

# Inhaltsverzeichnis

<b>Teil I Abenteuer Lambda</b> . . . . .	<b>1</b>
<b>1 Mehr denken, weniger tippen</b> . . . . .	<b>3</b>
1.1 Sag, was du willst! . . . . .	3
1.2 Der alte Weg . . . . .	4
1.3 Abstraktion der Hardware . . . . .	5
1.4 Was ist funktionale Programmierung? . . . . .	6
1.5 Funktionen – Unsere Bausteine . . . . .	6
1.5.1 Was ist eine Funktion? . . . . .	6
1.5.2 Was steckt in einer Funktion? . . . . .	7
1.5.3 Eigenschaften von Funktionen . . . . .	8
1.6 Die referentielle Transparenz . . . . .	11
1.7 Das wirkliche Leben . . . . .	12
<b>2 Ein Hauch von Babylon</b> . . . . .	<b>15</b>
2.1 Lisp – Die Mutter aller funktionalen Sprachen . . . . .	15
2.1.1 Listen – Der Stoff, aus dem Lisp gemacht ist . . . . .	16
2.1.2 Operatoren in Lisp . . . . .	17
2.1.3 Lambda-Ausdrücke . . . . .	18
2.1.4 Mythos Lisp . . . . .	20
2.2 ML – Der Pionier . . . . .	21
2.2.1 Typableitung – Der Compiler erkennt den Typ . . . . .	22
2.2.2 Muster und Ausdrücke . . . . .	22
2.2.3 Generische Funktionen und Datentypen . . . . .	24
2.3 Haskell – Funktionale Programmierung in Reinform . . . . .	25
2.3.1 Haskell – Ein fauler Hund . . . . .	26
2.4 Wann ist eine Sprache funktional? . . . . .	28
2.5 Und die Performance? . . . . .	28

2.6	Welche Sprache darf's denn sein? . . . . .	29
2.6.1	Java . . . . .	29
2.6.2	Scala . . . . .	30
2.7	Aufgaben . . . . .	31
<b>Teil II Die funktionale Seite von Java . . . . .</b>		<b>33</b>
<b>3</b>	<b>Rekursion . . . . .</b>	<b>35</b>
3.1	Eine Schleife – was ist das eigentlich? . . . . .	35
3.2	Die Rekursion – das unbekannte Wesen . . . . .	36
3.3	Ist Rekursion praxistauglich? . . . . .	37
3.4	Wie werden rekursive Funktionen verarbeitet? . . . . .	38
3.5	Endrekursion, die schnelle Rekursion . . . . .	40
3.6	Eine einfache Formel und ihre schwerwiegenden Folgen . . . . .	43
3.7	Aufgaben . . . . .	46
<b>4</b>	<b>Alles bleibt, wie es ist . . . . .</b>	<b>49</b>
4.1	Konsistenz . . . . .	49
4.2	Nichts ist beständiger als der Wandel . . . . .	50
4.3	Versteckte Daten sind gute Daten . . . . .	51
4.4	Invarianten – darauf ist Verlass . . . . .	52
4.5	Ein Beispiel . . . . .	54
4.6	Die Methode <code>equals</code> . . . . .	55
4.7	Sichere Ergebnisse mit defensiven Kopien . . . . .	55
4.8	Konkurrierende Zugriffe . . . . .	57
4.9	Geänderte Hash-Codes . . . . .	58
4.10	Unveränderbare Klassen . . . . .	59
4.11	Performance . . . . .	60
4.12	Unveränderbare Klassen in der Java-API . . . . .	61
4.13	Aufgaben . . . . .	62
<b>5</b>	<b>Funktionen höherer Ordnung . . . . .</b>	<b>63</b>
5.1	Arrays sortieren leicht gemacht . . . . .	63
5.2	Eine flexiblere Lösung . . . . .	65
5.3	Funktionen als Ergebnisse von Funktionen . . . . .	68
5.4	Aufgaben . . . . .	71

---

<b>6</b>	<b>Unveränderbare Listen</b>	<b>73</b>
6.1	Arrays raus!	73
6.2	Verkettete Listen – Vom Nobody zum Star	74
6.3	Nützliche Methoden für die Arbeit mit Listen	76
6.4	Das schwer erreichbare Listenende	77
6.5	Die Faltung – eine universelle Funktion	78
6.6	Eine Leihgabe aus der imperativen Programmierung	81
6.7	Aufgaben	83
<b>7</b>	<b>Anfragen an Listen</b>	<b>85</b>
7.1	Auswahlen aus Listen	86
7.2	Listenelemente abbilden	87
7.3	Quicksort	88
7.4	Primzahlen	90
7.5	Verknüpfungen von Listen	91
7.6	Aufgaben	93
<b>Teil III</b>	<b>Scala</b>	<b>95</b>
<b>8</b>	<b>Die Scala-Entwicklungsumgebung</b>	<b>97</b>
8.1	Ohne Java geht nichts	97
8.2	Installation und erste Schritte	98
8.3	Scala-Skripts	98
8.4	Eigene Typen mit Scala-Klassen	99
8.5	Noch ein Compiler	100
8.6	Der Scala-Bazar	101
8.7	Aufgaben	102
<b>9</b>	<b>Ausdrücke in Scala</b>	<b>105</b>
9.1	Erste Eindrücke mit einfachen Ausdrücken	105
9.2	Konstante	106
9.3	Variable	108
9.4	Alle kennen Predef	108
9.5	Kontrollstrukturen	109
9.6	Importe	110
9.7	Aufgaben	110

---

<b>10 Scala-Typsystem</b> . . . . .	<b>113</b>
10.1 In Scala gibt es keine primitiven Typen . . . . .	113
10.2 Alles ist ein Objekt . . . . .	114
10.3 Objekte vergleichen . . . . .	115
10.4 Werte- und Referenztypen . . . . .	116
10.5 Literale und Typumwandlungen . . . . .	116
10.6 Der Typ <code>Unit</code> . . . . .	117
10.7 Der Typ <code>Null</code> . . . . .	117
10.8 Der Typ <code>Nothing</code> . . . . .	118
10.9 Aufgaben . . . . .	119
<b>11 Methoden in Scala</b> . . . . .	<b>121</b>
11.1 Jede Methode hat einen Typ . . . . .	121
11.2 Generische Methoden . . . . .	122
11.3 Konstante als Grenzfall von Methoden . . . . .	122
11.4 Was steht in einer Methode? . . . . .	123
11.5 Methoden ohne Rückgabewert . . . . .	125
11.6 Methoden in Methoden . . . . .	125
11.7 Methoden für beliebig viele Argumente . . . . .	126
11.8 Endrekursion . . . . .	127
11.9 Aufgaben . . . . .	129
<b>12 Funktionen in Scala</b> . . . . .	<b>131</b>
12.1 Die Definition einer Funktion . . . . .	131
12.2 Funktionen sind auch Objekte . . . . .	133
12.3 Die partielle Anwendung einer Funktion . . . . .	134
12.4 Eine scharfe Sache: Das Curry-Prinzip . . . . .	136
12.5 Funktionen höherer Ordnung . . . . .	138
12.6 Aufgaben . . . . .	140
<b>13 Tupel</b> . . . . .	<b>143</b>
13.1 Eigenschaften von Tupeln . . . . .	144
13.2 Die Tupel-Klassen . . . . .	145
13.3 Mustererkennung für Tupel . . . . .	146
13.4 Aufgaben . . . . .	147

---

<b>14 Klassen und Vererbung in Scala</b> . . . . .	<b>149</b>
14.1 Klassendefinitionen in Scala . . . . .	149
14.1.1 Ein alter Bekannter zum Einstieg . . . . .	149
14.1.2 Konstruktoren . . . . .	150
14.1.3 Attribute und parameterfreie Methoden . . . . .	152
14.1.4 Mehrere Fliegen mit einer Klappe schlagen . . . . .	153
14.1.5 Was bedeutet „rechtsassoziativ“? . . . . .	154
14.2 Vererbung . . . . .	154
14.2.1 Methoden überschreiben . . . . .	154
14.2.2 Konstruktorverkettung . . . . .	156
14.2.3 Polymorphie . . . . .	157
14.2.4 Abstrakte Klassen . . . . .	158
14.3 Aufgaben . . . . .	159
<b>15 Singletons: Objekte können einsam sein</b> . . . . .	<b>161</b>
15.1 Es kann nur einen geben . . . . .	162
15.2 Statische Mitglieder waren gestern . . . . .	163
15.3 Begleiter . . . . .	163
15.4 Singletons zur Objektverwaltung . . . . .	165
15.5 Singletons importieren . . . . .	165
15.6 Aufgaben . . . . .	166
<b>16 Mustererkennung</b> . . . . .	<b>167</b>
16.1 Muster in Java . . . . .	168
16.2 Einfache Muster in Scala . . . . .	169
16.3 Muster für Tupel . . . . .	170
16.4 Welche Muster gibt es? . . . . .	171
16.5 Muster für Typen . . . . .	172
16.6 Muster in Funktionen . . . . .	174
16.7 Partielle Funktionen . . . . .	174
16.8 Exceptions und Mustererkennung . . . . .	175
16.9 Aufgaben . . . . .	176
<b>17 Extraktoren und Case-Typen</b> . . . . .	<b>177</b>
17.1 Besser als null: Der Typ Option . . . . .	178
17.2 Extraktormuster . . . . .	178
17.3 Optionale Attribute . . . . .	179
17.4 Extraktoren sind universelle Inspektoren . . . . .	180

---

17.5	Lesbarer Code mit <code>apply</code> . . . . .	182
17.6	Variable Extraktoren . . . . .	183
17.7	Alles frei Haus mit Case-Typen . . . . .	183
17.8	Aufgaben . . . . .	187
<b>18</b>	<b>Listen . . . . .</b>	<b>189</b>
18.1	Die leere Liste . . . . .	189
18.2	Listen erzeugen . . . . .	190
18.3	Einfache Methoden für Listen . . . . .	190
18.4	Listenmuster . . . . .	191
18.5	Weitere einfache Methoden . . . . .	192
18.6	Mengenoperationen . . . . .	194
18.7	Das Begleitobjekt . . . . .	195
18.8	Methoden höherer Ordnung für Listen . . . . .	196
	18.8.1 Beispiel: Quicksort . . . . .	197
	18.8.2 Die Faltung . . . . .	198
18.9	Das Springerproblem . . . . .	200
18.10	Aufgaben . . . . .	202
<b>19</b>	<b>Scala kann auch faul sein . . . . .</b>	<b>203</b>
19.1	Die Initialisierung kann warten . . . . .	203
19.2	Faule Parameter mit Call-By-Name . . . . .	205
19.3	Streams: Daten bei Bedarf . . . . .	206
19.4	Unendliche Streams . . . . .	208
19.5	Aufgaben . . . . .	209
<b>20</b>	<b>Es müssen nicht immer Listen sein . . . . .</b>	<b>211</b>
20.1	Mengen . . . . .	211
20.2	Der Typ <code>Set</code> . . . . .	213
20.3	Der Typ <code>Map</code> . . . . .	214
20.4	Collections in anderen Geschmacksrichtungen . . . . .	218
20.5	Aufgaben . . . . .	219
<b>21</b>	<b>Fast wie zu Hause: <code>for</code>-Ausdrücke . . . . .</b>	<b>221</b>
21.1	Eine nicht ganz so funktionale Methode höherer Ordnung . . . . .	221
21.2	Komplexere <code>for</code> -Ausdrücke . . . . .	222
21.3	<code>for</code> -Ausdrücke mit <code>filter</code> und <code>map</code> . . . . .	222
21.4	Mehr Übersicht mit <code>flatMap</code> . . . . .	223

---

21.5	<i>for</i> -Ausdrücke sind keine Schleifen . . . . .	224
21.6	<i>for</i> -Ausdrücke für eigene Typen . . . . .	225
21.7	Aufgaben . . . . .	226
<b>Teil IV Scala kann mehr . . . . .</b>		<b>227</b>
<b>22</b>	<b>Veränderbare Daten . . . . .</b>	<b>229</b>
22.1	Variable . . . . .	229
22.2	Veränderbare Attribute . . . . .	230
22.3	Die Rückkehr der Arrays . . . . .	232
22.4	Aufgaben . . . . .	233
<b>23</b>	<b>Traits . . . . .</b>	<b>235</b>
23.1	Traits und Java-Interfaces . . . . .	236
23.2	Konkrete Methoden . . . . .	236
23.3	Mehrfachvererbung . . . . .	238
23.4	Aufgaben . . . . .	239
<b>24</b>	<b>Varianz . . . . .</b>	<b>241</b>
24.1	Kovarianz von Java-Arrays . . . . .	241
24.2	Kovarianz von generischen Java-Typen . . . . .	242
24.3	Mehr Kovarianz in Java . . . . .	243
24.4	Kontravarianz in Java . . . . .	244
24.5	Varianz in Scala . . . . .	244
<b>25</b>	<b>Pakete und Sichtbarkeit . . . . .</b>	<b>249</b>
25.1	Pakete in Scala . . . . .	249
25.2	Sichtbarkeit in Scala . . . . .	250
25.3	Privater als privat . . . . .	252
<b>26</b>	<b>Typumwandlung . . . . .</b>	<b>255</b>
26.1	Implizite Methoden für implizite Casts . . . . .	255
26.2	Wozu noch explizite Casts? . . . . .	257
26.3	Angereicherte Typen . . . . .	257
26.4	Pimp Your Library! . . . . .	258
26.5	Sprachfeatures mit impliziten Casts umsetzen . . . . .	259
26.6	Aufgaben . . . . .	259

---

<b>27 Parallele Programmierung mit Aktoren . . . . .</b>	<b>261</b>
27.1 Viele Köche verderben den Brei . . . . .	262
27.2 Parallele Programmierung mal anders . . . . .	262
27.3 Erste Versuche mit Aktoren . . . . .	264
27.4 Der Typ Actor . . . . .	265
27.5 Die Actor-Fabrik . . . . .	266
27.6 Einfache Standardprobleme und ihre Lösung . . . . .	267
27.7 Aktoren können antworten . . . . .	268
27.8 Producer-Consumer-Probleme . . . . .	269
27.9 Blockierende Stacks . . . . .	271
27.10 Deadlocks – Nichts geht mehr . . . . .	272
27.11 Aufgaben . . . . .	276
<b>Literaturverzeichnis . . . . .</b>	<b>277</b>
<b>Stichwortverzeichnis . . . . .</b>	<b>279</b>