
Anhang A

Lösungen der Aufgaben

Kapitel 2, Aufgabe 1

```
and, or :: Bool -> Bool -> Bool
and x y = x && y
or  x y = x || y

and, or :: Bool -> Bool -> Bool
and True True = True
and _ _       = False

or True _     = True
or _ True     = True
or _ _       = False
```

Kapitel 3, Aufgabe 2

```
greater :: Int -> Int -> Int
greater a b = if (a>b) then a else b
```

Kapitel 3, Aufgabe 4

```
nand :: Bool -> Bool -> Bool
nand True True = False
nand _ _       = True
```

NAND und NOR stellen jeweils eine vollständige Basis dar. Das bedeutet, dass alle logischen Funktionen durch eine Kombination mit einem der beiden Funktionen darstellbar sind.

Kapitel 4, Aufgabe 1

```

ggT :: Int -> Int -> Int
ggT a b
  | a>b      = ggT (a-b) b
  | a<b      = ggT a (b-a)
  | otherwise = a

```

Kapitel 4, Aufgabe 2

```

isPan :: Int -> Int -> Bool
isPan x y = "123456789" == sort digits
  where digits = (show x) ++ (show y) ++ (show (x*y))

euler32 :: Int
euler32 = sum (nub (pan1 ++ pan2))
  where
    pan1 = [x*y | x<-[1..9] ,y<-[x..9999] ,(isPan x y)]
    pan2 = [x*y | x<-[10..99] ,y<-[x..999] ,(isPan x y)]

```

Kapitel 4, Aufgabe 3

```

euler4 :: Integer
euler4 = foldr max 0 [x*y | x<-[100..999] ,
                        y<-[100..999] ,
                        isPalindrom (x*y)]
  where isPalindrom n = show n == (reverse.show) n

```

Kapitel 5, Aufgabe 1

```

skalarP :: [Int] -> [Int] -> Int
skalarP [] [] = 0
skalarP (x:xs) (y:ys) = x*y + skalarP xs ys
skalarP _ _ = error "Listen nicht gleich lang"

```

Kapitel 5, Aufgabe 2

```
last :: [a] -> a
last xs = xs!!(length xs - 1)
```

Kapitel 5, Aufgabe 3

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

Kapitel 5, Aufgabe 4

```
euler1 :: Int -> Int
euler1 = sum.teil
  where teil n = [x | x<-[1..n-1], mod x 3==0 || mod x 5==0]
```

Kapitel 5, Aufgabe 5

```
euler2 :: Integer
euler2 = (sum.(filter even).(takeWhile (<4000000))) fibo
  where fibo = 1:2:zipWith (+) fibo (tail fibo)
```

Kapitel 6, Aufgabe 1

```
tausche :: (a->b->c) -> (b->a->c)
tausche f a b = f b a

umdrehen = foldl (tausche (:)) []
summe    = foldl (+) 0
produkt  = foldl (*) 1
```

Kapitel 6, Aufgabe 2

```
machStringGross :: String -> String
machStringGross = map machGross
```

Kapitel 6, Aufgabe 3

```
import Data.List
import Data.Char

primes :: [Int]
primes = 2:3:primes'
  where
    1:p:candidates = [6*k+r | k <- [0..], r <- [1,5]]
    primes'         = p:filter isPrime candidates
    divides n p     = mod n p == 0

isPrime n = all(not .(\p->(mod n p)==0))
           $ takeWhile (\p -> p*p<=n) primes

mirp :: Integer
mirp = [x | x<-[2..], isPrime x, isPrime (toDigit.reverse.show) x]
```

Kapitel 6, Aufgabe 4

```
euler5 :: Integer
euler5 = foldl lcm 1 [1..20]
```

Kapitel 6, Aufgabe 5

```
import Char

euler6 = sum (map (digitToInt) (show (2^1000)))
```

Kapitel 7, Aufgabe 3

```
euler = head (pFactors 317584931803 [])
  where
    pFactors 1 xs = xs
    pFactors p xs = pFactors (div p (save p)) xs ++ [save p]
    save x       = head [y | y <- [2..], mod x y == 0]
```

Kapitel 7, Aufgabe 4

```
import List
import Char

zahlen = map digitToInt
      "73167176531330624919225119674426574742355349194934
      96983520312774506326239578318016984801869478851843
      85861560789112949495459501737958331952853208805511
      12540698747158523863050715693290963295227443043557
      66896648950445244523161731856403098711121722383113
      62229893423380308135336276614282806444486645238749
      30358907296290491560440772390713810515859307960866
      70172427121883998797908792274921901699720888093776
      65727333001053367881220235421809751254540594752243
      52584907711670556013604839586446706324415722155397
      53697817977846174064955149290862569321978468622482
      83972241375657056057490261407972968652414535100474
      82166370484403199890008895243450658541227588666881
      16427171479924442928230863465674813919123162824586
      17866458359124566529476545682848912883142607690042
      24219022671055626321111109370544217506941658960408
      07198403850962455444362981230987879927244284909188
      84580156166097919133875499200524063689912560717606
      05886116467109405077541002256983155200055935729725
      71636269561882670428252483600823257530420752963450 "
```

```
euler = print.maximum.map (product.take 5).tails $ zahlen
```

Kapitel 8, Aufgabe 2

```
import Data.List

primes :: [Int]
primes = 2:3:primes'
```

```

where
  1:p:candidates = [6*k+r | k <- [0..], r <- [1,5]]
  primes'       = p:filter isPrime candidates
  divides n p   = mod n p == 0

isPrime n = all(not .(\p->(mod n p)==0)) $
            takeWhile (\p -> p*p<=n) primes

primefactors :: Int -> [Int]
primefactors n = primefactors' n n [] primes where
  primefactors' n m l (p:ps)
    | p>n      = l
    | mo == 0  = primefactors' n d (p:l) (p:ps)
    | otherwise = primefactors' n n l ps where
      (d,mo) = divMod m p

numOfFactors n = product (map ((+1).length)
                              (group (primefactors n)))
triangleNumber n = n*(n+1) 'div' 2

-- Achtung: Codezeile umgebrochen
euler12 = triangleNumber (head (filter (\n -> 500 <
                                         (numOfFactors.triangleNumber) n) [1..]))

```

Kapitel 8, Aufgabe 3

```
euler15 = div product [21..40] product [2..20]
```

Kapitel 9, Aufgabe 5

```

fac n = snd (until ((>n) . fst) (\(i,m) -> (i+1, i*m)) (1,1))
p34 = sum liste
      where liste = [x|x<- [2..1000000],
                    x==(sum.map (fac.digitToInt).show) x]

```

Kapitel 10, Aufgabe 1

```

buSort :: Ord a => (a -> a -> Bool) -> [a]->[[a]]
buSort f = bSort' [] where

```

```

bSort' buckets [] = buckets
bSort' buckets (x:xs) = bSort' (insert x buckets) xs
insert x [] = [[x]]
insert x ((e:b):bs)
  | f x e && f e x = (x:e:b):bs
  | f x e = [x] : (e:b) : bs
  | otherwise = (e:b) : (insert x bs)

```

Kapitel 16, Aufgabe 2

```

type Knoten      = Int
type AdjListe e  = DiffArray Knoten [(Knoten,e)]
type Kante e     = ((Knoten,Knoten),e)
newtype ListGraph e = G (AdjListe e)

-- Übung
erzeug :: [Knoten] -> [Kante e] -> ListGraph e
erzeug [] _ = G (array (0,0) [])
erzeug k l = G $ accumArray merge [] bounds edges where
  bounds = (lower,upper)
  lower  = minimum k
  upper  = maximum k
  edges  = map (\((a,b),e) -> (a,(b,e))) l
  merge l (n,e)
    | isJust (find (\(m,_) -> n==m) l) = 1
    | otherwise                        = (n,e):l

adj :: Knoten -> Knoten -> ListGraph e -> Bool
adj u v (G arr) = elem v $ map fst (arr!u)

adjL :: Knoten -> ListGraph e -> [(Knoten,e)]
adjL u (G arr) = arr!u

anzKnoten (G arr) = snd.bounds $ arr

istLeer g = 0 == anzKnoten g

knoten :: ListGraph e -> [Knoten]
knoten (G arr) = indices arr

kanten :: ListGraph e -> [Kante e]
kanten g = concatMap (\k -> map (\(n,e) -> ((k,n),e))
  (adjL k g)) (knoten g)

label :: Knoten -> Knoten -> ListGraph e -> Maybe e
label u v (G arr) = fmap snd $ find (\(v',_) -> v==v') (arr!u)

setzLabel :: e -> Knoten -> Knoten -> ListGraph e -> ListGraph e
setzLabel l u v (G arr) = G (arr//[u,list]) where

```

```
list = foldr update [] (arr!u)
update n@(v',_) r
  | v' == v = (v',l):r
  | otherwise = n:r
```

Kapitel 16, Aufgabe 6

```
import Graph
import Data.List

topSort :: ListGraph e -> [Knoten]
topSort g = reverse $ topSort' g where
  topSort' g
    | istLeer g = []
    | otherwise = let s = senke g in s : topSort'
                  (loeschKnoten s g)

-- Übung
topSort' :: ListGraph e -> [Knoten]
topSort' = reverse.unfoldr f where
  f g
    | istLeer g = Nothing
    | otherwise = let s = senke g in Just
                  (s, loeschKnoten s g)

senke g = case find (\k -> null (adjL k g)) (knoten g) of
  Nothing -> error "keine Senke gefunden"
  Just x   -> x

-- Achtung: Codezeile umgebrochen
loeschKnoten s g = erzeug (filter (/=s) (knoten g))
  [(n,m),e | ((n,m),e)<-kanten g, n /=s && m /= s]

exampleGraph = erzeug [1..5] kanten
  where kanten = [(1,2),1], [(1,3),4], [(1,4),4], [(2,5),4],
                [(2,3),1], [(3,4),1], [(3,5),4], [(4,5),1]
```

Literaturverzeichnis

1. O’Sullivan B, Goerzen J, Stewart D (2009) Real World Haskell, O’Reilly-Verlag
2. Block M (2009) Java Intensivkurs – In 14 Tagen lernen Projekte erfolgreich zu realisieren, Springer-Verlag, 2. Aufl
3. Neumann A (2009) Programmieren im Kranich-Stil, Bachelorarbeit an der Freien Universität Berlin
4. Block M, Rojas R (2009) Local Contrast Segmentation to Binarize Images, The Third International Conference on Digital Society (ICDS 2009), ISBN:978-1-4244-3550-6, Vol. 1, No. 1, pp. 294–299, Cancun/Mexiko
5. Schöning U (2008) Theoretische Informatik - kurz gefasst, 5. Aufl, Spektrum Akademischer Verlag
6. O’Neill ME (2008) The Genuine Sieve of Eratosthenes, Under consideration for publication in J. Functional Programming, <http://www.cs.hmc.edu/~oneill/papers/Sieve-JFP.pdf>
7. von Luxburg U (2007) A Tutorial on Spectral Clustering, in Statistics and Computing, Vol. 17, No. 4, pp. 395–416
8. Staab F (2007) Logik und Algebra: Eine praxisbezogene Einführung für Informatiker und Wirtschaftsinformatiker, Oldenbourg-Verlag
9. Dankmeier D (2006) Grundkurs Codierung: Verschlüsselung, Kompression, Fehlerbeseitigung, 3. Aufl, Vieweg-Verlag
10. Lang HW (2006) Algorithmen: in Java, 2. Aufl, Oldenbourg-Verlag
11. Bauer FL (2005) PYTHAGORÄISCHE TRIPEL, Informatik-Spektrum, Vol. 28, No. 5, pp. 417–423, Springer-Verlag
12. Kirmse A (Hrsg) (2004) Spieleprogrammierung Gems 4, Hanser-Verlag
13. Schulz R-H (2003) Codierungstheorie: Eine Einführung, 2. Aufl, Vieweg+Teubner
14. Okasaki C (2003) Fun with binary heap trees, The Fun of Programming, pp. 1–16
15. Claessen K, Hughes J (2003) Specification-based testing with QuickCheck, The Fun of Programming, pp. 17–39
16. Jones SP (Ed) (2002) Haskell 98, Report, <http://www.haskell.org/onlinereport/>
17. Schöning U (2001) Algorithmik, ISBN-13: 978-3827410924, Spektrum Akademischer Verlag
18. Pope B (2001) A tour of the Haskell Prelude, unpublished, <http://www.cs.mu.oz.au/~bjpop/>
19. Siefkes D (2000) Historische Paradigmenwechsel in der Informatik, Skript zur gleichnamigen Veranstaltung, TU Berlin, SoSe 2000
20. Hudak P, Peterson J, Fasel J (2000) A gentle introduction to Haskell Version 98, unpublished, <http://www.haskell.org/tutorial/>
21. Cormen TH, Leiserson CE, Rivest RL (2000) Introduction to Algorithms, MIT-Press
22. Bratko I (2000) Prolog Programming for Artificial Intelligence, 3. Aufl, Addison-Wesley-Verlag
23. Heun V (2000) Grundlegende Algorithmen, Vieweg-Verlag

24. Gibbons J, Jones G (1998) The Under-Appreciated Unfold, Proceedings of the third ACM SIGPLAN international conference on Functional programming, pp. 273–279, United States
25. Knuth DE (1998) The Art of Computer Programming, Vol. 2, ISBN 0-201-89684-2, Addison-Wesley-Verlag
26. Brodal GS (1996) Worst-Case Efficient Priority Queues, Proceedings of the seventh annual ACM-SIAM Symposium on Discrete Algorithms, pp. 52–58
27. Okasaki C (1996) Purely Functional Data Structures, Doktorarbeit, Carnegie Mellon University
28. Okasaki C (1995), Purely Functional Random-Access Lists, Functional Programming Languages and Computer Architecture, pp. 86–95
29. Broder A, Stolfi J (1986) Pessimistic Algorithms and Simplexity Analysis, ACM SIGACT News
30. Barendregt HP (1985) The Lambda Calculus, Its Syntax and Semantics, North Holland, Revised edition
31. Gardner M (1985) Magic numbers of Dr. Matrix, Prometheus Books
32. Tarjan RE (1985) Amortized computational complexity, SIAM Journal on Algebraic and Discrete Methods, Vol. 6, No. 2, pp. 306–318
33. Church A (1985) The calculi of lambda-conversion, Princeton University Press
34. Bayer R (1972) Symmetric binary B-Trees: Data structure and maintenance algorithms, Acta Informatica. 1, pp. 290–306
35. Adelson-Velsky GM, Landis EM (1962) An algorithm for the organization of information (Englische Übersetzung aus Doklady Akademii Nauk SSSR 146), Soviet Math. Doklady. 3, pp. 1259–1263
36. McCarthy J (1960) Recursive functions of symbolic expressions and their computation by machine, Part I, Communications of the ACM archive, Vol. 3, No. 4, pp. 184–195
37. Church A, Rosser JB (1936) Some properties of conversion, Transactions of the American Mathematical Society, Vol. 39, No. 3, pp. 472–482
38. Webseite des Euler-Projekts: <http://projecteuler.net/>
39. Webseite zu GHC: <http://www.haskell.org/ghc>
40. Webseite zur Funktionenübersicht: <http://www.haskell.org/ghc/docs/latest/html/libraries>
41. Webseite zu Opal: <http://user.cs.tu-berlin.de/~opal/>
42. Webseite zu Meta Language: <http://www.smlnj.org/>
43. Webseite zu Gofer: <http://web.cecs.pdx.edu/~mpj/goferarc/index.html>
44. Webseite zu Miranda: <http://miranda.org.uk/>
45. Webseite zu Scala: <http://www.scala-lang.org/>
46. Webseite zu Notepad++: <http://notepad-plus.sourceforge.net/de/site.htm>
47. Webseite zu TextWrangler: <http://www.barebones.com/products/textwrangler/>
48. Webseite zum Kate-Editor: <http://kate-editor.org/>
49. Webseite der Helium-IDE: <http://www.cs.uu.nl/wiki/Helium>
50. Wikibook zur Datenkompression: <http://de.wikibooks.org/wiki/Datenkompression>
51. Haskell-Funktionen: <http://www.zvon.org/other/haskell/Outputglobal/index.html>
52. Webseite Haskellprojekte: <http://hackage.haskell.org/packages/archive/pkg-list.html>
53. Projektwebseite Frag: <http://haskell.org/haskellwiki/Frag>
54. Projektwebseite Monadius: http://www.geocities.jp/takascience/haskell/monadius_en.html
55. Haskell-Suchmaschine Hoogle: <http://www.haskell.org/hoogle/>
56. Dokumentation von Hugs: <http://cvs.haskell.org/Hugs/pages/documentation.htm>
57. Bibliotheken von GHC: <http://www.haskell.org/ghc/docs/latest/html/libraries/>
58. Standardklassen von Haskell:
<http://www.haskell.org/onlinereport/basic.html#standard-classes>
59. Monaden: <http://www.haskell.org/onlinereport/exps.html#sect3.14>
60. Steuerzeichen: <http://www.haskell.org/onlinereport/lexemes.html#sect2.6>
61. Übersicht zu Monaden: http://www.haskell.org/all_about_monads/html/introII.html
62. Buffering-Bug: <http://hackage.haskell.org/trac/ghc/ticket/2189>
63. Webseite Isabell: <http://www.cl.cam.ac.uk/research/hvg/Isabelle/>
64. Webseite Coq: <http://coq.inria.fr/>
65. Webseite QuickCheck: <http://www.cs.chalmers.se/~rjmh/QuickCheck/manual.html>

66. Wikipedia: <http://www.wikipedia.com>
67. Wikipedia Bärtierchen: <http://de.wikipedia.org/wiki/B%C3%A4rtierchen>
68. Wikipedia ASCII-Tabelle:
http://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange
69. Webseite zum Unicode: <http://www.unicode.org/>
70. Webseite zu verschiedenen Fakultätsfunktionen:
<http://www.willamette.edu/~fruehr/haskell/evolution.html>
71. Blog zu Monaden:
<http://blog.sigfpe.com/2006/08/you-could-have-invented-monads-and.html>
72. Webseite zu Parsec: <http://hackage.haskell.org/package/parsec>
73. Webseite zu den Alligatoren: <http://worrydream.com/AlligatorEggs/>
74. Webseite A Neighborhood of Infinity:
<http://blog.sigfpe.com/2006/08/you-could-have-invented-monads-and.html>

Sachverzeichnis

- 1000-stellige Zahl, 110
- 2-Tupel, 252
- Y-Kombinator, 260
- (| |), 18, 129
- (\$), 84
- (&&), 17, 154
- (*), 105
- (+), 105
- (+), 85
- (++), 162, 238
- (++), 55
- (.), 83
- (...), 84
- (/), 22
- (//), 140
- (/=), 101
- (/=), 103, 109
- (:), 51, 55
- (:), 169
- (<=), 101
- (==), 101
- (==), 103, 109
- (==>), 242
- (>>), 213
- (>>=), 213
- (>>=), 222
- (\Leftrightarrow), 19
- (\Rightarrow), 19
- (_), 28
- (\equiv), 19
- (\n), 225
- (-), 17, 239
- (∇), 19
- ($\bar{\nabla}$), 19
- (\vee), 18, 239
- (\wedge), 17
- [Char], 90
- Abarbeitungsformel, 169
- Abbruchkriterium, 80
- Abhängigkeiten, 205
- Ableitungsregeln, 107
- abs, 105
- Absorption, 20
- Abstrakter Datentyp, 137
- Abstraktionsgrad, 89
- Abstraktionsniveau, 111
- accent grave, 106
- Ackermann-Funktion, 45
- Addition, 11, 126, 256
- adjazent, 194
- Adjazenzliste, 195, 205, 207
- Adjazenzmatrix, 195, 207
- Adressbuch, 90, 114
- ADT, 137
- Akkumulator, 141
- Alan Turing, 260
- Algebra
 - boolesche, 24
- Algorithmus, 3, 121, 123
- Alonso Church, 246
- α -Konversion, 248
- Alter des Universums, 122
- Alternative, 243
- Analysemethoden, 128
- AND, 251
- AND, 17
- Anführungszeichen, 65
- ansonsten, 33
- Antivalenz, 18
- Apostroph, 65
- Applikation, 247, 249
- Äquivalenz, 18
 - semantisch, 19
- Arbitrary, 242

- arbitrary, 243
- Arithmetik, 254
- Arrayinhalt, 140
- Arrays, 137
 - dynamisch, 140
 - erstellen, 138
 - Grenzen, 139
 - statisch, 138
- ASCII-Code, 66
- Assoziativität, 35, 214
- asymptotisch, 124
- atomar, 15
- Aufzählungstypen, 101
- Ausdrücke
 - arithmetisch, 152
 - verschachtelt, 152
- Ausgaben, 224
 - Bildschirm, 226
- Auslesen
 - Dateien, 230
- Ausprobieren, 233
- Aussagen, 237
- Auswertung, 153
 - bedarfsgesteuert, 59
- Auswertungsreihenfolge, 250
- Automat, 220
- Automatische Listenerzeugung, 56, 63, 71, 73, 146
- Automatisches Aufzählen, 57
- Automatisches Auswerten, 108
- average-case, 127

- Bücherstapel, 142
- banker's method*, 159
- Bankiermethode, 159
 - modifiziert, 182
- Bärtierchen, 230
- Basisdatentyp, 26, 49
- Basisfunktionen
 - arithmetisch, 254
- Baum, 165
 - Balanceeigenschaften, 175
 - balanciert, 167
 - binär, 166
 - falten, 169
 - Höhe, 167
 - vollständig, 166
- Bäume
 - balanciert, 181, 201
 - verschmelzen, 167, 180
- Bedingung, 34
- benachbart, 194
- Berechenbarkeit, 245
- Berechenbarkeitstheorie, 45, 246

- best-case, 127
- β -Reduktion, 249
- Beweiskonzept, 235
- Bezeichner, 246
- Bibliotheken, 10, 66
- Binärbaum, 170, 178, 186, 242
 - vollständig, 187
- Bindungsstärke, 35, 38, 71, 84
- Bit, 16
- Blatt, 167, 173
- Blatt, 243
- BlockBuffering, 228
- Bool, 16, 241
- Boolean, 103
- Boolesche Algebra, 19, 250
- boolesche Terme, 239
- Bounded, 102
- Box, 93
- Breitensuche, 200
- Breitensuchenbaum, 200
- BubbleSort, 144
- Buchstaben, 228
- buckets*, 149
- BucketSort, 149
- Buffering, 227
- BufferModes, 228
- Byte, 24

- case, 31, 96
- Char, 23
- Church Turing These, 246
- Church-Kodierung, 254
- Church-Rosser-Theorem, 250
- class, 109
- Client-Server-Architekturen, 10
- Codeoptimierung, 46
- Comp, 107
- compare, 104
- Compiler, 3–5
- Computer, 3
- Computeralgebrasystem, 110
- Containerdatentyp, 141, 157
- Control.Monad, 223, 243
- Control.Monad.Reader, 220
- Control.Monad.State, 221
- Coq, 240
- Currying, 85, 86

- DAG, 204
- Dameproblem, 64
- data, 91, 153
- Data.List.sort, 241
- Data.Map, 263
- Data.Set, 263

- Datamining, 9
- Datenabstraktion, 49
- Datenbankanwendungen, 10
- Datenbanken, 9
- Datenkonstruktor, 91, 94
 - null-stellig, 92
- Datenstruktur, 4
 - einfach, 49
- Datentyp
 - Bool, 16
 - Boolesche Werte, 92
 - einfacher, 15
 - Either, 94
 - Maybe, 95
 - mehrere Felder, 96
 - Tupel, 94
- De Morgan, 20
- Debug-Ausgaben, 210
- Defaultimplementierungen, 103, 109
- Definitionen
 - lokal, 31, 32
- deriving, 92, 100
- Deutschland*, 231
- Dezimalzahl, 22
- Differentiation
 - numerisch, 105
 - symbolisch, 105
- directed acyclic graph*, 204
- Disjunktion, 18, 251
- divide-and-conquer*, 132
- Division, 267
- do-Notation, 214, 223, 266
 - Umwandlungsregeln, 214
- Doktorand, 178
- Dokumentation, 103
- Double, 23, 107
- Dreieckszahl, 117

- Echo, 224
- Editor, 3
- Effizienz, 32
- Eimer, 149
- Einfügen, 167
- Eingabekonsole, 50
- Eingaben, 224
 - Pufferung, 224
 - stream-basiert, 224
 - Tastatur, 227
- Eingabeparameter, 26
- Eingabepufferung, 227
- Eingabestrom, 225
- Eingabetypen
 - definieren, 211
- Einlesen
 - Dateien, 230
 - Einrückung, 32
 - Either, 95
 - Elementtypen, 50
 - else, 34
 - empty, 253
 - Endlosrekursion, 43
 - Entfaltung, 80
 - Entwicklungsumgebung, 4
 - Entwurfstechnik, 41, 98, 132
 - Enum, 101
 - Eq, 100, 103, 109
 - Eulerprojekt, 110
 - Eurobetrag, 236
 - Evaluationsreihenfolge, 154
 - Exemplar
 - zufällig, 242
 - Exklusiv-Oder, 18
 - Exponentialfunktion, 126
 - exponentiell, 127
 - Exportliste, 114

 - Fachterminovokabular, 166
 - faktoriell, 127
 - Fakultätsfunktion, 42, 128, 133, 236, 259
 - Fallunterscheidung, 262
 - Fallunterscheidungen, 30, 33, 34
 - False, 250
 - False, 16
 - Faltung, 73, 130, 261, 264
 - links mit Startwert, 78
 - links ohne Startwert, 79
 - rechts mit Startwert, 74
 - rechts ohne Startwert, 77
 - Unterschiede, 79
 - Farbinformationen, 219
 - Fehlerlokalisierung, 111
 - Fehlermeldung, 27, 99
 - Fehlersituation, 95
 - Fibonacci-Zahlen, 44, 67, 139
 - FIFO-Prinzip, 142
 - FilePath, 230
 - Filtereigenschaften, 73
 - Filtern, 72
 - First, 252
 - Fixpunktkombinatoren, 259
 - Flaschen, 222
 - Float, 23
 - Flugpläne, 198
 - Folgeliteratur, 10
 - Formel, 153
 - Rang, 239
 - Fünf-Euro-Münze, 236
 - Funktion, 25

(!), 55
a, 261
accumArray, 139
adj, 196
adjL, 196
amap, 140
and, 77
ansonsten, 33
anzKnoten, 197
app, 261, 263
arbitrary, 242
array, 139
assocs, 140
ausListe, 189
automat, 221
bSort, 146
bubble, 145
buSort, 149, 150
cat, 141
concat, 56, 76, 152
concatMap, 152
cons, 141, 189
curry, 86
damen, 65
dekrementiere, 29, 32, 83
delete, 143
deQ, 179
dequeue, 158, 160, 178, 179
diff', 108
div, 22, 23
dreheUm, 67
drittesElement, 63
drop, 62
einfuegen, 144
einheit, 211, 231
eintrag, 90
eintragMitNamen, 97
elem, 54, 129, 196, 238
elementAn, 186, 190
elementAt, 54
elems, 140
enqueue, 158, 160, 178, 179
enthalten, 54
error, 43
erstes, 52
erstesElement, 63, 70
erzeuge, 207
eval, 108, 153
evaluate, 261
f, 210, 212, 218, 224, 231
f', 210
fail, 213
fakul, 128
fakultaet, 42, 237
faltenLinks, 78
faltenLinks1, 79
faltenRechts, 74, 76
fassezusammen, 55
fibs, 139
filterListe, 72
find, 197, 205
finde, 172, 173
fold, 169
foldl, 78, 87, 131
foldl1, 79
foldr, 76, 87
foldr1, 77
formel, 32
fromInteger, 106
fromList, 183
fst, 63, 94
fuegEin, 172, 174
funktion1, 112
funktion2, 112
g, 210, 212, 218
g', 210
geld, 221
getChar, 227
getLine, 227
getNaechstes, 202
ggT, 46
greater, 39
h, 210, 212, 218
hangman, 228
head, 52
height, 167
hSetBuffering, 228
id, 224
import, 66
init, 54
inkrementiere, 29, 32, 83
insert, 149
intbool, 27
interact, 224
isInfix, 230
iSort, 144
istDrin, 129, 147
istKlein, 67
istLeer, 197
kanten, 197
kgV, 46
klammerTest, 154
klammerTest', 154
knoten, 197
kopf, 141, 189
l, 261
label, 197
laengeListe, 58, 72

- last, 54, 67
- leer, 158, 189
- leeresWBuch, 115
- letztes, 53, 67
- levelorder, 169
- liesFarben, 219
- lift, 211, 213
- lines, 225
- listeUmwandeln, 70
- loesche, 115, 172, 174
- loesche', 174
- loeschKnoten, 205
- makeStringGross, 87
- main, 226
- mal142, 70
- map, 72, 161, 170
- mapEntfaltung, 81
- mapFaltung, 76, 81
- mapListe, 71
- mappend, 217
- maxElement, 174, 175
- maybeApp, 265
- mconcat, 217
- mempty, 217
- merge, 148, 168, 179, 181
- minElement, 179
- minimum, 102, 133, 143
- minus, 35
- mod, 22, 23
- modify, 264
- mSort, 149
- mystic, 26
- n_damen, 64
- nachListe, 189
- nimm, 62
- nimmElement, 191
- nimmnicht, 62
- null, 141, 160, 179, 189
- ohneletztes, 53
- oneof, 243
- opNAND, 39
- or, 77
- parse, 266
- parseL, 266
- passt, 154
- plus, 35
- pop, 163
- pot, 132
- prettyPrint, 170
- primes, 141
- primzahlen, 61
- print, 226
- printWord, 228
- produkt, 74, 87
- prop_sorted, 241
- push, 163
- putChar, 226
- putStr, 226
- putStrLn, 66, 226
- qSort, 146
- quadrat, 39
- quadrante, 59
- queue, 160
- quickCheck, 241
- random, 212
- read, 100, 227
- readFile, 230
- readLn, 227
- reduce, 261
- rest, 52, 141, 189
- return, 213
- reverse, 130
- rSort, 150, 152
- rSort', 151
- scanl, 163
- senke, 205, 207
- setLabel, 197
- show, 100, 103, 163
- sicher, 65
- sieb, 61
- simplify, 110
- singleton, 167, 179, 189
- size, 167
- snd, 63, 94
- splitAtBrace, 265
- sSort, 143
- substitute, 262
- summe, 74, 87
- summeBaum, 170
- summeListe, 76
- sumTo, 46
- tail, 53
- take, 62
- teilen, 95
- toList, 183
- topSort, 205
- traverse, 202
- traverse', 202
- tueRein, 93
- umdrehen, 87, 130
- uncurry, 86
- unfold, 80
- unfoldr, 82, 207
- unlines, 225
- unzip3, 83
- unzippe, 83, 84
- updateAn, 186, 190
- v, 261

- verbinde, 211, 213, 231
- vollerName, 97
- writeFile, 230
- xor, 19, 30, 33, 37, 85
- zip, 82
- zip3, 83
- zippe, 82, 84
- zumQuadrat, 70
- zweiundvierzig, 27, 28
- Funktionen
 - ableiten, 105
 - als Argumente, 69
 - anonym, 70
 - Ausgabeparameter, 26
 - definieren, 26
 - Eingabeparameter, 26
 - höherer Ordnung, 69
 - monomorph, 102, 241
 - namenlos, 39
 - nicht-deterministisch, 222
 - nicht-monadisch, 243
 - polymorph, 102
 - positive, 122
 - Sichtbarkeit, 112
 - Signatur, 26
 - Signaturen zusammenfassen, 27
 - verketteten, 211
 - zu Operatoren, 37
 - zweistellig, 35, 38
- Funktionskörper, 247
- Funktionskomposition, 83, 114, 209, 210, 225
 - Klammerung, 84
- Funktionsname, 26, 258
- Ganzzahlquotient, 22
- Gauß-Summe, 130
- Gaußsche Summenformel, 235
- Geldbetrag, 188
- Geldrückgabe, 188
- Geldschein, 188
- Gemeinsamkeiten, 90
- Generatorfunktion, 80
- Geschwindigkeitsunterschiede, 140
- Getränkeautomat, 220
- Gewichte, 194
- GHC, 5
- Giuseppe Peano, 234
- Glasgow Haskell Compiler, 5
- Gleichheit, 101
- Grafik, 9
- Graph, 193, 222
 - gerichtet, 204
- Graphentheorie, 193
- Greedy-Strategie, 187
- Grenzen der Machbarkeit, 246
- Grenzwert, 125
- Großbuchstaben, 247
- Guards, 33, 34, 266
- Gültigkeitsbereich, 32
- Guthaben, 222
- Hangman, 228
- Haskell Brooks Curry, 75
- Haskell-Community, 5
- Haskell-Skript, 8
 - hs, 8, 25, 34
 - lhs, 9
- Haskellspezifikation, 26
- head, 51, 253
- Heapordnung, 178, 183
- Helium, 6
- Hilfsfunktion, 31
- Hintergrund, 198
- Histogramm, 139
- Hoogle, 26
- Hugs, 5, 6
 - Installationsordner, 10
 - Komponenten, 6
 - Systemfunktionen, 7
- Hugsbefehle
 - :?, 7
 - :also, 7
 - :browse, 7
 - :cd, 7
 - :edit, 7
 - :find, 7
 - :gc, 7
 - :info, 7
 - :kind, 94
 - :load, 7, 112, 139
 - :main, 7
 - :module, 7
 - :names, 7
 - :quit, 7
 - :reload, 7
 - :set, 7
 - :t, 26
 - :type, 7, 26
 - :version, 7
- Idempotenz, 20
- Identitätsfunktion, 211, 224
- if, 34
- if-then-else, 34, 215, 223
- Implikation, 242
- import, 113
- Importierung
 - Reihenfolge, 113

- in, 32
- Induktionsanker, 235
- Induktionsaxiom, 235
- Induktionsschritt, 235
- Induktionsvoraussetzung, 235
- infixl, 38
- infixr, 38
- Infixschreibweise, 35, 37
- inorder, 168, 186
- InsertionSort, 143
- Installation, 5
- instance, 103
- Instanzendefinitionen, 201
- Instanzieren
 - manuell, 102
- Int, 21, 107, 238
- Integer, 22, 106, 128
- Interfaces, 111
- Interpreter, 3–5, 264
- Intervallende, 59
- intuitiv berechenbar, 246
- Involution, 20
- IO, 226
- Isabelle, 240

- Jahreszeiten, 99
- Jedimeister Yoda, 152
- Just, 95

- Kante
 - gerichtet, 194
 - ungerichtet, 194
- Kanten, 165, 194
- Kate, 3
- Kategorientheorie, 209
- Kellerspeicher, 142
- kind*, 94
- Kinder, 166
- Klammertest, 153
- Klasse
 - Arbitrary, 242
 - Char, 65
 - Control.Monad, 223
 - Control.Monad.Reader, 220
 - Control.Monad.State, 221
 - Data.Char, 66
 - Eg, 150
 - Monad, 213
 - Monoid, 217
 - Num, 29
 - Queue, 157
 - Show, 163, 183, 191
 - SuchStrukt, 201
- Klassen
 - definieren, 109
 - Kleidung, 204
 - Kleinbuchstaben, 246
 - KLR, 169
 - Knoten, 165, 194
 - Knoten, 243
 - Kochrezept, 3
 - Kommentare, 8, 9, 34
 - Kommutativität, 20
 - Komplementarität, 20
 - Komplexität, 238
 - Konjunktion, 17, 251
 - Konkatenation, 55, 142
 - Konsole, 6, 7
 - konstant, 127
 - Konstanten, 106
 - konsumieren, 80
 - Konvention, 58, 112
 - Kopf-Rest-Prinzip, 52, 58
 - Korrektheit, 4, 152
 - Korrektheitsbeweis
 - unvollständig, 240
 - Kosten, 159
 - Kredite, 159, 162
 - Kreide, 198
 - kreisfrei, 194
 - Kreisfreiheit, 199
 - Kreuzungen, 199
 - kubisch, 127
 - Künstliche Spieler, 193

 - Labyrinth, 198, 207
 - λ -Abstraktionen, 247
 - λ -Ausdrücke
 - auswerten, 248
 - reguläre, 247
 - repräsentieren, 261
 - λ -Funktion, 247
 - λ -Interpreter, 260
 - Lambda-Kalkül, 245
 - λ -Kalkül, 2, 246
 - Lambda-Notation, 38
 - λ -Notation, 39
 - λ -Parser, 265
 - Landau-Symbole, 123
 - Laufzeit
 - O , 123
 - o , 123
 - exponentiell, 127
 - faktoriell, 127
 - konstant, 127
 - kubisch, 127
 - linear, 127
 - logarithmisch, 127

- Ω , 123
- ω , 123
- polylogarithmisch, 127
- polynomiell, 127
- quadratisch, 127
- Θ , 123
- Laufzeitanalyse, 121, 127
 - amortisiert, 159, 161, 181
- Laufzeiten
 - Übersicht, 127
- Lazy evaluation, 59, 128, 160, 223
- Leerzeichen, 32
- Lesbarkeit, 28
- let, 32
- let-in, 32
- levelorder, 169
- lexikographisch, 23, 64, 101
- liftM3, 243
- linear, 127
- LineBuffering, 228
- Linksassoziativität, 38
- Linksidentität, 214
- Linux, 5
- List, 141, 222
- List comprehensions, 56, 80
- Liste, 50, 137, 141, 158, 178, 252
 - automatisch erzeugen, 56, 58
 - Kopf, 51
 - Rest, 51
 - sortieren, 142
 - unendlich, 59, 71
 - verkettet, 185
 - zerlegen, 61
 - zusammenfassen, 55
- Listenfunktionen, 98, 189
 - rekursiv, 53
- Logarithmen, 126
- logarithmisch, 127
- Löschen, 167
- Lösungen
 - effizient, 122
- Lösungsstrategien, 193
- Mac OS, 5
- Main, 112
- Makros, 256
- Mapping, 71, 81
- Maschinencode, 3
- maxBound, 102
- Maybe, 95, 265
- Median, 148
- Mengenoperationen, 201, 263
- MergeSort, 148, 178
- Mikrowellenkuchen, 3
- minBound, 102
- Minimum, 102, 133
- Mirp-Zahlen, 87
- Mitschrift, 210
- Mittelwert, 129
- Modul, 112
 - A, 112
 - Adressbuch, 115
 - B, 112
 - C, 112
 - Data.Array.Diff, 140
 - Data.Array.IArray, 138
 - Data.IntMap, 175
 - Data.List, 205
 - Data.Map, 175
 - Data.Random, 212
 - Data.Set, 201
 - Eulerloesungen, 117
 - Test.QuickCheck, 241
 - TestAdressbuch, 116
 - Woerterbuch, 114
- Modul.Funktion, 113
- Modularisierung, 111
- module, 112
- Module definieren, 112
- Monad, 213
- Monade, 209
 - IO, 225
 - List, 222
 - Maybe, 265
 - Reader, 218
 - State, 220, 226
 - Writer, 217
- Monoid, 217
- Multiplikation, 126, 257
- Münze, 188
- Mustererkennung, 9
- n-Tupel, 62, 96
- Nachfolger, 234
- Nachfolgerfunktion, 254
- nachname, 98
- Namenskonflikte, 113
- NAND, 39
- Natürliche Zahlen, 110, 234, 254
- Navigationsgerät, 197
- negate, 105
- Negation, 17, 251
- Neutralität, 20
- newtype, 91, 93, 196, 201
- Nil, 253
- NoBuffering, 228
- NOR, 39
- Normal-Order-Reduction, 250

- Normalform, 250
- NOT, 17, 251
- not, 17
- Notepad++, 3
- Nothing, 95, 202, 230
- Num, 105

- OpenGL, 9
- Operator
 - (\$), 84
 - (+), 85
 - (++), 55
 - (.), 83
 - (...), 84
 - update, 140
- Operatorbaum, 106, 108
 - Höhe, 239
- Operatoren, 35
 - Assoziativität, 35
 - Bindungsstärke, 35
 - definieren, 35, 37
 - Infixschreibweise, 106
 - Signaturdefinition, 36
 - zu Funktionen, 36
- OR, 251
- OR, 18
- Ord, 100–102, 104, 241
- Ordnungseigenschaft, 173, 180
- otherwise, 33
- outputsensitiv, 176

- Paare
 - verschachtelt, 252
- Palindromzahl, 47
- pandigital, 47
- Parameterübergabe
 - implizit, 76
- Parsec, 265
- Parsefehler, 265
- Pattern matching, 30, 96, 266
 - case, 31
 - guards, 33
 - Zeilenweise, 30
- Peano-Axiome, 234
- Permutationen, 223
- Personenkette, 157
- Pferde, 243
- Pivotelement, 146, 175
- Platzhalter, 26, 258
- polylogarithmisch, 127
- Polymorphie, 106
 - eingeschränkt, 102
- Polynome, 126
 - berechnen, 107
 - polynomiell, 127
- postorder, 168, 206
- Potenzen, 126, 131, 134
- Potenzieren, 146
- Prädikat, 70, 72
- Präfix, 225
- Präfixschreibweise, 36
- Prelude, 10
- preorder, 168, 169, 186, 190
- Primfaktoren, 110
- Primzahl, 60, 87, 141
- Primzahlkandidaten, 60
- Prioritäten, 177
- Prioritätswarteschlange, 177
- Problemgrößen, 123
- Produktnotation, 123
- Professor, 178
- Programme
 - nicht-terminierend, 237
 - testen, 233
 - verifizieren, 233
- Programmiersprachen
 - imperativ, 2
- Projekte, 111
- Projektionseigenschaften, 251
- Prolog, 222
- prop, 241
- Property, 241
- Pseudozufallszahlen, 212
- Punkt-Operator, 83
- Punkt-vor-Strichrechnung, 38
- Punktnotation, 113
- Pythagoräisches Tripel, 63

- quadratisch, 127
- qualified, 114
- Qualified imports, 114
- Quelle, 204, 207
- Quersumme, 87
- Queue, 157
- queue, 157
- QuickCheck, 240
 - eigene Typen, 242
 - Sortieren, 241
- QuickSort, 146, 175
- Quotientenfolge, 125
- Quotientenkriterium, 125

- RadixSort, 150
- Random, 226
- Random-Access Listen, 185
- Reader, 218, 222
- RealWorld, 226
- Rechenmaschinen, 245

- rechtsassoziativ, 51
- Rechtsassoziativität, 38
- Rechtsfaltung, 169
- Rechtsidentität, 214
- Recordsyntax, 97
- recurrere, 41
- Redex, 250
- Reduktionsstrategie, 250
- Reihenfolge, 27
- Rekursion, 2, 41, 258
 - baumartig, 44
 - endständig, 46
 - kaskadenförmig, 44
 - linear, 43
 - verschachtelt, 45
 - wechselseitig, 46
- Rekursionsabbruch, 154
- Rekursionsanker, 41, 128, 234
- Rekursionsgleichung, 128
 - geschlossene Form, 132
- Rekursionsgrad, 101
- Rekursionskette, 43
- Rekursionsmuster, 99
- Rekursionsschritt, 234
- Rekursionstiefe, 168, 175
- Relation, 101
- return, 243
- Roboter, 193
- Rotation, 161
- Rückgabewert, 210
- runIO, 226

- S, 255
- Saison, 99, 100
- Satellit, 231
- Schlüssel, 172, 177
- Schlüsselwörter, 28
- Schnittstellen, 111
- Schnittstellendefinitionen, 112
- Schranke
 - obere, 123
 - starke obere, 124
 - starke untere, 125
 - Umgang und Regeln, 125
 - untere, 124
- Schrittweite, 58
- Schulden, 162
- Schuldeneinheit, 182
- Schulmathematik, 105
- Schweden, 231
- scope, 32
- Second, 252
- Segmentierung, 198
- Seiteneffekte, 234

- SelectionSort, 142
- Semantik, 214
- semantisch äquivalent, 239
- Senke, 205
- Show, 100, 103, 163, 266
- Show, 183, 191
- Sieb des Eratosthenes, 60, 141
- Signaturen, 26, 29
 - überladen, 30
 - polymorph, 75
- signum, 105
- Skalarprodukt, 67
- SlowSort, 223
- Sortieralgorithmus, 223
 - stabil, 151
- Sortieren
 - BubbleSort, 144
 - BucketSort, 149
 - InsertionSort, 143
 - lexikographisch, 150
 - MergeSort, 148, 178
 - QuickSort, 146, 175
 - RadixSort, 150
 - SelectionSort, 142
 - SlowSort, 223
 - topologisch, 204
 - vergleichsbasiert, 149
- Sortieren durch Auswahl, 142
- Sortieren durch Einfügen, 143
- Sortierwettbewerb, 122
- Spezifikation, 241
- Stack, 137, 141, 152, 163, 199
- Stack Overflow, 43
- Standard-Unicode, 23
- Standardmodul, 112
- Stapel, 142
- Startknoten, 165, 201
- state, 220, 226
- stdGen, 212
- stdin, 228
- stdout, 228
- Steuerzeichen, 66
- Stirling Formel, 126
- Strahlung, 230
- Stream-Abstraktion, 224
- string, 63, 90, 224
- strukturelles Pattern, 52
- Student, 178
- Substitutionen, 263
- Subtraktion, 258
- successor, 234
- Suchbaum, 130, 173, 186
- Summennotation, 123
- Superhelden, 204

- Supermarktkasse, 157, 177
- Symbole, 15
- Synonym. f., 114
- System. IO, 228

- Tabulator, 66
- tail, 51, 253
- Taschenrechner, 7, 9, 22
- Teilausdruck, 248
- Teilbaum, 167, 168, 180
- Teile-und-Herrsche, 132, 133, 146, 148
- Teillisten, 61
- Teilung
 - Rest, 22
- Telefonbuch durchsuchen, 95
- Termauswertung, 38
- Testdatengeneratoren, 242
- Testfälle
 - automatisch generieren, 241
- Texteditor, 4
- TextWrangler, 3
- then, 34
- Theoretische Informatik, 246
- Tiefensuchbaum, 199
- Tiefensuche, 199, 206
- Traversierung, 198
 - ebenenweise, 170
- Traversierungen, 168
- True, 250
- True, 16
- Tupel, 62, 172, 194, 251
- Tupel, 94
- Turing-Fixpunktkombinator, 260
- Türme von Hanoi, 44
- Typüberprüfung, 2
- type, 90
- type variable*, 52
- Typen
 - algebraisch, 91
 - algebraisch rekursiv, 98
 - eigene, 89
 - monomorph, 93
 - polymorph, 90
- Typgleichheit, 27
- Typklassen, 29, 100, 103, 109, 138, 213
 - Abhängigkeiten, 105
 - automatisch instanzieren, 99
 - eigene, 89
- Typkontext, 106
- Typparameter, 91, 94
- Typsicherheit, 2, 29
 - absolut, 29
 - explizit, 91
- Typsynonym, 90
 - monomorph, 91
 - polymorph, 91
- Typvariable, 52

- Umbenennungswörterbuch, 263
- Umgebungsvariablen, 218
- Umgekehrte Polnische Notation, 152
- Umkehrfunktion, 105
- Umkehroperation, 81
- undefined, 97, 262
- unfold, 80
- Ungleichoperator, 19
- unsafePerformIO, 226
- Unterbäume, 180
- Unzip, 82

- Variablen, 247
 - frei, 248, 263
 - gebunden, 248, 263
- Vergleichsoperatoren, 101
- Vergleichsrelation, 150
- Verifikation, 233
- Vier-Euro-Münze, 236
- vollständige Induktion, 234
 - Bäume, 239
 - Listen, 238
 - Strukturen, 237
- Vordergrund, 198
- Vorgängerfunktion, 257
- vorname, 98

- Wachstum
 - asymptotisch gleich, 125
- Wächter, 33
- Wahrheitswerte, 15, 16
 - als Funktionen, 250
- Warteschlange, 157, 163, 200
 - Analyse, 160
- Wartung, 111
- WBuch, 114
- Webanwendungen, 10
- Webseite, 5
- Webserver, 218
- Wegefindung, 193
- Welt, 225, 226
- Wertebereich, 16
- Wertetabelle, 16, 19
- where, 32
- Widerspruchsbeweis, 188
- Wiederverwendbarkeit, 111
- Wildcard, 28, 31
- Windows, 5
- winhugs, 5, 7
- Wochentage, 151

worst-case, 127
Wörterbuch, 171, 178, 195, 219, 220, 263
Wörterbucheintrag, 264
Writer, 217
Wurzel, 165, 178, 187
Wurzelknoten, 168

XOR, 18

Zahlen, 15
Zahlentypen, 21

Zeichenketten, 65, 225
Zeilenumbruch, 66
Zeiteinheit, 122
Zero, 254
Zip, 82
Zufallszahlen, 212
Zugriffszeit, 138
Zurücklaufen, 199
zusammenhängend, 194
Zustand, 16, 221
Zustandsobjekt, 226