

# Contents

1	Introduction (or, Why I Wrote This Book)	1
2	Who This Book is For	5
3	Goals, Part 1: “Soft” Goals of This Book	9
4	Goals, Part 2: Concrete Goals	13
5	Goals, Part 3: A Disclaimer	15
6	Question Everything	17
7	Rules for Programming in this Book	23
8	One Rule for Reading this Book	27
9	What is “Functional Programming”?	29
10	What is This Lambda You Speak Of?	39
11	The Benefits of Functional Programming	43
12	Disadvantages of Functional Programming	59
13	The “Great FP Terminology Barrier”	75
14	Pure Functions	81
15	Benefits of Pure Functions	89
16	Pure Functions and I/O	97
17	Pure Function Signatures Tell All	101
18	Functional Programming as Algebra	107

19	A Note About Expression-Oriented Programming	119
20	Functional Programming is Like Unix Pipelines	123
21	Functions Are Variables, Too	137
22	Using Methods As If They Were Functions	151
23	How to Write Functions That Take Functions as Input Parameters	161
24	How to Write a 'map' Function	179
25	How to Use By-Name Parameters	185
26	Functions Can Have Multiple Parameter Groups	197
27	Partially-Applied Functions (and Currying)	211
28	Recursion: Introduction	223
29	Recursion: Motivation	225
30	Recursion: Let's Look at Lists	229
31	Recursion: How to Write a 'sum' Function	235
32	Recursion: How Recursive Function Calls Work	241
33	Visualizing the Recursive sum Function	245
34	Recursion: A Conversation Between Two Developers	253
35	Recursion: Thinking Recursively	255
36	JVM Stacks and Stack Frames	261
37	A Visual Look at Stacks and Frames	269
38	Tail-Recursive Algorithms	275
39	A First Look at "State"	283
40	A Functional Game (With a Little Bit of State)	287

## CONTENTS

41	A Quick Review of Case Classes	301
42	Update as You Copy, Don't Mutate	305
43	A Quick Review of for-Expressions	315
44	How to Write a Class That Can Be Used in a for-Expression	323
45	Creating a Sequence Class to be Used in a for Comprehension	325
46	Making Sequence Work In a Simple for Loop	331
47	How To Make Sequence Work as a Single Generator in a for Expression	335
48	Enabling Filtering in a for Expression	339
49	How to Enable the Use of Multiple Generators in a for Expression	345
50	A Summary of the for Expression Lessons	355
51	Pure Functions Tell No Lies	357
52	Functional Error Handling (Option, Try, Or, and Either)	363
53	Embrace The Idioms!	373
54	What to Think When You See That Opening Curly Brace	377
55	A Quick Review of How flatMap Works	391
56	Option Naturally Leads to flatMap	397
57	flatMap Naturally Leads to for	401
58	for Expressions are Better Than getOrElse	403
59	Recap: Option $\rightarrow$ flatMap $\rightarrow$ for	407
60	A Note About Things That Can Be Mapped Over	415
61	Starting to Glue Functions Together	417
62	The "Bind" Concept	421

63	Getting Close to Using <code>bind</code> in <code>for</code> Expressions	427
64	Using a Wrapper Class in a <code>for</code> Expression	429
65	Making Wrapper More Generic	437
66	Changing “new Wrapper” to “Wrapper”	439
67	A Quick Note About Case Classes and Companion Objects	445
68	Using <code>bind</code> in a <code>for</code> Expression	447
69	How <code>Debuggable</code> , <code>f</code> , <code>g</code> , and <code>h</code> Work	459
70	A Generic Version of <code>Debuggable</code>	471
71	One Last <code>Debuggable</code> : Using <code>List</code> Instead of <code>String</code>	475
72	Key Points About Monads	481
73	Signpost: Where We’re Going Next	483
74	Introduction: The <code>IO</code> Monad	485
75	How to Use an <code>IO</code> Monad	487
76	Assigning a <code>for</code> Expression to a Function	491
77	The <code>IO</code> Monad and a <code>for</code> Expression That Uses Recursion	493
78	Diving Deeper Into the <code>IO</code> Monad	495
79	I’ll Come Back to the <code>IO</code> Monad	501
80	Functional Composition	503
81	An Introduction to Handling State	507
82	Handling State Manually	509
83	Getting State Working in a <code>for</code> Expression	515
84	Handling My Golfing State with a State Monad	517

## CONTENTS

85	The State Monad Source Code	523
86	Signpost: Getting IO and State Working Together	527
87	Trying to Write a for Expression with IO and State	529
88	Seeing the Problem: Trying to Use State and IO Together	531
89	Solving the Problem with Monad Transformers	533
90	Beginning the Process of Understanding StateT	535
91	Getting Started: We're Going to Need a Monad Trait	539
92	Now We Can Create StateT	543
93	Using StateT in a for Expression	545
94	Trying to Combine IO and StateT in a for Expression	551
95	Fixing the IO Functions with Monadic Lifting	555
96	A First IO/StateT for-Expression	559
97	The Final IO/StateT for Expression	563
98	Summary of the StateT Lessons	567
99	Signpost: Modeling the world with Scala/FP	569
100	What is a Domain Model?	571
101	A Review of OOP Data Modeling	573
102	Modeling the "Data" Portion of the Pizza POS System with Scala/FP	581
103	First Attempts to Organize Pure Functions	585
104	Implementing FP Behavior with Modules	591
105	Implementing the Pizza POS System Using a Modular Approach	599
106	The "Functional Objects" Approach	617

107	Demonstrating the “Functional Objects” Approach	621
108	Summary of the Domain Modeling Approaches	627
109	The Problem with the IO Monad	629
110	Lenses, to Simplify “Update as You Copy”	635
111	Signpost: Concurrency	639
112	Concurrency and Mutability Don’t Mix	641
113	Scala Concurrency Tools	649
114	Akka Actors	653
115	Akka Actor Examples	659
116	Scala Futures	669
117	A Second Futures Example	677
118	Key Points About Scala Futures	689
119	A Few Notes About Real World Functional Programming	691
120	Signpost: Wrapping Things Up	699
121	The Learning Path	701
122	Final Summary	703
123	Where To Go From Here	709