

Functional Programming Using F#

This introduction to the principles of functional programming using F# shows how to apply theoretical concepts to produce succinct and elegant programs. The book shows how mainstream computer science problems can be solved using functional programming. It introduces a model-based approach exploiting the rich type system of F#. It also demonstrates the role of functional programming in a wide spectrum of applications including databases and systems that engage in a dialogue with a user. Coverage also includes advanced features in the .NET library, the imperative features of F# and topics such as sequences, computation expressions and asynchronous computations.

With a broad spectrum of examples and exercises, the book is intended for courses in functional programming as well as for self-study. Enhancing its use as a text is a website with downloadable programs, lecture slides, mini-projects and links to further F# sources.

Michael R. Hansen is an Associate Professor in Computer Science at the Technical University of Denmark. He is the author of *Introduction to Programming Using SML* (with Hans Rischel) and *Duration Calculus: A Formal Approach to Real-Time Systems* (with Zhou Chaochen).

Hans Rischel is a former Associate Professor in Computer Science at the Technical University of Denmark. He is the author of *Introduction to Programming Using SML* (with Michael R. Hansen).

Cambridge University Press
978-1-107-01902-7 - Functional Programming Using F#
Michael R. Hansen and Hans Rischel
Frontmatter
[More information](#)

Cambridge University Press
978-1-107-01902-7 - Functional Programming Using F#
Michael R. Hansen and Hans Rischel
Frontmatter
[More information](#)

Functional Programming Using F#

MICHAEL R. HANSEN

Technical University of Denmark, Lyngby

HANS RISCHEL

Technical University of Denmark, Lyngby



CAMBRIDGE
UNIVERSITY PRESS

Cambridge University Press
978-1-107-01902-7 - Functional Programming Using F#
Michael R. Hansen and Hans Rischel
Frontmatter
[More information](#)

CAMBRIDGE
UNIVERSITY PRESS

32 Avenue of the Americas, New York, NY 10013-2473, USA

Cambridge University Press is part of the University of Cambridge.

It furthers the University’s mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107684065

© Michael R. Hansen and Hans Rischel 2013

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2013
Reprinted 2015

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data

Hansen, Michael R., author.
Functional programming Using F# / Michael R. Hansen, Technical University of Denmark, Lyngby, Hans Rischel, Technical University of Denmark, Lyngby.
pages cm
Includes bibliographical references and index.
ISBN 978-1-107-01902-7 (hardback) – ISBN 978-1-107-68406-5 (paperback)
1. Functional programming (Computer science) 2. F# (Computer program language)
I. Rischel, Hans, author. II. Title.
QA76.62.H37 2013
005.1’14—dc23 2012040414

ISBN 978-1-107-01902-7 Hardback
ISBN 978-1-107-68406-5 Paperback

Additional resources for this publication at <http://www.cambridge.org/9781107019027>

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet websites referred to in this publication and does not guarantee that any content on such websites is, or will remain, accurate or appropriate.

Contents

<i>Preface</i>	<i>page ix</i>
1 Getting started	1
1.1 Values, types, identifiers and declarations	1
1.2 Simple function declarations	2
1.3 Anonymous functions. Function expressions	4
1.4 Recursion	6
1.5 Pairs	11
1.6 Types and type checking	13
1.7 Bindings and environments	14
1.8 Euclid’s algorithm	15
1.9 Evaluations with environments	17
1.10 Free-standing programs	19
Summary	19
Exercises	20
2 Values, operators, expressions and functions	21
2.1 Numbers. Truth values. The <code>unit</code> type	21
2.2 Operator precedence and association	23
2.3 Characters and strings	24
2.4 If-then-else expressions	28
2.5 Overloaded functions and operators	29
2.6 Type inference	31
2.7 Functions are first-class citizens	31
2.8 Closures	34
2.9 Declaring prefix and infix operators	35
2.10 Equality and ordering	36
2.11 Function application operators <code> ></code> and <code>< </code>	38
2.12 Summary of the basic types	38
Summary	39
Exercises	39
3 Tuples, records and tagged values	43
3.1 Tuples	43
3.2 Polymorphism	48
3.3 Example: Geometric vectors	48
3.4 Records	50

vi	<i>Contents</i>	
3.5	Example: Quadratic equations	52
3.6	Locally declared identifiers	54
3.7	Example: Rational numbers. Invariants	56
3.8	Tagged values. Constructors	58
3.9	Enumeration types	62
3.10	Exceptions	63
3.11	Partial functions. The option type	64
	Summary	65
	Exercises	66
4	Lists	67
4.1	The concept of a list	67
4.2	Construction and decomposition of lists	71
4.3	Typical recursions over lists	74
4.4	Polymorphism	78
4.5	The value restrictions on polymorphic expressions	81
4.6	Examples. A model-based approach	82
	Summary	88
	Exercises	89
5	Collections: Lists, maps and sets	93
5.1	Lists	93
5.2	Finite sets	104
5.3	Maps	113
	Summary	119
	Exercises	119
6	Finite trees	121
6.1	Chinese boxes	121
6.2	Symbolic differentiation	127
6.3	Binary trees. Parameterized types	131
6.4	Traversal of binary trees. Search trees	133
6.5	Expression trees	137
6.6	Trees with a variable number of sub-trees. Mutual recursion	138
6.7	Electrical circuits	142
	Summary	144
	Exercises	145
7	Modules	149
7.1	Abstractions	149
7.2	Signature and implementation	150
7.3	Type augmentation. Operators in modules	153
7.4	Type extension	155
7.5	Classes and objects	156
7.6	Parameterized modules. Type variables in signatures	157
7.7	Customizing equality, hashing and the <code>string</code> function	159
7.8	Customizing ordering and indexing	161
7.9	Example: Piecewise linear plane curves	162

	<i>Contents</i>	vii
	Summary	170
	Exercises	170
8	Imperative features	175
8.1	Locations	175
8.2	Operators on locations	176
8.3	Default values	179
8.4	Sequential composition	179
8.5	Mutable record fields	180
8.6	References	182
8.7	While loops	183
8.8	Imperative functions on lists and other collections	184
8.9	Imperative tree traversal	185
8.10	Arrays	186
8.11	Imperative set and map	188
8.12	Functions on collections. Enumerator functions	190
8.13	Imperative queue	194
8.14	Restrictions on polymorphic expressions	195
	Summary	195
	Exercises	196
9	Efficiency	197
9.1	Resource measures	197
9.2	Memory management	198
9.3	Two problems	204
9.4	Solutions using accumulating parameters	206
9.5	Iterative function declarations	209
9.6	Tail recursion obtained using continuations	212
	Summary	216
	Exercises	216
10	Text processing programs	219
10.1	Keyword index example: Problem statement	219
10.2	Capturing data using regular expressions	221
10.3	Text I/O	229
10.4	File handling. Save and restore values in files	230
10.5	Reserving, using and disposing resources	232
10.6	Culture-dependent information. String orderings	232
10.7	Conversion to textual form. Date and time	235
10.8	Keyword index example: The <code>IndexGen</code> program	238
10.9	Keyword index example: Analysis of a web-source	242
10.10	Keyword index example: Putting it all together	245
	Summary	248
	Exercises	249
11	Sequences	251
11.1	The sequence concept in F#	251
11.2	Some operations on sequences	254

viii	<i>Contents</i>	
11.3	Delays, recursion and side-effects	256
11.4	Example: Sieve of Eratosthenes	258
11.5	Limits of sequences: Newton-Raphson approximations	260
11.6	Sequence expressions	262
11.7	Specializations of sequences	266
11.8	Type providers and databases	267
	Summary	277
	Exercises	277
12	Computation expressions	279
12.1	The agenda when defining your own computations	280
12.2	Introducing computation expressions using sequence expressions	281
12.3	The basic functions: <code>For</code> and <code>Yield</code>	282
12.4	The technical setting when defining your own computations	284
12.5	Example: Expression evaluation with error handling	285
12.6	The basic functions: <code>Bind</code> , <code>Return</code> , <code>ReturnFrom</code> and <code>Zero</code>	286
12.7	Controlling the computations: <code>Delay</code> and <code>Start</code>	288
12.8	The basic function: <code>Delay</code>	290
12.9	The fundamental properties of <code>For</code> and <code>Yield</code> , <code>Bind</code> and <code>Return</code>	291
12.10	Monadic parsers	293
	Summary	309
	Exercises	309
13	Asynchronous and parallel computations	311
13.1	Multi-core processors, cache memories and main memory	311
13.2	Processes, threads and tasks	312
13.3	Challenges and pitfalls in concurrency	314
13.4	Asynchronous computations	316
13.5	Reactive programs	321
13.6	Parallel computations	328
	Summary	335
	Exercises	336
<i>Appendix A</i>	Programs from the keyword example	339
A.1	Web source files	339
A.2	The <code>IndexGen</code> program	342
A.3	The <code>NextLevelRefs</code> program	344
<i>Appendix B</i>	The <code>TextProcessing</code> library	346
<i>Appendix C</i>	The dialogue program from Chapter 13	350
<i>References</i>		353
<i>Index</i>		355

Preface

The purpose of this book is to introduce a wide range of readers – from the professional programmer to the computer science student – to the rich world of functional programming using the F# programming language. The book is intended as the textbook in a course on functional programming and aims at showing the role of functional programming in a wide spectrum of applications ranging from computer science examples over database examples to systems that engage in a dialogue with a user.

Why functional programming using F#?

Functional programming languages have existed in academia for more than a quarter of a century, starting with the untyped Lisp language, followed by strongly typed languages like Haskell and Standard ML.

The penetration of functional languages to the software industry has, nevertheless, been surprisingly slow. The reason is probably lack of support of functional languages by commercial software development platforms, and software development managers are reluctant to base software development on languages living in a non-commercial environment.

This state of affairs has been changed completely by the appearance of F#, an open-source, full-blown functional language integrated in the Visual Studio development platform and with access to all features in the .NET program library. The language is also supported on Linux and MAC systems using the Mono platform.

The background

The material in this book has been developed in connection with courses taught at the Technical University of Denmark, originating from the textbook *Introduction to Programming Using SML* by Hansen and Rischel (Addison-Wesley, 1999).

It has been an exciting experience for us to learn the many elegant and useful features of the F# language, and this excitement is hopefully transferred to the reader of this book.

The chapters

- Chapter 1: The basic concepts of F#, including values, types and recursive functions, are introduced in a manner that allows readers to solve interesting problems from the start.
- Chapter 2: A thorough introduction to the basic types in F# is given, together with a gentle introduction to the notion of higher-order functions.
- Chapter 3: The simplest composite types of F#, tuples and records, are introduced. They allow several values to be grouped together into one component. Furthermore, tagged values are introduced.

- Chapter 4: A list is a finite sequence of values with the same type. Standard recursions on lists are studied and examples illustrating a model-based approach to functional programming are given.
- Chapter 5: The concepts of sets and maps are introduced and the powerful F# collection libraries for lists, sets and maps are studied and applied in connection with a model-based approach.
- Chapter 6: The concept of finite tree is introduced and illustrated through a broad selection of examples.
- Chapter 7: It is shown how users can make their own libraries by means of modules consisting of signature and implementation files. Furthermore, object-oriented features of F# are mentioned.
- Chapter 8: Imperative features of F# are introduced, including the array part of the collection library and the imperative sets and maps from the .NET framework.
- Chapter 9: The memory management concepts, stack, heap and garbage collection, are described. Tail-recursive functions are introduced and two techniques for deriving such functions are presented: one using accumulating parameters, the other continuations. Their efficiency advantages are illustrated.
- Chapter 10: A variety of facilities for processing text are introduced, including regular expressions, file operations, web-based operations and culture-dependent string ordering. The facilities are illustrated using a real-world example.
- Chapter 11: A sequence is a, possibly infinite, collection of elements that are computed on-demand only. Sequence functions are expressed using library functions or sequence expressions that provide a step-by-step method for generating elements. Database tables are viewed as sequences (using a type provider) and operations on databases are expressed using query expressions.
- Chapter 12: The notion of computation expression, which is based on the theory of monads, is studied and used to hide low-level details of a computation from its definition. Monadic parsing is used as a major example to illustrate the techniques.
- Chapter 13: This last chapter describes how to construct asynchronous reactive programs, spending most of their time awaiting a request or a response from an external agent, and parallel programs, exploiting the multi-core processor of the computer.

The first six chapters cover a standard curriculum in functional programming, while the other chapters cover more advanced topics.

Further material

The book contains a large number of exercises, and further material is available at the book's homepage. A link to this homepage is found at:

<http://www.cambridge.org/9781107019027>

This material includes a complete set of slides for a course in functional programming plus a collection of problems and descriptions of topics to be used in student projects.

Preface

xi

Acknowledgments

Special thanks go to Peter Sestoft, Don Syme and Anh-Dung Phan. The idea to make a textbook on functional programming on the basis of F# originates from Peter, who patiently commented on the manuscript during its production and helped with advice and suggestions. From the very start of this project we had the support of Don. This is strongly appreciated and so is the help, clarifications and constructive comments that we received throughout this project. Phan helped with many comments, suggestions and insights about the platform. We are grateful for this help, for many discussions and for careful comments on all the chapters.

Furthermore, we are grateful to Nils Andersen, Mary E. Böker, Diego Colombo and Niels Hallenberg for reading and commenting on the complete manuscript.

Earlier versions of this manuscript have been used in connection with courses at the Technical University of Denmark and the IT-University of Copenhagen. The comments we received from the students in these courses are greatly appreciated.

Lyngby, July 31, 2012

Michael R. Hansen and Hans Rischel