objc ↑↓ Functional Swift

Updated for Swift 4

By Chris Eidhof, Florian Kugler and Wouter Swiersta

Functional Swift

Chris Eidhof

Florian Kugler

Wouter Swierstra

objc.io

Functional Swift

1. Introduction
2. Thinking Functionally
    1. Example: Battleship
    2. First-Class Functions
    3. Type-Driven Development
    4. Notes
3. Case Study: Wrapping Core Image
    1. The Filter Type
    2. Theoretical Background: Currying
    3. Discussion
4. Map, Filter, Reduce
    1. Introducing Generics
    2. Filter
    3. Reduce
    4. Putting It All Together
    5. Generics vs. the Any Type
    6. Notes
5. Optionals
    1. Case Study: Dictionaries
    2. Working with Optionals
    3. Why Optionals?
6. Case Study: QuickCheck
    1. Building QuickCheck
    2. Making Values Smaller

7. The Value of Immutability
    1. Variables and References
    2. Value Types vs. Reference Types
    3. Discussion
8. Enumerations
    1. Introducing Enumerations
    2. Associated Values
    3. Adding Generics
    4. Swift Errors
    5. Optionals Revisited
    6. The Algebra of Data Types
    7. Why Use Enumerations?
9. Purely Functional Data Structures
    1. Binary Search Trees
    2. Autocompletion Using Tries
    3. Discussion
10. Case Study: Diagrams
    1. Drawing Squares and Circles
    2. The Core Data Structures
    3. Discussion
11. Iterators and Sequences
    1. Iterators
    2. Sequences
    3. Case Study: Traversing a Binary Tree
    4. Case Study: Better Shrinking in QuickCheck
12. Case Study: Parser Combinators
    1. The Parser Type
    2. Combining Parsers
    3. Parsing Arithmetic Expressions
    4. A Swifty Alternative for the Parser Type

13. Case Study: Building a Spreadsheet Application
    1. Parsing
    2. Evaluation
    3. User Interface
14. Functors, Applicative Functors, and Monads
    1. Functors
    2. Applicative Functors
    3. The M-Word
    4. Discussion
15. Conclusion
    1. Further Reading
    2. Closure
    3. Bibliography

Introduction

Why write this book? There's plenty of documentation on Swift readily available from Apple, and there are many more books on the way. Why does the world need yet another book on yet another programming language?

This book tries to teach you to think *functionally*. We believe that Swift has the right language features to teach you how to write *functional programs*. But what makes a program functional? And why bother learning about this in the first place?

It's hard to give a precise definition of functional programming — in the same way, it's hard to give a precise definition of object-oriented programming, or any other programming paradigm for that matter. Instead, we'll try to focus on some of the *qualities* that we believe well-designed functional programs in Swift should exhibit:

- Modularity: Rather than thinking of a program as a sequence of assignments and method calls, functional programmers emphasize that each program can be repeatedly broken into smaller and smaller pieces, and all these pieces can be assembled using function application to define a complete program. Of course, this decomposition of a large program into smaller pieces only works if we can avoid sharing state between the individual components. This brings us to our next point.

- A Careful Treatment of Mutable State: Functional programming is sometimes (half-jokingly) referred to as 'value-oriented programming.' Object-oriented programming focuses on the design of classes and objects, each with their own encapsulated state. Functional programming, on the other hand, emphasizes the importance of programming with values, free of mutable state or other side effects. By avoiding mutable state, functional

programs can be more easily combined than their imperative or object-oriented counterparts.

- Types: Finally, a well-designed functional program makes careful use of *types*. More than anything else, a careful choice of the types of your data and functions will help structure your code. Swift has a powerful type system that, when used effectively, can make your code both safer and more robust.

We feel these are the key insights that Swift programmers may learn from the functional programming community. Throughout this book, we'll illustrate each of these points with many examples and case studies.

In our experience, learning to think functionally isn't easy. It challenges the way we've been trained to decompose problems. For programmers who are used to writing for loops, recursion can be confusing; the lack of assignment statements and global state is crippling; and closures, generics, higher-order functions, and monads are just plain weird.

Throughout this book, we'll assume that you have previous programming experience in Objective-C (or some other object-oriented language). We won't cover Swift basics or teach you to set up your first Xcode project, but we will try to refer to existing Apple documentation when appropriate. You should be comfortable reading Swift programs and familiar with common programming concepts, such as classes, methods, and variables. If you've only just started to learn to program, this may not be the right book for you.

In this book, we want to demystify functional programming and dispel some of the prejudices people may have against it. You don't need to have a PhD in mathematics to use these ideas to improve your code! Functional programming isn't the *only* way to program in Swift. Instead, we believe that learning about functional programming adds an important new tool to your toolbox, which will make you a better developer in any language.

Updates to the Book

As Swift evolves, we'll continue to make updates and enhancements to this book. Should you encounter any mistakes, or if you'd like to send any other kind of feedback our way, please file an issue in our GitHub repository.

Acknowledgements

We'd like to thank the numerous people who helped shape this book. We wanted to explicitly mention some of them:

Natalye Childress is our copy editor. She has provided invaluable feedback, not only making sure the language is correct and consistent, but also making sure things are understandable.

Sarah Lincoln designed the cover and the layout of the book.

Wouter would like to thank *Utrecht University* for letting him take time to work on this book.

We'd also like to thank the beta readers for their feedback during the writing of this book (listed in alphabetical order):

Adrian Kosmaczewski, Alexander Altman, Andrew Halls, Bang Jun-young, Daniel Eggert, Daniel Steinberg, David Hart, David Owens II, Eugene Dorfman, f-dz-v, Henry Stamerjohann, J Bucaran, Jamie Forrest, Jaromir Siska, Jason Larsen, Jesse Armand, John Gallagher, Kaan Dedeoglu, Kare Morstol, Kiel Gillard, Kristopher Johnson, Matteo Piombo, Nicholas Outram, Ole Begemann, Rob Napier, Ronald Mannak, Sam Isaacson, Ssu Jen Lu, Stephen Horne, TJ, Terry Lewis, Tim Brooks, Vadim Shpakovski.

Chris, Florian, and Wouter