreihen

```
type Zeichenreihe = [Char]                    (Typsynonym)
revertiere :: Zeichenreihe -> Zeichenreihe
revertiere ""     = ""     ("" leere Zeichenreihe)
revertiere (z:zs) = (revertiere zs) ++ [z]
```

++ Listenkonkatenator

...ein Bsp. für eine Funktion über selbstgewähltem sprechenden Typnamen, Zeichenreihe, statt [Char], list of characters, statt:

```
revertiere :: [Char] -> [Char]
```

Aufrufe:

```
revertiere ""          ->> ""
revertiere "stressed"  ->> "desserts"
revertiere "desserts"  ->> "stressed"
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

7) Transformieren von Listen

...durch Anwendung einer Fkt. auf alle Elemente einer Liste:

```
map :: (a -> b) -> [a] -> [b]        (Fkt. als Arg.)
map _ [] = []
map f (x:xs) = (f x) : map f xs
```

Liste mit Kopf `x` *u. Rest* `xs`

`:` *sog. Listenkonstruktor*

...ein Beispiel für eine Funktion höherer Ordnung, für Funktionen als erstrangige Sprachelemente (engl. first class citizens).

Aufrufe:

```
map (2*) [1,2,3,4,5] ->> [2,4,6,8,10]
map (\x -> x*x) [1,2,3,4,5] ->> [1,4,9,16,25]
map (>3) [2,3,4,5] ->> [False,False,True,True]
map isEven [2,3,4,5] ->> [True,False,True,False]
map length ["functional","programming","is","fun"]
                                    ->> [10,11,2,3]
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

8) Addieren von Zahlen

```
(+) :: Num a => a -> a -> a   (Num sog. Typklasse)
```

...ein Beispiel für eine überladene Funktion.

Aufrufe:

```
(+) 2 3      ->> 5       (+ auf ganzen Z., Präfixop.)
2 + 3        ->> 5         (+ als Infixop. auf g.Z.)
(+) 2.1 1.4 ->> 3.5 (+ auf Gleitkommaz., Präfixop.)
2.1 + 1.4    ->> 3.5         (+ als Infixop. auf Gkz)
(+) 7.81 2   ->> 9.81    (automatische Typanpassung)
```

```
((+) 1) :: Integer -> Integer  (Inkrementfunktion)
```

```
inc :: Integer -> Integer
inc = (+) 1          (vgl. die Funktion '(binom 49)')
```

```
inc' :: Integer -> Integer
inc' = (+1)        ((+1) ein sog. Operatorabschnitt)
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

9) Binomialkoeffizientenberechnung (i)

...geben die Anzahl der Kombinationen k-ter Ordnung von n Elementen ohne Wiederholung an:

$$\binom{\cdot}{\cdot} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$\forall\, n, k \in \mathbb{N}.\ \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

```
binom' :: (Integer,Integer) -> Integer
binom' (n,k) = div (fac n) (fac k * fac (n-k))
```

...ein Beispiel für eine musterbasierte Funktionsdefinition mit hierarchischer Abstützung auf eine andere Funktion ('Hilfsfunktion'), hier die Fakultätsfunktion.

Aufrufe:

```
binom' (49,6) ->> 13.983.816
binom' (45,6) ->>  8.145.060
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

9) Binomialkoeffizientenberechnung (ii)

Es gilt:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

```
binom'' :: (Integer,Integer) -> Integer
binom'' (n,k)
  | k==0 || n==k = 1
  | otherwise    = binom'' (n-1,k-1) + binom'' (n-1,k)
```

...ein Beispiel für eine musterbasierte (kaskaden- oder baum-artig-) rekursive Funktionsdefinition.

Aufrufe:

```
binom'' (49,6) ->> 13.983.816
binom'' (45,6) ->>  8.145.060
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

9) Binomialkoeffizientenberechnung (iii)

Uncurryfiziert

```
binom' :: (Integer,Integer) -> Integer
binom' (n,k) = div (fac n) (fac k * fac (n-k))
```

Curryfiziert

```
binom :: Integer -> (Integer -> Integer)
binom n k  = div (fac n) (fac k * fac (n-k))
```

Aufrufe:

```
binom' (49,6) ->> 13.983.816
binom' (45,6) ->>  8.145.060
binom 49 6  ->> 13.983.816
binom 45 6  ->>  8.145.060
binom 49
binom 45 ...sind ebenfalls zulässige Ausdrücke!
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

9) Binomialkoeffizientenberechnung (iv)

Die Aufrufe

```
binom 49
binom 45
```

...sind gültige Ausdrücke von einem funktionalen Wert:

```
(binom 49) :: Integer -> Integer
(binom 45) :: Integer -> Integer
```

...und repräsentieren die Funktionen '49_über_k' (entsprechend 'k_aus_49') und '45_über_k" (entsprechend 'k_aus_45'):

$$\binom{49}{\cdot} : \mathbb{N} \to \mathbb{N} \qquad \binom{45}{\cdot} : \mathbb{N} \to \mathbb{N}$$

$$\forall\, k \in \mathbb{N}.\ \binom{49}{k} = \frac{49!}{k!(49-k)!} \qquad \forall\, k \in \mathbb{N}.\ \binom{45}{k} = \frac{45!}{k!(45-k)!}$$

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

9) Binomialkoeffizientenberechnung (v)

In der Tat können wir als Funktionen definieren:

```
k_aus_49 :: Integer -> Integer
k_aus_49 k = binom 49 k

k_aus_45 :: Integer -> Integer
k_aus_45 k = binom 45 k
```

...und punktfrei (d.h., argumentlos) noch knapper:

```
k_aus_49 :: Integer -> Integer
k_aus_49 = binom 49

k_aus_45 :: Integer -> Integer
k_aus_45 = binom 45
```

Aufrufe:

```
k_aus_49 6 ->> binom 49 6 ->> 13.983.816
k_aus_45 6 ->> binom 45 6 ->>  8.145.060
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

10) Sieb des Eratosthenes (276-194 v.Chr.) (i)

...zur Berechnung des unendlichen Stroms der Primzahlen:

1. Schreibe alle natürlichen Zahlen ab 2 hintereinander auf.
2. Die kleinste nicht gestrichene Zahl in dieser Folge ist eine Primzahl. Streiche alle Vielfachen dieser Zahl.
3. Wiederhole Schritt 2 mit der nächstkleinsten noch nicht gestrichenen Zahl.

Nach Schritt 1:

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19...

Nach Schritt 2 für Zahl 2:

2 3 5 7 9 11 13 15 17 19...

Nach Schritt 2 für Zahl 3:

2 3 5 7 11 13 17 19...

usw.

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

10) Das Sieb des Eratosthenes (ii)

```
primes :: [Integer]
```
Zahlenstromtyp

(primes, der (Prim-) Zahlenstrom als Integer-Liste)

```
primes = sieve [2..]
```
Strom der nat. Zahlen ab 2

(leistet Schritt 1)

```
sieve :: [Integer] -> [Integer]
```
Argumentstromtyp *Resultatstromtyp*

```
sieve (x:xs) = x : sieve [y | y <- xs, mod y x > 0]
```
Argumentstrom *Resultatstrom*

(leistet Schritt 2 für die Zahlen 2, 3, 5, 7, 11, usw.)

...ein Beispiel für die Programmierung mit Strömen.

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

10) Das Sieb des Eratosthenes (iii)

```
primes :: [Integer]
```

(Prim−) Zahlenstromtyp

```
sieve :: [Integer]      ->      [Integer]
```

Argumentstromtyp — *Resultatstromtyp*

```
sieve (x:xs)          = x : sieve [y | y <- xs, mod y x > 0]
```

Strom der nat. Zahlen ab 2 als Argument — *Strom der Primzahlen als Resultat*

Strom der nat. Zahlen ab 2

```
primes = sieve [2..]
```

Strom der Primzahlen

Aufruf:

```
primes ->> sieve [2..] ->> 2 : sieve [3,5..]
 ->> ... ->> [2,3,5,7,11,13,17,19,23,29,31,37,41,...
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

10) Das Sieb des Eratosthenes (iv)

Im Überblick und (fast) ohne Farbspiele:

```
primes :: [Integer]
primes = sieve [2..]

sieve :: [Integer] -> [Integer]
sieve (x:xs) = x : sieve [y | y <- xs, mod y x > 0]
```

Aufrufe:

```
primes ->> [2,3,5,7,11,13,17,19,23,29,31,37,41,...
take 5 primes ->> [2,3,5,7,11]
drop 3 primes ->> [7,11,13,17,19,23,29,31,37,41,...
take 3 (drop 3 primes) ->> [7,11,13]
primes!!0 ->> 2                    ((!!) Zugriffsoperator)
primes!!1 ->> 3
primes!!5 ->> 13
```

Vortrag I
Präludium
Teil I
Kap. 1
1.1
1.1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Im Rückblick: Die ersten zehn Beispiele (1)

...jedes der Beispiele zeigt etwas, das funktionales Programmieren ausmacht:

1. Ein- und Ausgabe
   - *Hello, World!*
2. Rekursive Funktionen
   - !: Die Fakultätsfunktion
3. Hierarchische Systeme von Funktionen
   - Euklidischer Algorithmus
4. Systeme wechselweise rekursiver Funktionen
   - Gerade/ungerade-Test für ganze Zahlen
5. Parametrisch polymorphe Funktionen
   - Längenberechnung von Listen

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Im Rückblick: Die ersten zehn Beispiele (2)

6. Funktionen über sprechenden Typnamen
   - Umkehren von Zeichenreihen
7. Fkt. höherer Ordnung, Fkt. als 'Bürger erster Klasse'
   - Transformieren von Listen
8. Überladene Funktionen
   - Addieren von Zahlen
9. Musterbasierte, curryfizierte und uncurryfizierte Funktionsdefinitionen, partiell ausgewertete Funktionen
   - Binomialkoeffizientenberechnung
10. Stromprogrammierung
    - Das Sieb des Eratosthenes

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Zusammenfassend halten wir fest

Funktionale Programme sind

- Systeme (wechselweise) rekursiver Funktionsvorschriften (oder Rechenvorschriften).

Funktionen

- sind zentrales Abstraktionsmittel in funktionalen Programmen (wie Prozeduren (Methoden) in prozeduralen (objektorientierten) Programmen).

Funktionale Programme

- werten Ausdrücke aus. Das Resultat dieser Auswertung ist ein Wert eines bestimmten Typs. Dieser Wert kann elementar oder funktional sein; er ist die Bedeutung, die Semantik des Ausdrucks.

Der kürzeste Programmiererwitz:
Jetzt kann ich's.

unbekannt

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Übung

Es ist nicht genug zu wissen, man muss auch anwenden; es ist nicht genug zu wollen, man muss auch tun.

Johann Wolfgang von Goethe (1749-1832)

1. Implementieren Sie die 10 Beispielprogramme aus Kapitel 1.1.1 in einer Programmiersprache Ihrer Wahl, z.B. Java, und vergleichen Sie sie miteinander. Welche Unterschiede gibt es? In der Länge? Im Aufwand (konzeptuell, programmiertechnisch)? In der Verständlichkeit? In der Performanz? Warum?
2. Probieren Sie die Programme auch in Haskell aus!

Programmieren ist wie schwimmen. Man kann jahrelang zusehen, ohne es zu lernen.

unbekannt

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Drei Buchempfehlungen als Lesevorschläge

1. Sie sind mit objektorientierter Programmierung groß geworden und fühlen sich dort heimisch?

   Haskell: Eine Einführung für Objektorientierte von Ernst-Erich Doberkat könnte Ihre Wahl sein.

2. Sie möchten die Welt funktionaler Programmierung zugleich mit Beispielen weiterer funktionaler Sprachen erkunden?

   Funktionale Programmierung in OPAL, ML, Haskell und Gofer von Peter Pepper bietet sich als Ihre Wahl an.

3. Sie suchen ein Buch, das möglichst weite Teile der Vorlesung überstreicht?

   Haskell: The Craft of Functional Programming von Simon Thompson könnte sich für Sie lohnen.

Lege multum, non multa!
Lies viel, nicht vieles!

Plinius der Jüngere (um 61 - um 113 n.Chr.)
röm. Beamter und Schriftsteller
beschrieb den Ausbruch des Vesuvs im Jahr 79 n.Chr.

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Kapitel 1.4

Leseempfehlungen

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Basisleseempfehlungen für Kapitel 1 (1)

Richard Bird. *Thinking Functionally with Haskell*. Cambridge University Press, 2015. (Kapitel 1, What is functional programming? Kapitel 2.1, A session with GHCi)

Ernst-Erich Doberkat. *Haskell: Eine Einführung für Objektorientierte*. Oldenbourg Verlag, 2012. (Kapitel 1, Erste Schritte; Anhang A, Zur Benutzung des Systems)

Chris Done. *Try Haskell*. Online Hands-on Haskell Tutorial. tryhaskell.org.

Marco Block-Berlitz, Adrian Neumann. *Haskell Intensivkurs*. Springer-V., 2011. (Kapitel 1, Motivation und Einführung)

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Basisleseempfehlungen für Kapitel 1 (2)


Manuel Chakravarty, Gabriele Keller. *Einführung in die Programmierung mit Haskell*. Pearson Studium, 2004. (Kapitel 1, Einführung; Kapitel 2, Programmierumgebung; Kapitel 4.1, Rekursion über Zahlen; Kapitel 6, Die Unix-Programmierumgebung)


Neal Ford. *Functional Thinking: Why Functional Programming is on the Rise*. IBM developerWorks, 11 pages, 2013. `www.ibm.com/developerworks/java/library/j-ft20/j-ft20-pdf.pdf`


Paul Hudak, Joseph Fasel, John Peterson. *A Gentle Introduction to Haskell*. Technischer Bericht, Yale University, 1996. `https://www.haskell/org/tutorial`


Mihai Maruseac. *Haskell: A Language for Modern Times*. Crossroads, the ACM Magazine for Students 24(1):64-66, 2017.

Basisleseempfehlungen für Kapitel 1 (3)

Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley/Pearson, 3. Auflage, 2011. (Kapitel 1, Introducing functional programming; Kapitel 2, Getting started with Haskell and GHCi)

Hugs-Benutzerhandbuch. *The Hugs98 User Manual*. `https://www.haskell.org/hugs/pages/hugsman/index.html`

GHCi-Benutzerhandbuch. *Glasgow Haskell Compiler User's Guide*. `http://www.haskell.org/ghc/docs/latest/html/users_guide/ghci.html`

Haskells Standard-Präludium. `https://www.haskell.org/onlinereport/standard-prelude.html`

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Weiterführende Leseempfehlungen f. Kap. 1 (1)


Christopher Allen, Julie Moronuki. *Haskell Programming from First Principles*. ebook. `http://haskellbook.com`


Sergio Antoy, Michael Hanus. *Functional Logic Programming*. Communications of the ACM 53(4):74-85, 2010.


John W. Backus. *Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs*. Communications of the ACM 21(8):613-641, 1978.


Henri E. Baal, Dick Grune. *Programming Language Essentials*. Addison-Wesley, 1994. (Chapter 4, Functional Languages; Chapter 7, Other Paradigms)

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Weiterführende Leseempfehlungen f. Kap. 1 (2)

H. Conrad Cunningham. *Notes on Functional Programming with Haskell*. Course Notes, University of Mississippi, 2007. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.114.2822&rep=rep1&type=pdf (Chapter 1.2, Excerpts from Backus' 1977 Turing Award Address; Chapter 1.3, Programming Language Paradigms; Chapter 1.4, Reasons for Studying Functional Programming; Chapter 1.5, Objections Raised Against Functional Programming; Chapter 4, Using the Hugs Interpreter)

Hal Daumé III. *Yet Another Haskell Tutorial*. wikibooks.org-Ausgabe, 2007. https://en.wikibooks.org/wiki/Yet_Another_Haskell_Tutorial

Weiterführende Leseempfehlungen f. Kap. 1 (3)


Antonie J.T. Davie. *An Introduction to Functional Programming Systems using Haskell*. Cambridge University Press, 1992. (Kapitel 1.1, The von Neumann Bottleneck; Kapitel 1.2, Von Neumann Languages)


Frank DeRemer, Hans H. Kron. *Programming-in-the-Large vs. Programming-in-the-Small*. IEEE Transactions on Software Engineering 2(2):80-86, 1976.


Atze Dijkstra, Jeroen Fokker, S. Doaitse Swierstra. *The Architecture of the Utrecht Haskell Compiler*. In Proceedings of the 2nd ACM SIGPLAN Symposium on Haskell (Haskell 2009), 93-104, 2009.


Atze Dijkstra, Jeroen Fokker, S. Doaitse Swierstra. *UHC Utrecht Haskell Compiler*, 2009. www.cs.uu.nl/wiki/UHC

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Weiterführende Leseempfehlungen f. Kap. 1 (4)


Robert W. Floyd. *The Paradigms of Programming*. Turing Award Lecture. Communications of the ACM 22(8):455-460, 1979.


Bastiaan Heeren, Daan Leijen, Arjan van IJzendoorn. *Helium, for Learning Haskell*. In Proceedings of the ACM SIGPLAN 2003 Haskell Workshop (Haskell 2003), 62-71, 2003.


Konrad Hinsen. *The Promises of Functional Programming*. Computing in Science and Engineering 11(4):86-90, 2009.


C.A.R. Hoare. *Algorithm 64: Quicksort*. Communications of the ACM 4(7):321, 1961.


C.A.R. Hoare. *Quicksort*. The Computer Journal 5(1):10-15, 1962.

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Weiterführende Leseempfehlungen f. Kap. 1 (5)


Paul Hudak. *Conception, Evolution and Applications of Functional Programming Languages*. Communications of the ACM 21(3):359-411, 1989.


John Hughes. *Why Functional Programming Matters*. The Computer Journal 32(2):98-107, 1989.


Graham Hutton. *Programming in Haskell*. Cambridge University Press, 2. Auflage, 2016. (Kapitel 1, Introduction; Kapitel 2, First Steps)


Arjan van IJzendoorn, Daan Leijen, Bastiaan Heeren. *The Helium Compiler*. www.cs.uu.nl/helium.


Mark P. Jones, Alastair Reid et al. *The Hugs98 User Manual*, 1999. `www.haskell.org/hugs`

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Weiterführende Leseempfehlungen f. Kap. 1 (6)


Jerzy Karczmarczuk. *Scientific Computation and Functional Programming*. Computing in Science and Engineering 1(3):64-72, 1999.


Donald E. Knuth. *Literate Programming*. The Computer Journal 27(2):97-111, 1984.


Konstantin Läufer, George K. Thiruvathukal. *The Promises of Typed, Pure, and Lazy Functional Programming: Part II*. Computing in Science and Engineering 11(5):68-75, 2009.


Bruce MacLennan. *Functional Programming: Practice and Theory*. Addison-Wesley, 1990.

Weiterführende Leseempfehlungen f. Kap. 1 (7)


Martin Odersky. *Funktionale Programmierung*. In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 599-612, 2006. (Kapitel 5.1, Funktionale Programmiersprachen; Kapitel 5.2, Grundzüge des funktionalen Programmierens)


Bryan O'Sullivan, John Goerzen, Don Stewart. *Real World Haskell*. O'Reilly, 2008. (Kapitel 1, Getting Started)


Peter Pepper. *Funktionale Programmierung in OPAL, ML, Haskell und Gofer*. Springer-V., 2. Auflage, 2003. (Kapitel 1, Was die Mathematik uns bietet; Kapitel 2, Funktionen als Programmiersprache)


Peter Pepper, Petra Hofstedt. *Funktionale Programmierung: Sprachdesign und Programmiertechnik*. Springer-V., 2006. (Kapitel 1, Grundlagen der funktionalen Programmierung)

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Weiterführende Leseempfehlungen f. Kap. 1 (8)

Tomas Petricek, Jon Skeet. *Real World Functional Programming: With Examples in F# and C#*. Manning Publications Co., 2009. (Chapter 1, Thinking differently; Chapter 2, Core concepts in functional programming)

Simon Peyton Jones (Hrsg.). *Haskell 98: Language and Libraries. The Revised Report*. Cambridge University Press, 2003. `www.haskell.org/definitions`. (Kapitel 8, Standard Prelude; Kapitel 8.1, Module `Prelude`)

Chris Sadler, Susan Eisenbach. *Why Functional Programming?* In *Functional Programming: Languages, Tools and Architectures*, Susan Eisenbach (Hrsg.), Ellis Horwood, 9-20, 1987.

Neil Savage. *Using Functions for Easier Programming*. Communications of the ACM 61(5):29-30, 2018.

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Weiterführende Leseempfehlungen f. Kap. 1 (9)

Curt J. Simpson. *Experience Report: Haskell in the "Real World": Writing a Commercial Application in a Lazy Functional Language*. In Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming (ICFP 2009), 185-190, 2009.

Simon Thompson. *Where Do I Begin? A Problem Solving Approach in Teaching Functional Programming*. In Proceedings of the 9th International Symposium on Programming Languages: Implementations, Logics, and Programs (PLILP'97), Springer-Verlag, LNCS 1292, 323-334, 1997.

Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley/Pearson, 2. Auflage, 1999. (Kapitel 1, Introducing functional programming; Kapitel 2, Getting started with Haskell and Hugs)

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Weiterführende Leseempfehlungen f. Kap. 1 (10)

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Reinhard Wilhelm, Helmut Seidl. *Compiler Design – Virtual Machines*. Springer-V., 2010. (Kapitel 3, Functional Programming Languages; Kapitel 3.1, Basic Concepts and Introductory Examples)

Interview mit John Hughes über 'Funktionale Programmierung und Haskell'.
`https://www.youtube.com/watch?v=LnX3B9oaKzw`

Simon Peyton Jones. Escape from the Ivory Tower: The Haskell Journey. Eingeladener Hauptvortrag, 2017.
`https://www.youtube.com/watch?v=re96UgMk6GQ`

Weiterführende Leseempfehlungen f. Kap. 1 (11)

Welches Paradigma, welche Sprache sollte ich nutzen?


Peter J. Landin. *The next 700 Programming Languages*. Communications of the ACM 9(3):157-166, 1966.


Jeffrey S. Foster. *Shedding New Light on an Old Language Debate*. Communications of the ACM 60(10):90, 2017.


Baishakhi Ray, Daryl Posnett, Premkunar Devanbu, Vladimir Filkov. *A Large-Scale Study of Programming Languages and Code Quality in GitHub*. Communications of the ACM 60(10):91-100, 2017.


Rachel Harrison, L. G. Smaraweera, Mark R. Dobie, Paul H. Lewis. *Comparing Programming Paradigms: An Evaluation of Functional and Object-Oriented Programs*. Software Engineering Journal 11(4):247-254, 1996.

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Und wenn Sie denken...

Quis leget haec?
Wer soll das [alles] lesen?

Persius (34 - 62 n.Chr.)
röm. Dichter

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

...denken Sie an Plinius!

Lege multum, non multa!
Lies viel, nicht vieles!

Plinius der Jüngere (um 61 - um 113 n.Chr.)
röm. Beamter und Schriftsteller

und diese drei Vorschläge:

1. Sie sind mit objektorientierter Programmierung groß geworden und fühlen sich dort heimisch?

   Haskell: Eine Einführung für Objektorientierte von Ernst-Erich Doberkat könnte Ihre Wahl sein.

2. Sie möchten die Welt funktionaler Programmierung zugleich mit Beispielen weiterer funktionaler Sprachen erkunden?

   Funktionale Programmierung in OPAL, ML, Haskell und Gofer von Peter Pepper bietet sich als Ihre Wahl an.

3. Sie suchen ein Buch, das möglichst weite Teile der Vorlesung überstreicht?

   Haskell: The Craft of Functional Programming von Simon Thompson könnte sich für Sie lohnen.

Vortrag I
Präludium
Teil I
Kap. 1
1.1
Rückblick
Übung
Drei Lesevorschläge
1.4
Hinweis
Aufgabe

Hinweis

...für das Verständnis von Vorlesungsteil I ist eine über den unmittelbaren Inhalt von Vortrag I hinausgehende weitergehende und vertiefende Beschäftigung mit dem Stoff nötig; siehe:

- vollständige Lehrveranstaltungsunterlagen

...verfügbar auf der Webseite der Lehrveranstaltung:

`http://www.complang.tuwien.ac.at/knoop/fp185A05_ws2021.html`

Aufgabe bis Dienstag, 13.10.2020

...selbstständiges Durcharbeiten von Teil I 'Einführung', Kap. 1 'Motivation' und von Leit- und Kontrollfragenteil I zur Selbsteinschätzung und als Grundlage für die umgekehrte Klassenzimmersitzung am 13.10.2020:

Vortrag, umgek. Klassenz.	Thema Vortrag	Thema umgek. Klassenz.
Di, 06.10.2020, 08:15-09:45	**Teil I**	**n.a. / Vorbesprechung**
Di, 13.10.2020, 08:15-09:45	**Teil II**	**Teil I**
Di, 27.10.2020, 08:15-09:45	Teil III	Teil II
Mi, 04.11.2020, 08:15-09:45	Teil IV	Teil III
Mi, 18.11.2020, 08:15-09:45	Teil V	Teil IV
Mi, 02.12.2020, 08:15-09:45	Teil VI	Teil V
Mi, 16.12.2020, 08:15-09:45	Teil VII	Teil VI

Zusätzlich: Nutzen Sie die beiden Läufe des Testsystems am Mittwoch, 14.10.2020, und Freitag, 16.10.2020, mit der

- unbeurteilten Abgabe der 'Teste das Testsystem'-Angabe!

Siehe dazu `Angabe_TdT` auf der LVA-Webseite!