

# Stylish F#

Crafting Elegant Functional Code  
for .NET and .NET Core

**Kit Eason**

Apress®

## ***Stylish F#: Crafting Elegant Functional Code for .NET and .NET Core***

Kit Eason  
Farnham, Surrey, UK

ISBN-13 (pbk): 978-1-4842-3999-5  
<https://doi.org/10.1007/978-1-4842-4000-7>

ISBN-13 (electronic): 978-1-4842-4000-7

Library of Congress Control Number: 2018963299

Copyright © 2018 by Kit Eason

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr

Acquisitions Editor: Joan Murray

Development Editor: Laura Berendson

Coordinating Editor: Jill Balzano

Cover image designed by Kit Eason. Contains OS data © Crown copyright and database right 2018

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [rights@apress.com](mailto:rights@apress.com), or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at [www.apress.com/9781484239995](http://www.apress.com/9781484239995). For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*To Val, Matt, Meg, Kate, Noah, and Darwin:  
my own persistent collection.*

# Table of Contents

<b>About the Author</b> .....	<b>xv</b>
<b>About the Technical Reviewer</b> .....	<b>xvii</b>
<b>Acknowledgments</b> .....	<b>xix</b>
<b>Introduction</b> .....	<b>xxi</b>
<b>Chapter 1: The Sense of Style</b> .....	<b>1</b>
Why a Style Guide? .....	1
Understanding Beats Obedience.....	2
Good Guidance from Bad Code .....	2
What About Testability?.....	8
Complexity Explosions .....	8
Summary.....	9
<b>Chapter 2: Designing Functions Using Types</b> .....	<b>11</b>
Miles and Yards (No, Really!).....	11
Converting Miles and Yards to Decimal Miles.....	12
How to Design a Function .....	13
Sketch the Signature of the Function .....	13
Naïvely Code the Body of the Function.....	14
Review the Signature for Type Safety.....	14
Review and Refine.....	18
A Final Polish .....	22
Recommendations .....	24

TABLE OF CONTENTS

- Summary..... 24
- Exercises..... 25
- Exercise Solutions..... 26
- Chapter 3: Missing Data ..... 29**
  - A Brief History of Null..... 29
  - Option Types versus Null..... 32
  - Consuming Option Types..... 34
    - Pattern Matching on Option Types..... 35
    - The Option Module ..... 36
  - Option Type No-Nos..... 43
  - Designing Out Missing Data..... 44
  - Interoperating with the Nullable World ..... 47
    - Leaking In of Null Values ..... 47
    - Defining a SafeString Type ..... 49
    - Using Option.ofObj..... 50
    - Using Option.ofNullable ..... 50
    - Leaking Option Types and DUs Out..... 51
    - Using Option.toObj..... 51
    - Using Option.toNullable ..... 53
    - The Future of Null ..... 54
  - The ValueOption Type ..... 54
  - Recommendations ..... 55
  - Summary..... 56
  - Exercises..... 56
  - Exercise Solutions..... 58
- Chapter 4: Working Effectively with Collection Functions..... 61**
  - Anatomy of a Collection Function ..... 61
  - Picking the Right Collection Function ..... 64
  - Detailed Collection Function Tables ..... 66

Practicing with Collection Functions.....	72
Exercise Setup.....	72
Single Collection Function Exercises.....	73
Multiple Collection Function Exercises.....	77
Partial Functions.....	80
Coding Around Partial Functions.....	82
Using the “try” Idiom for Partial Functions.....	84
Consuming Values from try... Functions.....	86
Try... Function Exercises.....	86
Functions for Other Kinds of Collections.....	87
When the Collection Function Is Missing.....	88
Common Mistakes.....	89
Recommendations.....	93
Summary.....	94
Exercise Solutions.....	94
<b>Chapter 5: Immutability and Mutation.....</b>	<b>99</b>
These Folks Are Crazy!.....	99
Classic Mutable Style.....	99
Immutability Basics.....	101
Common Mutable Patterns.....	103
Linear Search.....	103
Guarded Linear Search.....	105
Process All Items.....	106
Repeat Until.....	108
Find Extreme Value.....	111
Summarize a Collection.....	112
Recommendations.....	114
Summary.....	115
Exercises.....	115
Exercise Solutions.....	117

**Chapter 6: Pattern Matching ..... 119**

- Weaving Software with Patterns..... 119
- Pattern Matching Basics ..... 119
- When Guards..... 122
- Pattern Matching on Arrays and Lists ..... 122
- Pattern Matching on Tuples ..... 125
- Pattern Matching on Records..... 126
- Pattern Matching on Discriminated Unions..... 128
- Pattern Matching on DUs in Function Parameters..... 130
- Pattern Matching in Let Bindings..... 132
- Pattern Matching in Loops and Lambdas..... 133
- Pattern Matching and Enums..... 135
- Active Patterns..... 137
  - Single Case Active Patterns..... 137
  - Multi-Case Active Patterns ..... 139
  - Partial Active Patterns ..... 140
  - Parameterized Active Patterns ..... 141
- Pattern Matching with ‘&’ ..... 143
- Pattern Matching on Types..... 144
- Pattern Matching on Null ..... 146
- Recommendations ..... 147
- Summary..... 149
- Exercises..... 150
- Exercise Solutions..... 152

**Chapter 7: Record Types ..... 157**

- Winning with Records..... 157
- Record Type Basics ..... 157
- Record Types and Immutability ..... 158
- Default Constructors, Setters, and Getters..... 160

Records versus Classes .....	161
Structural Equality by Default.....	162
Records as Structs .....	165
Mapping from Instantiation Values to Members.....	167
Records Everywhere? .....	168
Pushing Records to the Limit.....	169
Generic Records .....	169
Recursive Records.....	171
Records with Methods.....	172
Records with Methods – A Good Idea? .....	175
Record Layout .....	175
Recommendations .....	177
Summary.....	178
Exercises.....	178
Exercise Solutions.....	180
<b>Chapter 8: Classes .....</b>	<b>183</b>
The Power of Classes.....	183
Asymmetric Representation.....	183
Constructor Bodies.....	185
Values as Members.....	187
Getters and Setters .....	188
Additional Constructors.....	189
Explicit Getters and Setters.....	190
Internal Mutable State .....	192
Generic Classes .....	193
Named Parameters and Object Initializer Syntax.....	196
Indexed Properties .....	198
Interfaces .....	202
Object Expressions.....	207



TABLE OF CONTENTS

- Abstract Classes ..... 210
  - Abstract Members ..... 211
  - Default Member Implementations ..... 211
- Class Equality and Comparison ..... 213
  - Implementing Equality ..... 213
  - Implementing Comparison ..... 218
- Recommendations ..... 221
- Summary ..... 222
- Exercises ..... 223
- Exercise Solutions ..... 225
- Chapter 9: Programming with Functions ..... 229**
  - Functions First ..... 229
  - Functions as Values ..... 229
  - Currying and Partial Application ..... 231
  - Mixing Tupled and Curried Styles ..... 233
  - Function Signatures Revisited ..... 235
  - Type Hints for Functions ..... 236
  - Functions That Return Functions ..... 238
  - Function Composition ..... 240
  - Recommendations ..... 243
  - Summary ..... 244
  - Exercises ..... 244
  - Exercise Solutions ..... 247
- Chapter 10: Asynchronous and Parallel Programming ..... 251**
  - Ordering Pizza ..... 251
  - A World Without Async ..... 252
  - Running the Synchronous Downloader ..... 258
  - Converting Code to Asynchronous ..... 259
  - Locking Shared Resources ..... 265

Testing Asynchronous Downloads.....	266
Batching.....	267
Throttling.....	271
C# Task versus F# Async.....	274
Recommendations .....	276
Summary.....	277
Exercises.....	278
Exercise Solutions.....	279
<b>Chapter 11: Railway Oriented Programming .....</b>	<b>283</b>
Going Off the Rails .....	283
On the Factory Floor.....	284
Adapting Functions for Failure.....	288
Writing a Bypass Adapter.....	289
Writing a Pass-Through Adapter .....	290
Building the Production Line.....	290
Making It Official.....	295
Love Your Errors.....	296
Recommendations .....	300
Summary.....	301
Exercises.....	302
Exercise Solutions.....	305
<b>Chapter 12: Performance.....</b>	<b>309</b>
Design Is Compromise .....	309
Some Case Studies .....	310
BenchmarkDotNet.....	310
Case Study: Inappropriate Collection Types .....	312
Avoiding Indexed Access to Lists .....	315
Using Arrays Instead of Lists .....	317
Use Sequences Instead of Arrays .....	318

TABLE OF CONTENTS

- Avoiding Collection Functions ..... 319
- Avoiding Loops Having Skips..... 320
- Inappropriate Collection Types – Summary ..... 322
- Case Study: Short-Term Objects ..... 324
  - Sequences Instead of Arrays ..... 327
  - Avoiding Object Creation ..... 328
  - Reducing Tuples ..... 329
  - Using Struct Tuples..... 330
  - Operator Choice..... 332
  - Short-Term Objects – Summary ..... 334
- Case Study: Naive String Building..... 336
  - StringBuilder to the Rescue..... 338
  - Using String.Join ..... 339
  - Using Array.Parallel.map..... 340
  - Naive String Building – Summary ..... 342
- Other Common Performance Issues ..... 343
  - Searching Large Collections..... 343
  - Comparison Operators and DateTimes ..... 343
  - Concatenating Lists..... 343
  - For Loop with Unexpected List Creation..... 343
- F# 4.5 and Span Support ..... 344
- The Importance of Tests..... 344
- Recommendations ..... 346
- Summary..... 347
- Exercises..... 347
- Exercise Solutions..... 349
- Chapter 13: Layout and Naming ..... 351**
  - Where Are My Braces?..... 351
  - It's Okay Pluto, I'm Not a Planet Either..... 352
  - Some Infelicitous Code ..... 354

Convenience Functions.....	358
Column Extraction Functions.....	359
The Observation Range Type.....	361
The Importance of Alignment.....	364
The Minor Planet Type.....	366
Recommendations.....	373
Summary.....	374
Exercise.....	375
Exercise Solution.....	375
<b>Chapter 14: Summary.....</b>	<b>377</b>
F# and the Sense of Style.....	377
Designing Functions with Types.....	377
Missing Data.....	378
Collection Functions.....	378
Immutability and Mutation.....	379
Pattern Matching.....	379
Record Types.....	379
Classes.....	380
Programming with Functions.....	380
Asynchronous and Parallel Programming.....	381
Railway Oriented Programming.....	382
Performance.....	382
Layout and Naming.....	383
Onwards!.....	383
<b>Index.....</b>	<b>385</b>

# About the Author

**Kit Eason** is a software developer and educator with more than 20 years of experience. He has been programming in F# since 2011 and is employed at Perpetuum Ltd., working on an extensive network of energy-harvesting vibration sensors fitted to railway rolling stock. Kit is an avid F# user who is passionate about teaching others. He has contributed to several publications, as well as to the books *Beginning F# 4.0* (Apress 2016) and *F# Deep Dives* (Manning 2014). He often teaches and presents on F#, and his popular videos appear on Lynda.com and YouTube.

# About the Technical Reviewer

**Quinton Coetzee** was born and raised in a small town not too far from Johannesburg, South Africa, which means he can just about remember how to speak Afrikaans. He played paintball at a relatively high level (for a South African team), traveling to the USA and Europe to compete, before moving to the UK in 2011 to pursue career opportunities in London. Since then, he has worked on real-time trading systems and various applications in one way or another related to trading, primarily in F#. As a productivity tool, F# is really hard to beat and would be his first choice in most production environments. That said, he enjoys playing with other functional languages like Clojure and Scala in his spare time.

# Acknowledgments

I am grateful for the generous help I received in putting *Stylish F#* together. Thanks to Quinton Coetzee for his exceedingly diligent and constructive technical reviews. To Val Eason for reading every chapter before submission, detecting many typos and poor turns of phrase. To Jon Harrop for providing detailed technical feedback on Chapter 12, and to several other F# community members who reviewed a code sample for Chapter 8. To Jason Heeris for kindly giving permission to reproduce the cartoon in Chapter 1. To Don Syme and the F# community for the never-ending stream of compiler and tooling improvements that propel F# forward. And to Matt Jones and the amazing team at Perpetuum for providing the best working environment I've ever experienced. Thanks also to the tireless crew at Apress: Joan Murray, Jill Balzano, and Laura Berendson.

Any errors, omissions, or plain wrong-headedness are, of course, still my own responsibility.

# Introduction

There are three distinct philosophies that you can apply to computer programming. You can think of programming as a *science*, where the measure of progress is how well you discover and reflect fundamental mathematical concepts in your code. You can think of it as a *discipline*, where you seek to establish and follow rules about how code should be written and organized. Or, best of all, you can think of it as a *craft*, where, yes, you apply some of the science and some of the discipline; but you leaven those with a generous helping of human creativity. To do this successfully, you need a fair bit of experience, because crafting something is an inherently intuitive process. This book aims to get you to a level where you can craft code confidently. It does this by distilling and passing on my own experience of writing F# systems in numerous different industries over the past eight years.

Before you start this book, you'll need at least some knowledge of F# syntax and concepts. Maybe you've read some of the wide range of beginner material that's available, and probably you'll have written at least a few simple F# programs yourself. You may well have deeper experience of other languages and environments, such as C# and .NET. That said, I have framed the book so that C# knowledge is not a hard prerequisite: I learned F# before I learned C#, and if I can do it, so can you! Also you definitely don't need any background in Computer Science or functional programming. I don't have even a trace of formal education in either of these areas.

So what's between the covers? In Chapter 1, I'll establish some *principles* that will help us decide whether we are coding well, and say a little bit about why coding stylishly is important. In Chapter 2, we'll pick up the basic tools of our craft and learn to chisel out elegant and reliable *functions*. In Chapter 3, we'll tackle the thorny issue of *missing data*, learning some effective techniques for writing dependable code when certain values might not be available. In Chapter 4, we'll pick up some more powerful crafting tools: the so-called *collection functions*, and explore how you can use them to achieve a surprising amount with very little code. In Chapter 5, we'll delve into the strange world of *immutability*: how you can write programs that achieve a result without explicitly changing anything. In Chapter 6, we'll look at *pattern matching*, a concept you may have looked at a little when you learned F# syntax, but which is surprisingly pervasive and



## INTRODUCTION

powerful in quality F# code. In Chapter 7, we'll explore *record types*, F#'s go-to structure for storing groups of labeled values. In Chapter 8, we'll cover some ground that might already be familiar to C# developers: *object oriented classes*. In Chapter 9, we'll return to the topic of F# *functions*, and explore what it means for a function to also be a first-class value. In Chapter 10, we'll tame the apparent complexity of *asynchronous and parallel programming*: it needn't be as hard as you think! In Chapter 11, we'll look at *Railway Oriented Programming*, an interesting metaphor you can use to help you think about processing pipelines. In Chapter 12, we'll investigate *performance*: can you really write code that is both elegant and fast? In Chapter 13, we'll establish some useful techniques for *laying out* your code and *naming* items to maximize readability. In Chapter 14, I'll briefly reiterate what we've learned.

As this book is primarily about the language, you'll find relatively few references to other libraries. Of course, to build substantial systems, you'll almost always want to pull in NuGet packages for requirements such as unit testing, serialization, web serving, and so forth. But these libraries constitute a large and fast-changing landscape, so I've chosen to pare things down to the F# essentials for this book. This also means that almost all the examples can be typed in and simply run as F# scripts, and they are provided in script form in the downloadable code samples. In the small number of cases where you need to write a compilable program, I take you through the process alongside the example.

Likewise, you won't find many references to specific integrated development environments (IDEs) such as Visual Studio, Visual Studio Code, Xamarin Studio, or JetBrains Rider. Any of these can be used to edit and run the examples in this book, and all are available as free editions if you don't already have something installed. If your IDE doesn't know about F# "out of the box," simply search online for "F# <your IDE> getting started" to find setup instructions. The samples should work without change on any platform where F# is installed, except you may need to change some paths where the sample code accesses local files.

I very much hope you enjoy sharing my F# experience as much as I enjoyed acquiring it. Don't forget to have fun!